Gathering despite a linear number of weakly Byzantine agents

Jion Hirose*, Junya Nakamura†, Fukuhito Ooshita‡, and Michiko Inoue*
*Nara Institute of Science and Technology, Ikoma, Japan
Email: {hirose.jion.hg4, konoue}@is.naist.jp
†Toyohashi University of Technology, Toyohashi, Japan
Email: junya@imc.tut.ac.jp
‡Fukui University of Technology, Fukui, Japan
Email: f-oosita@fukui-ut.ac.jp

***Abstract*—We study the gathering problem to make multiple agents initially scattered in arbitrary networks gather at a single node. There exist k agents with unique identifiers (IDs) in the network, and f of them are weakly Byzantine agents, which behave arbitrarily except falsifying their identifiers. The agents behave in synchronous rounds, and each node does not have any memory like a whiteboard. In the literature, two algorithms for solving the gathering problem have been proposed. The first algorithm assumes that the number n of nodes is given to agents and achieves the gathering in $O(n^4 \cdot |\Lambda_{good}| \cdot X(n))$ rounds, 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. The second algorithm assumes that the upper bound N of n is given to agents and at least $4f^2 + 8f + 4$ non-Byzantine agents exist, and achieves the gathering in $O((f + |\Lambda_{good}|) \cdot X(N))$ rounds. Both the algorithms allow agents to start gathering at different times. The first algorithm can terminate agents simultaneously, while the second one not. In this paper, we seek an algorithm that solves the gathering problem efficiently with the intermediate number of non-Byzantine agents since there is a large gap between the numbers of non-Byzantine agents in the previous works. The resultant gathering algorithm works with at least $8f + 8$ non-Byzantine agents when agents start the algorithm at the same time, agents may terminate at different times, and N is given to agents. To reduce the number of agents, we propose a new technique to simulate a Byzantine consensus algorithm for synchronous message-passing systems on agent systems. The proposed algorithm achieves the gathering in $O(f \cdot |\Lambda_{good}| \cdot X(N))$ rounds. This algorithm is faster than the first existing algorithm and requires fewer non-Byzantine agents than the second existing algorithm if n is given to agents, although the guarantees on simultaneous termination and startup delay are not the same.**

***Index Terms*—Distributed algorithms, Byzantine environments, Gathering**

I. Introduction

A. Background

Mobile agents (in short, agents) are software programs that can move autonomously in a distributed system. A problem to make multiple agents initially scattered in the system meet at a single node is called *gathering*. This problem is fundamental to various cooperative behavior of agents [1] and allows the agents to share information and plan for future tasks efficiently.

Since agents are software programs, they are exposed to bugs, cracking, and other threats. Thus, as the number of agents increases, it is inevitable that some of those agents become faulty. Among various faults of agents, Byzantine faults are known to be the most severe because we have no control over the behavior of the faulty agents (called Byzantine agents). For example, Byzantine agents can stay at the current node, move to a neighbor node, and convey arbitrary information to other agents, ignoring their algorithms.

In this paper, we consider the gathering problem in the presence of Byzantine agents and propose a deterministic synchronous gathering algorithm to solve this problem.

B. Related Works

The gathering problem has widely been studied in the literature. When the number of agents is two, the gathering problem is called the rendezvous problem. Those studies assume the gathering problem in various environments, which is a combination of the assumptions (e.g., agent synchronization, anonymity, presence/absence of memory on a node (called whiteboard), presence/absence of randomization, topology). Then, those studies have clarified the solvability and, if solvable, they have analyzed its cost (e.g., time, the number of moves, memory space, etc.). Pelc [1] has extensively surveyed deterministic rendezvous problems under the various assumptions. Also, Alpern and Gal [4] have described an extensive survey of randomized rendezvous problems under the various assumptions.

Recently many studies have considered the gathering problem in the presence of Byzantine agents [2], [3], [5]–[9]. These studies consider two types of Byzantine agents, weakly and strongly ones. While weakly Byzantine agents can behave arbitrarily except falsifying their own IDs, strongly ones can behave arbitrarily including falsifying their own IDs. Table I summarizes this study and the related studies in the presence of weakly Byzantine agents. Note that the assumption of startup delay in Table I means agents may start an algorithm at different times but agents can wake up sleeping agents at the visited node.

Dieudonné et al. [2] introduced the gathering problem in synchronous environments in the presence of weakly Byzantine agents. They provided two gathering algorithms under the assumption that k agents exist in an arbitrary network

TABLE I

A SUMMARY OF SYNCHRONOUS GATHERING ALGORITHMS IN THE PRESENCE OF WEAKLY BYZANTINE AGENTS ASSUMING THAT AGENTS HAVE UNIQUE IDS. HERE, INPUT IS THE INFORMATION THAT IS INITIALLY GIVEN TO ALL AGENT, n IS THE NUMBER OF NODES, N IS THE UPPER BOUND OF n, $X(n)$ IS THE NUMBER OF ROUNDS REQUIRED TO EXPLORE ANY NETWORK COMPOSED OF n NODES, $|\Lambda_{good}|$ IS THE LENGTH OF THE LARGEST ID AMONG NON-BYZANTINE AGENTS, $|\Lambda_{all}|$ IS THE LENGTH OF THE LARGEST ID AMONG AGENTS, k IS THE NUMBER OF AGENTS, f IS THE NUMBER OF BYZANTINE AGENTS, AND F IS THE UPPER BOUND OF f.

	Input	Condition of #Byzantine agents	Startup delay	Simultaneous termination	Time complexity
[2]	n	$f+1 \leq k$	Yes	Yes	$O(n^4 \cdot |\Lambda_{good}| \cdot X(n))$
[2]	F	$2F+2 \leq k$	Yes	Yes	Poly. of n & $|\Lambda_{good}|$
[3]	N	$4f^2+9f+4 \leq k$	Yes	No	$O((f+|\Lambda_{good}|) \cdot X(N))$
[3]	N	$4f^2+9f+4 \leq k$	Yes	Yes	$O((f+|\Lambda_{all}|) \cdot X(N))$
This	N	$9f+8 \leq k$	No	No	$O(f \cdot |\Lambda_{good}| \cdot X(N))$

composed of n nodes and at most F of them are Byzantine agents. The first algorithm solves the gathering problem in $O(n^4 \cdot |\Lambda_{good}| \cdot X(n))$ rounds if $k \geq f+1$ holds (i.e., at least one non-Byzantine agent exists) and n is given to agents, where f is the number of Byzantine agents, $|\Lambda_{good}|$ is the length of the largest ID among non-Byzantine agents, and $X(n)$ is the number of rounds required to explore any network composed of n nodes. The second algorithm achieves the gathering in polynomial time of $|\Lambda_{good}|$ and $X(n)$ if $k \geq 2F+2$ holds (i.e., at least $F+2$ non-Byzantine agents exist) and F is given to agents. The numbers of non-Byzantine agents in these algorithms match the lower bounds under the assumptions. Hirose et al. [3] provided the two gathering algorithms with lower time complexity by assuming $\Omega(f^2)$ non-Byzantine agents. If the upper bound N of n is given to agents and $k \geq 4f^2+9f+4$ holds (i.e., at least $4f^2+8f+4$ non-Byzantine agents exist), the first algorithm achieves the gathering with non-simultaneous termination in $O((f+|\Lambda_{good}|) \cdot X(N))$ rounds, and the second one 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. Tsuchida et al. [8] reduced the time complexity using authenticated whiteboards (i.e., each agent has a dedicated area for each node and can leave the information on its area using its ID). Their algorithm achieves the gathering in $O(Fm)$ rounds if F is given to agents and $F < k$ holds, where m is the number of edges. To efficiently achieve the gathering, the authors proposed a technique for agents to simulate a consensus algorithm for Byzantine message-passing systems. However, this technique requires each node to have an authenticated whiteboard. Tsuchida et al. [9] also proposed a gathering algorithm in asynchronous environments with authenticated whiteboards.

Dieudonné et al. [2] also introduced the gathering problem in synchronous environments in the presence of strongly Byzantine agents for the first time and have provided two gathering algorithms under different assumptions. The first algorithm solves the gathering problem in exponential time of $|\Lambda_{good}|$ and $X(n)$ if $k \geq 3F+1$ holds (i.e., at least $2F+1$ non-Byzantine agents exist) and n and F are given to agents. The second algorithm achieves the gathering in exponential time of $|\Lambda_{good}|$ and $X(n)$ if $k \geq 5F+2$ holds (i.e., at least $4F+2$ non-Byzantine agents exist) and F is given to agents. On the other hand, the lower bounds on the number of non-Byzantine agents required to solve the gathering problems under these assumptions are $F+1$ and $F+2$, respectively. Bouchard et al. [5] have provided two algorithms using the number of non-Byzantine agents that match the lower bounds on the gathering problems under these assumptions. Bouchard et al. [6] reduced the time complexity to polynomial time complexity by assuming that $\Omega(f^2)$ non-Byzantine agents exist. Their algorithm assume that $\lceil \log \log n \rceil$ is given to agents and $k \geq 5f^2+7f+2$ holds (i.e., at least $5f^2+6f+2$ non-Byzantine agents exist), and achieves the gathering in polynomial time of n and $|\lambda_{good}|$, where $|\lambda_{good}|$ is the length of the smallest ID among non-Byzantine agents, and f is the number of Byzantine agents. Miller et al. [7] have proposed the gathering algorithm in small time complexity by additional assumption. They assume that $k \geq 2f+1$ holds (i.e., $f+1$ non-Byzantine agents exist) and an agent can get the subgraph induced by nodes within distance D_r from its current node and the state of agents in the subgraph, where D_r is the radius of the graph. Their algorithm achieves the gathering in $O(kn^2)$ rounds.

C. Contribution

We provide an efficient algorithm that achieves the gathering with non-simultaneous termination in synchronous environments where $\Omega(k)$ weakly Byzantine agents exist. Due to space limitations, we omit the details of the proposed algorithm and give them in the technical report [10]. In the literature, the algorithm by Dieudonné et al. [2] tolerates any number of Byzantine agents with high time complexity, while the algorithm by Hirose et al. [3] works with low time complexity restricting the number of Byzantine agents as $4f^2+9f+4 \leq k$. There is a large gap between the assumptions of the number of agents in the above two works, and thus it is natural to consider a gathering algorithm under the intermediate assumption. The proposed algorithm achieves the gathering with non-simultaneous termination when N is given to agents, at least $9f+8$ agents exist in the network, and agents start the algorithm at the same time. This algorithm is faster than that of Dieudonné et al. [2] and requires fewer non-Byzantine agents than that of Hirose et al. [3] if n is given to agents, although the guarantees on simultaneous termination and startup delay are not the same. To solve the gathering, we propose a new technique to simulate a Byzantine consensus algorithm proposed by Khanchandani et al. [11] for a synchronous message-passing system on an agent system, where one agent imitates one process. This technique can be used not only for Byzantine gatherings but also for other problems.

II. Preliminaries

A. Agent Model

Agent 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. We define $d(v)$ as the degree of node v. Each incident edge of node v is assigned a locally-unique port number in $\{1, \ldots, d(v)\}$. That is, on node v, the port number of edge (v, u) is different from that of edge (v, w) for node $w \neq u$. Nodes do not have IDs or memories.

We denote by $MA = \{a_1, a_2, \ldots, a_k\}$ the set of k agents. Each agent $a_i \in MA$ has an unique ID denoted by $a_i.id \in \mathbb{N}$ and has an infinite amount of memory. Agents know the upper bound N of the number of nodes, but they know neither n, k, nor the IDs of other agents. Agents cannot mark visited nodes or traversed edges in any way. An agent is modeled as a state machine (S, δ), where S is a set of agent states and δ is a state transition function. A state is represented by a tuple of the values of all the variables that an agent has. Each agent has a special state that indicates the termination of an algorithm, called a terminal state. If an agent transitions into a terminal state, it never moves or updates its state after that.

All agents start an algorithm at the same time, and the initial nodes of the agents are chosen by an adversary. All agents repeatedly and synchronously execute a round. In each round, every agent a_i executes the following operations:

- **Look**: Agent a_i learns the state of a_i, the degree $d(u)$ of the current node u, and the port number i of the edge through which the agent arrived at node u (or a_i notices that u is an initial node). Also, if multiple agents exist at node u, a_i learns the states of all agents at node u, including agents in a terminal state. We define $A_i \subseteq MA$ as the set of agents existing at node u and a_i.
- **Compute**: Agent a_i computes function δ using the information obtained in the previous Look operation as input. The output is the next agent state, whether it stays or leaves, and the outgoing port number if it leaves.
- **Move**: If a_i decides to stay, it stays at the current node until the beginning of the next round. If a_i decides to leave, it leaves through the decided outgoing port number and arrives at the destination node before the beginning of the next round. If two agents traverse the same edge in different directions at the same time, the agents cannot notice this fact.

There are f weakly Byzantine agents, which act arbitrarily apart from an algorithm, except changing their IDs. If multiple agents meet Byzantine agents at the same node, all of them learn the same statuses of the Byzantine agents in the Look operation. We call all agents except weakly Byzantine agents *good* agents and denote by $g = k - f$ the number of good agents. Good agents know neither the actual number f of Byzantine agents nor the upper bound of f.

B. Gathering Problem

The gathering problem requires all good agents to transition into the terminal state at the same node. Note that this definition does not require agents to enter a terminal state at the same time. We measure the time complexity of a gathering algorithm by the number of rounds required for the last good agent to transition into the terminal state.

C. Rendezvous Procedure

The rendezvous procedure allows two different agents to meet at the same node in any connected graph composed of at most N nodes if each of them gives a different ID and N as inputs to the procedure. The procedure is a well-known combination of an ID transformation procedure and an exploration procedure, the former is proposed by Dessmark et al. [12], and the latter is based on universal exploration sequences (UXS) and is a corollary of the result of Ta-Shma et al. [13]. We call this procedure $\texttt{REL}(id)$, where id is an ID given as input, and the execution time $t_{REL}(id)$ of $\texttt{REL}(id)$ is $O(N^5 \log(N) \log(id))$ rounds. This procedure guarantees that, if two agents with distinct IDs l_1 and l_2 start $\texttt{REL}(l_1)$ and $\texttt{REL}(l_2)$ in round r_1 and r_2, respectively, they meet at the same node before rounds $\max(r_1, r_2) + t_{REL}(\min(l_1, l_2))$.

D. Parallel Byzantine Consensus Algorithm for Synchronous Message-Passing Systems

The proposed algorithm uses the parallel Byzantine consensus algorithm [11] for synchronous message-passing systems, which is called $\texttt{PCONS}(S)$ where S is a set given as input, by simulating the algorithm on agent systems. This algorithm assumes mostly the well-known synchronous message-passing model; thus, we mention only rare assumptions of the algorithm here. In the model, each agent does not know the number m of nodes, the number b of weakly Byzantine nodes, or the IDs of other nodes. However, every message has the ID of its sender; thus, when a node receives a message, it can know the sender's ID. Node v can send a message *msg* in two ways: (1) v broadcasts *msg* to all nodes, or (2) v sends *msg* to a specific node that v knows its ID. There is no restriction on the actions of Byzantine nodes except for falsifying their IDs to a directly communicating node.

Each good node v has a set S_v composed of k_v input pairs (id_v^i, x_v^i) $(1 \leq i \leq k_v)$, where id_v^i is an ID of the input pair and x_v^i is an input number. We say an algorithm solves the parallel Byzantine consensus problem if, when each good node v starts with set S_v as an input, each node outputs a set of pairs subject to the following conditions called PBC property:

- **Validity 1** If (id, x) is an input pair of every good node and $x \neq \perp$, the output set of a good node includes (id, x).
- **Validity 2** If (id, x) is not an input pair of any good node, the output set of any good node does not include (id, x).
- **Agreement** If the output set of a good node includes (id, x), then the output sets of all other good nodes must include (id, x) as well.

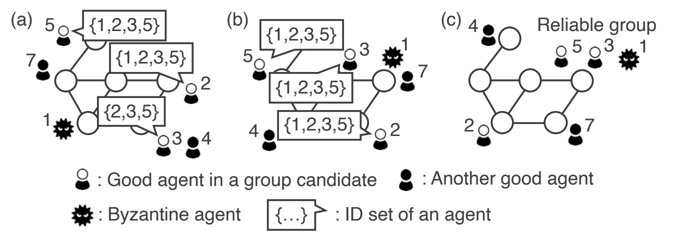


Fig. 1. The reliable group creation strategy of the proposed algorithm. Each figure corresponds to Steps (a) to (c) in Section III-A. Agents with IDs 1, 2, 3, and 5 belong to the same group candidate.

- **Termination** Every good node outputs a set of pairs exactly once in a finite number of phases.

Lemma 1 ([11]). *Assume that more than $3b$ nodes exist in a system. If every good node v simultaneously starts $PCONS(S_v)$ with a set S_v as input, its execution satisfies the PBC property. Every good node outputs a set in $O(b)$ phases, and its output time differs by at most one phase among good nodes.*

III. Byzantine Gathering Algorithm

Throughout the paper, we assume $k = g + f \geq 9f + 8$, which implies that there are at least $8f + 8$ good agents in the network. Recall that agents know N, but do not know n, k, or f.

A. Overview

We give the overview of the proposed Byzantine gathering algorithm, which aims to gather all good agents at a single node. For simplicity, we assume that agents know f here, and will remove this assumption later. Due to page limitation, we explain the details in the full paper [10]. The underlying idea of the algorithm is made of the following three steps:

(1) All agents collect all agent IDs using the rendezvous algorithm REL.
(2) All agents decide a common ID as a target ID.
(3) An agent a_{target} with the target ID stays at the current node and the other agents search for a_{target} using REL.

If there are no Byzantine agents, all agents can decide a common target ID by choosing the smallest ID of the collected IDs and gather at the node where the agent with the smallest ID exists. However, if there is a Byzantine agent, that idea fails. Assume that a Byzantine agent $Byz \in MA$ has the smallest ID. If Byz meets only a part of good agents in Step (1), the other good agents do not choose $Byz.id$ as a target ID. Therefore, good agents are divided into two or more groups. Even if all good agents know $Byz.id$, Byz can avoid meeting the other agents in Step (3), and all good agents keep searching endlessly for a_{target}.

To solve these problems, the proposed algorithm suppresses the influence of Byzantine agents by letting several agents create a reliable group such that good agents can trust the behavior of the group. After collecting all good agents' IDs, agents execute the following three steps as shown in Fig. 1:

(a) Agents create a group candidate of at least $3f+1$ agents.
(b) Agents in the group candidate make a common ID set by using the parallel Byzantine consensus algorithm.
(c) By using the common ID set, agents in the group candidate gather to create a reliable group composed of at least $f + 1$ good agents.

The goal of Steps (a) and (b) is to make at least $3f + 1$ agents make a common ID set. To do this, we use the parallel Byzantine consensus algorithm in Section II-D. Since the consensus algorithm assumes message-passing systems, agents simulate the system by using rendezvous algorithm REL. Simply put, agents exchange messages when they meet other agents using REL. In Step (a), agents create a group candidate of at least $3f+1$ agents such that they can exchange messages with each other among the group candidate. In Step (b), as an input of the consensus algorithm, each agent uses the set of agent IDs (known to the agent) in the same group candidate. If the group candidate is composed of at least $3f+1$ agents, the output is common and includes IDs of all good agents in the group candidate. In Step (c), agents in a group candidate decide a target ID based on the common ID set and gather at the node where an agent with the target ID exists. If at least $2f + 1$ agents gather, they create a reliable group composed of at least $f+1$ good agents. Otherwise, the agents determine the next target ID and find the agent with the new target ID. The algorithm ensures that all good agents in the group candidate eventually gather and create a reliable group.

Once at least one reliable group is created, the proposed algorithm can achieve the gathering as follows. Good agents in the group decide the smallest agent ID in the group as a group ID and execute REL using the group ID. On the other hand, good agents not in the reliable group execute REL using their own IDs. When a good agent meets a reliable group with a smaller group ID, it accompanies the group. All agents in the reliable group are at the same node, act identically, and have the same group ID. If a good agent meets the reliable group, the agent trusts the group since the group consists of at least $f + 1$ agents with the same group ID, which implies that the group contains at least one good agent. As a result, all good agents accompany the reliable group with the smallest ID and achieve the gathering.

The key to the algorithm is the reliable group creation procedure. The existing algorithm of Hirose et al. [3] employs another strategy to create a reliable group after collecting IDs. In the algorithm, each good agent searches for one of the agents with the smallest $f+1$ IDs of the collected IDs. Because good agents may be divided into $\Omega(f)$ nodes, this strategy requires at least $\Omega(f^2)$ good agents to guarantee that a reliable group is created at some of those nodes. On the other hand, our proposed algorithm uses the strategy such that $\Omega(f)$ good agents make a common ID set and search for a target agent one by one synchronously. This strategy requires $\Omega(f)$ good agents, which reduces the number of good agents required to achieve the gathering.

B. Strategy to Create a Reliable Group

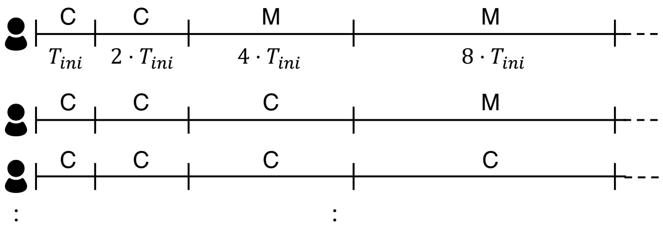
In this section, we explain a strategy to create a reliable group, called MakeReliableGroup. We give the overview of the strategy while assuming that agents know f, and then discuss the way to remove this assumption at the end of this section.

1) Overview: As mentioned in Section III-A, in MakeReliableGroup, agents in the group candidate make a common ID set and search for agents with target IDs. Algorithm MakeReliableGroup uses the parallel Byzantine consensus algorithm PCONS for at least $2f+1$ good agents to have a common ID set. However, since PCONS assumes a synchronous message-passing model, we cannot use PCONS directly. Therefore, MakeReliableGroup simulates the model on an agent system as follows.

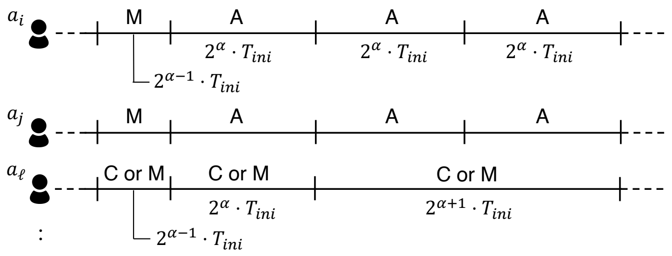
First, MakeReliableGroup selects a group candidate composed of at least $3f+1$ agents and finds time T that is sufficiently long to meet all the agents in the group candidate by REL. Then, agents regard the time interval of T rounds as a phase in the message-passing model, and simulate the behavior of the phase during the T rounds. More concretely, each agent a_i in the group candidate executes $\texttt{REL}(a_i.id)$ for T rounds and, if a_i meets another agent a_j in the group candidate, a_i shares a message that a_i sent in the previous phase. Then, in the last round of the interval, a_i executes the computation of the phase using the messages collected by the current phase. Since a_i can meet all good agents in the group candidate during the T rounds, this behavior can simulate the broadcast of messages in the synchronous message-passing model.

This simulation requires agents to (1) construct a group candidate composed of at least $3f+1$ agents, (2) start PCONS simultaneously with the agents in the same group candidate, and (3) know time T that is long enough to meet the agents in the same group candidate. To meet the requirements, agents synchronously repeat a *cycle*. Each agent sets the length of the first cycle as T_{ini} rounds, where T_{ini} is a given positive integer, and doubles the length every cycle, i.e., that of the second cycle is $2 \cdot T_{ini}$ rounds, that of the third cycle is $4 \cdot T_{ini}$ rounds, and so on. If the length of a cycle is long enough for agent a_i to meet all other good agents by $\texttt{REL}(a_i.id)$, a_i starts $\texttt{REL}(a_i.id)$ for the cycle. If the length of a cycle is long enough for at least $3f+1$ agents to meet each other, MakeReliableGroup regards them as a group candidate and makes them start PCONS simultaneously. By this behavior, MakeReliableGroup can achieve both (1) and (2). Furthermore, since the length of the cycle is long enough for at least $3f+1$ agents to meet each other, MakeReliableGroup can achieve (3) by defining T as the length of the cycle when the agents start PCONS.

Algorithm MakeReliableGroup consists of four stages: *CollectID*, *MakeCandidate*, *AgreeID*, and *MakeGroup* stages. In the *CollectID* stage, agents collect IDs of all good agents. In the *MakeCandidate* stage, agents select the group candidate. In the *AgreeID* stage, agents in the group candidate obtain a common ID set by using consensus algorithm PCONS. In the *MakeGroup* stage, agents create a reliable group. The overall flow of MakeReliableGroup is shown in Fig. 2. Each stage consists of multiple cycles and an agent a_i executes those stages in order. Agents have the same length of the first cycle and double the length every cycle until they start the *AgreeID* stage. Then, if an agent is in the *AgreeID* stage or the *MakeGroup* stage, the agent does not update the length of a cycle. Thus, agents in the *CollectID* stage or the *MakeCandidate* stage start their cycles at the same time.



(a) At starting MakeReliableGroup


a_i
M
A
A
A
$2^{\alpha} \cdot T_{ini}$
$2^{\alpha} \cdot T_{ini}$
$2^{\alpha} \cdot T_{ini}$
$2^{\alpha-1} \cdot T_{ini}$
a_j
M
A
A
A
a_ℓ
C or M
C or M
C or M
$2^{\alpha} \cdot T_{ini}$
$2^{\alpha+1} \cdot T_{ini}$
$2^{\alpha-1} \cdot T_{ini}$

(b) At starting the *AgreeID* stage

Fig. 2. The stage flow of Algorithm MakeReliableGroup. Notions C, M, and A represent cycles of the *CollectID* stage, *MakeCandidate* stage, and the *AgreeID* stage, respectively.

In MakeReliableGroup, an agent a_i can check whether another agent belongs to the same group candidate as follows. Assume that a_i starts the *AgreeID* stage in round r. Let a_j be an agent that also starts the *AgreeID* stage in round r. Before round r, a_i and a_j have the same length of their cycles. Also, an agent does not update the length of a cycle in the *AgreeID* stage or the *MakeGroup* stage. Thus, a_i and a_j always have the same length of their cycles. Hence, when a_i meets a_j at the current node after a_i starts the *AgreeID* stage, a_i can understand that a_j is a member of the same group candidate. On the other hand, let us consider that another good agent a_ℓ starts the *AgreeID* stage in round r' ($r' \neq r$). The total number of cycles that a_ℓ has executed in the *CollectID* stage and the *MakeCandidate* stage is greater or less than that of a_i. The length of a cycle that a_ℓ starts the *AgreeID* stage is different from the length of a cycle that a_i does. Hence, when a_i meets a_ℓ at the current node after a_i starts the *AgreeID* stage, a_i can understand that a_ℓ started the *AgreeID* stage at a different round, that is, a_ℓ is a member of the group candidate different from a_i.

We guarantee that in the case where good agent a_i starts the *AgreeId* stage faster than good agent a_ℓ (shown in Fig. 2b), when a_ℓ starts a cycle, a_i also starts a cycle at the same time.

Since a_i does not update the length of a cycle anymore, the length of each cycle in the *AgreeID* stage and the *MakeGroup* stage is identical. On the other hand, a_ℓ doubles the length of a cycle every cycle until it starts the *AgreeID* stage. This implies that the length of a cycle of a_ℓ is a multiple of the length of a cycle of a_i. Hence, when a_ℓ starts a cycle, a_i also starts a cycle.

2) CollectID stage: In the *CollectID* stage, agents collect IDs of all good agents. To do this, if the length of a cycle is not long enough for a_i to meet all other good agents, a_i stays at the current node. Otherwise, a_i executes $\texttt{REL}(a_i.id)$ throughout the current cycle. When a_i meets another agent a_j while executing $\texttt{REL}(a_i.id)$, a_i records ID of a_j. Once a_i completes $\texttt{REL}(a_i.id)$, a_i moves to the *MakeCandidate* stage in the next cycle. When a_i finishes the *CollectID* stage, the agent has at least the IDs of all good agents.

3) MakeCandidate stage: In the *MakeCandidate* stage, agents select the group candidate. More concretely, agents make at least $3f + 1$ good agents start the *AgreeID* stage at the same time and regard these agents as a group candidate. To do this, each agent a_i maintains variable $a_i.\mathit{ready}$. Initially, $a_i.\mathit{ready} = \mathit{False}$. During the *MakeCandidate* stage, a_i continues to execute $\texttt{REL}(a_i.id)$. At the beginning of a cycle, a_i checks whether at least $8f + 1$ agents have already started the *MakeCandidate* stage, and if it is true, a_i stores *True* in $a_i.\mathit{ready}$. Note that, for an agent a_j, the starting time of the *MakeCandidate* stage of a_j only depends on $a_j.id$, because in the *CollectID* stage a_j executes $\texttt{REL}(a_j.id)$ if the length of the cycle is sufficiently long, and then moves to the *MakeCandidate* stage. Consequently, a_i can make this decision by the set of collected IDs. During the *CollectID* stage and the *MakeCandidate* stage, when a_i meets another agent a_j, a_i also records the value of $a_j.\mathit{ready}$. If a_i has witnessed at least $4f + 1$ agents with $\mathit{ready} = \mathit{True}$ at the beginning of a cycle, a_i also stores *True* in variable *ready*. After that, if a_i has witnessed at least $6f + 1$ agents with $\mathit{ready} = \mathit{True}$ at the beginning of a cycle, a_i starts the *AgreeID* stage from next cycle (after a_i finishes the current cycle).

In the following, we explain why this behavior makes at least $3f + 1$ agents start the *AgreeID* stage at the same time. Since f Byzantine agents exist, if no good agent has stored *True* in *ready*, there exist at most f agents with $\mathit{ready} = \mathit{True}$. Thus, when the first good agent stores *True* in *ready*, it decides that at least $8f + 1$ agents have started the *MakeCandidate* stage. Hence, when good agent a_i stores *True* in *ready*, a_i knows that the length of a cycle is long enough for at least $(4f + 1) - f = 3f + 1$ good agents to meet each other. Moreover, assume that agent a_{ini} is the first good agent that starts the *AgreeID* stage and let c^γ be the γ-th cycle in which a_{ini} does so. In this case, we have the following two facts.

(a) Each good agent in the *MakeCandidate* stage has witnessed at least $(6f + 1) - f = 5f + 1$ good agents with $\mathit{ready} = \mathit{True}$ by the beginning of cycle $c^{\gamma-1}$.

(b) At least $(8f + 1) - f = 7f + 1$ good agents have started the *MakeCandadite* stage by the beginning of cycle $c^{\gamma-1}$.

Let a_j be an arbitrary good agent that has started the *MakeCandadite* stage by the beginning of cycle $c^{\gamma-1}$. By Fact (a), a_j has witnessed at least $5f + 1$ good agents with $\mathit{ready} = \mathit{True}$ by the beginning of cycle $c^{\gamma-1}$. Thus, a_j has stored *True* in *ready* at the beginning of cycle $c^{\gamma-1}$. This means that at least $7f + 1$ good agents have stored *True* in *ready* at the beginning of cycle $c^{\gamma-1}$. Therefore, a_j witnesses at least $7f + 1$ agents with $\mathit{ready} = \mathit{True}$ during cycle $c^{\gamma-1}$. Thus, at least $7f + 1$ good agents start the *AgreeID* stage from either cycle c^γ or cycle $c^{\gamma+1}$. Hence, at least $3f + 1$ good agents start the *AgreeID* stage at the same time.

4) AgreeID stage: In the *AgreeID* stage, agents in the group candidate obtain a common ID set by using consensus algorithm PCONS. To do this, a_i collects IDs of all good agents in the group candidate of a_i, say *GC*, and makes a consensus on collected IDs of agents in *GC* using PCONS. To simulate PCONS, as mentioned at section III-B1, agents in *GC* simulate one phase of the message-passing model by executing REL during one cycle. By the behavior of the *MakeCandidate* stage, since at least $3f + 1$ agents start this stage simultaneously, the execution of PCONS by agents in *GC* satisfies the PBC property. Therefore, all good agents in *GC* have a common ID set. Moreover, since collected IDs of every good agent in *GC* contains the IDs of all good agents in *GC*, the common ID set also contains them by PBC Validity 1.

5) MakeGroup stage: In the *MakeGroup* stage, agents create a reliable group. To do this, as mentioned in Section III-A, by using the common ID set, agents in the group candidate decide on a target ID based on the common ID set and gather at a single node. When agents in *GC* decide a target ID, the agents use variable *count* in addition to the common ID set. Variable *count* maintains the number of cycles that elapsed since the beginning of the *AgreeID* stage. Since agents in *GC* start the *AgreeID* stage at the same time, they have the same *count*. Therefore, agents in *GC* decide the same target ID in each cycle of the *MakeGroup* stage. The strategy for reliable group creation is as follows. Agents in *GC* decide a common target ID. Concretely, letting S be the common ID set, agents in *GC* uses the *count* $\bmod |S|$-th smallest ID in S as the target ID. Let a_{target} be the agent with the target ID. Agent a_{target} stays at the node while the other agents in *GC* search for a_{target} using REL. If at least $2f + 1$ agents gather at a single node, they form a reliable group. If agents in *GC* fail to create a reliable group at the end of the current cycle, they decide on a new target ID and repeat the above behavior. Since an agent updates *count* at the end of each cycle, these target IDs are different. Also, since the common ID set contains at least one ID of good agents in *GC* and does not include IDs of good agents not in *GC*, good agents choose their IDs as the target ID at least once and create a reliable group before they repeat $f + 1$ times.

6) Way to remove knowledge of f: Finally, we explain the way to remove knowledge of f. Since agents do not know f in practice, they use the number of collected IDs in the *Collected* stage, say $\tilde{k}$, for their decision of the *MakeCandidte* and the *MakeGroup* stages. Since there exist at least $8f + 8$

good agents in the network, and the collected IDs contain at least the IDs of all good agents, $\tilde{k} \geq 8f + 8$ holds. Also, since there exist f Byzantine agents, the number of IDs of Byzantine agents in collected IDs is at most f. From those facts, agents can calculate the condition without using f by giving the appropriate coefficients to $\tilde{k}$. However, the number of IDs of Byzantine agents in collected IDs is different among good agents. To solve this problem, we set the total number of agents to exceed the condition threshold by a comfortable margin.

Theorem 1. *Let n be the number of nodes, k be the number of agents, f be the number of weakly Byzantine agents, and a_{max} be a good agent with the largest ID among good agents. If the upper bound N of n is given to agents and $k \geq 9f + 8$ holds, Algorithm `MakeReliableGroup` makes good agents create at least one reliable group in $O(f \cdot t_{REL}(a_{max}.id))$ rounds.*

Proof. Here we briefly explain the complexity. We can prove all agents start the *AgreeID* stage no later than the end of the second cycle of *MakeCandidate* stage of a_{max}. This implies all agents start the *AgreeID* stage in $O(t_{REL}(a_{max}.id))$ rounds and the length of a cycle is $O(t_{REL}(a_{max}.id))$ rounds. Since the *AgreeID* stage requires $O(f)$ cycles by Lemma 1 and the *MakeGroup* stage requires $O(f)$ cycles, the algorithm requires $O(f \cdot t_{REL}(a_{max}.id))$ rounds. □

C. Gathering Algorithm

Here, we give a strategy of an algorithm `ByzantineGathering` that solves the gathering problem. Every good agent executes `MakeReliableGroup` unless it does not stay with a reliable group at a node. After creating a reliable group, good agents in the group collect the other good agents. To do this, when agents create the reliable group, all good agents in the reliable group execute `REL` with the smallest ID among IDs of the agents in the reliable group. They continue `REL` for the same number of rounds as the last cycle of `MakeReliableGroup`. Then, when a good agent meets the reliable group with a smaller group ID, it accompanies the group. We can guarantee that all good agents can meet during the execution of `REL`. Intuitively, this is because every good agent not in a reliable group (i.e., executing `MakeReliableGroup`) stays at a node or executes `REL` while a reliable group executes `REL`.

Theorem 2. *Let n be the number of nodes, k be the number of agents, f be the number of weakly Byzantine agents, a_{max} be a good agent with the largest ID among good agents, $|\Lambda_{good}|$ be the length of $a_{max}.id$, and $X(n)$ be the number of rounds required to explore any network composed of n nodes. If the upper bound N of n is given to agents and $k \geq 9f + 8$ holds, Algorithm `ByzantineGathering` solves the gathering problem in $O(f \cdot t_{REL}(a_{max}.id)) = O(f \cdot |\Lambda_{good}| \cdot X(N))$ rounds.*

IV. Conclusion

In this paper, we have developed an algorithm that achieves the gathering with non-simultaneous termination in weakly Byzantine environments. The algorithm reduces time complexity compared to the existing algorithm if n is given to agents, although the guarantees on simultaneous termination and startup delay are not the same. More specifically, the proposed algorithm achieves the gathering in $O(f \cdot |\Lambda_{good}| \cdot X(N))$ rounds if the upper bound N of the number of nodes is given to agents and at least $9f + 8$ agents exist in the network. In the algorithm, several good agents first create a reliable group so that good agents can trust the behavior of the group to suppress the influence of Byzantine agents. After that, the reliable group collects the other good agents, and all good agents gather at a single node. To create a reliable group, several good agents make a common ID set by simulating a parallel Byzantine consensus algorithm and gather by using the common ID set. As future work, it is interesting to consider the case that agents start at different times. In the existing gathering algorithm [3], when an agent starts the algorithm, it executes an exploring algorithm to wake up sleeping agents. By this behavior, this algorithm creates an upper bound on the startup delay between good agents, and thus it deals with the startup delay in a similar way to simultaneous startup. We will examine whether this approach can be taken for the proposed algorithm as well. It is also worth studying a gathering algorithm using a Byzantine consensus algorithm with less than $9f + 8$ agents.

References

[1] A. Pelc, "Deterministic rendezvous algorithms," in *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, P. Flocchini, G. Prencipe, and N. Santoro, Eds. Springer, 2019, pp. 423–454.

[2] Y. Dieudonné, A. Pelc, and D. Peleg, "Gathering despite mischief," *ACM Transactions on Algorithms*, vol. 11, no. 1, pp. 1–28, 2014.

[3] J. Hirose, J. Nakamura, F. Ooshita, and M. Inoue, "Weakly Byzantine gathering with a strong team," *IEICE Transactions on Information and Systems*, vol. 105, no. 3, pp. 541–555, 2022.

[4] S. Alpern and S. Gal, *The theory of search games and rendezvous*. Springer Science & Business Media, 2006.

[5] S. Bouchard, Y. Dieudonné, and B. Ducourthial, "Byzantine gathering in networks," *Distributed Computing*, vol. 29, no. 6, pp. 435–457, 2016.

[6] S. Bouchard, Y. Dieudonné, and A. Lamani, "Byzantine gathering in polynomial time," *Distributed Computing*, vol. 35, no. 3, p. 235–263, 2022.

[7] A. Miller and U. Saha, "Fast Byzantine gathering with visibility in graphs," in *16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2020*. Springer International Publishing, 2020, pp. 140–153.

[8] M. Tsuchida, F. Ooshita, and M. Inoue, "Byzantine-tolerant gathering of mobile agents in arbitrary networks with authenticated whiteboards," *IEICE Transactions on Information and Systems*, vol. 101-D, no. 3, pp. 602–610, 2018.

[9] ——, "Byzantine-tolerant gathering of mobile agents in asynchronous arbitrary networks with authenticated whiteboards," *IEICE Transactions on Information and Systems*, vol. 103-D, no. 7, pp. 1672–1682, 2020.

[10] J. Hirose, J. Nakamura, F. Ooshita, and M. Inoue, "Gathering despite a linear number of weakly Byzantine agents," 2022. [Online]. Available: https://arxiv.org/abs/2205.14937

[11] P. Khanchandani and R. Wattenhofer, "Byzantine agreement with unknown participants and failures," in *35th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2021*. IEEE, 2021, pp. 952–961.

[12] A. Dessmark, P. Fraigniaud, D. R. Kowalski, and A. Pelc, "Deterministic rendezvous in graphs," *Algorithmica*, vol. 46, no. 1, pp. 69–96, 2006.

[13] A. Ta-Shma and U. Zwick, "Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences," *ACM Transactions on Algorithms*, vol. 10, no. 3, pp. 12:1–12:15, 2014.