

Gathering despite a linear number of weakly Byzantine agents

Jion Hirose¹, Junya Nakamura², Fukuhito Ooshita³, and Michiko Inoue¹

¹Nara Institute of Science and Technology

²Toyohashi University of Technology

³Fukui University of Technology

May 31, 2022

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 for 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.

1 Introduction

1.1 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 exchange 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

Table 1: A summary of synchronous gathering algorithms in the presence of weakly Byzantine agents assuming that agents have unique IDs. Here, input is the information initially given to all agents, 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	$f + 1 \leq k$	Possible	Possible	$O(n^4 \cdot \Lambda_{good} \cdot X(n))$
[2]	F	$2F + 2 \leq k$	Possible	Possible	Poly. of n & $ \Lambda_{good} $
[3]	N	$4f^2 + 9f + 4 \leq k$	Possible	No guarantee	$O((f + \Lambda_{good}) \cdot X(N))$
[3]	N	$4f^2 + 9f + 4 \leq k$	Possible	Possible	$O((f + \Lambda_{all}) \cdot X(N))$
This study	N	$9f + 8 \leq k$	Impossible	No guarantee	$O(f \cdot \Lambda_{good} \cdot X(N))$

stay at the current node, move to a neighbor node, and convey arbitrary information to other agents, deviating from 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.

1.2 Related Works

The gathering problem has widely been studied in the literature. In particular, many of those studies deal with the gathering problem for exactly two agents, which 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 been clarified the solvability such as the gathering problem and, if solvable, they have been 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 et al. [4] have described an extensive survey of randomized rendezvous problems under the various assumptions. In the rest of this section, we describe the existing results for the deterministic gathering in the network, on which we focus in this paper.

If agents are anonymous (i.e., they do not have IDs) and no whiteboard exists (i.e., agents cannot leave any information on nodes), they cannot achieve the rendezvous for some graphs and initial arrangements because they cannot break the symmetry. Therefore, several studies [5, 6, 7] break the symmetry by attaching unique IDs to agents, and have clarified the time complexity of the rendezvous algorithm for arbitrary graphs under the assumption that agents behave synchronously in the network. Dessmark et al. [5] have provided the rendezvous algorithm in polynomial time of n , $|\lambda|$, and τ , where n is the number of nodes, $|\lambda|$ is the length of the smallest ID, and τ is the delay between the starting time of agents. Kowalski et al. [6] and Ta-Shma et al. [7] have proposed the algorithms whose time complexity are independent of τ . Also, Miller et al. [8] have investigated the trade-offs between cost and time required to solve the rendezvous problem. In contrast, some studies [9, 10] aim to optimize the memory space, and have investigated the time or the number of moves in the case where agents are anonymous. In this case, they have provided algorithms for solvable graphs and arrangements. Fraigniaud et al. [9, 11] and Czyzowicz et al. [10] have proposed algorithms for trees and an algorithm for arbitrary graphs, respectively. Furthermore, Dieudonné et al. [12] have provided the gathering algorithm in the case where agents are anonymous.

Several studies [10, 13, 14, 15, 16] have considered the rendezvous problem in asynchronous environments (i.e., different agents move at different constant speeds or move asynchronously). In the latter case, the adversary determines the speed of every agent at each time. Also, agents cannot achieve the rendezvous possibly, and thus this case allows agents to meet inside an edge.

Recently, some studies [2, 3, 17, 18, 19, 20, 21] have considered the gathering problem in the presence of Byzantine agents assuming that agents have unique IDs, which this study also address. These studies consider two types of Byzantine agents, weakly and strongly ones. While weakly

Byzantine agents can behave arbitrarily except for falsifying their own IDs, strongly Byzantine agents can behave arbitrarily including falsifying their own IDs. Table 1 summarizes this study and the related studies in the presence of weakly Byzantine agents. Note that the assumption of startup delay in Table 1 means agents may start an algorithm at different times but agents can wake up sleeping agents at the visited node.

Dieudonné et al. [2] first introduced the gathering problem in synchronous environments in the presence of weakly Byzantine agents. They have provided two gathering algorithms under the assumption that k agents exist in an arbitrary network 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 exist) 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 achieve 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 agent exist) and F is given to agents. The numbers of non-Byzantine agents used in these algorithms match the lower bounds of the number of non-Byzantine agents required to solve the gathering problem under the assumptions. Hirose et al. [3] provided the two gathering algorithm 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. [20] 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 assumes that F is given to agents and $F < k$ holds and achieves the gathering in $O(Fm)$ rounds, 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. [21] have proved that agents achieve the gathering in asynchronous environments by assuming that authenticated whiteboards exist.

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 the different assumptions. The first algorithm solve the gathering problem in exponential 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 achieve the gathering in exponential 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. [17] 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. [18] 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. [19] 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.

1.3 Contribution

Our goal is to provide an efficient algorithm that achieves the gathering with non-simultaneous termination in synchronous environments where $\Omega(k)$ weakly Byzantine agents exist. Dieudonné et al. [2] proposed the first algorithm that achieves the gathering in weakly Byzantine environments.

The algorithm achieves the gathering with simultaneous termination in $O(n^4 \cdot |\Lambda_{good}| \cdot X(n))$ rounds under the assumption that the number n of nodes is given to agents and at least $f + 1$ agents exist in the network, 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. Hirose et al. [3] proposed an algorithm with lower time complexity by assuming that $\Omega(f^2)$ non-Byzantine agents exist for f Byzantine agents in the network. The algorithm achieves the gathering with non-simultaneous termination in $O((f + |\Lambda_{good}|) \cdot X(N))$ rounds if the upper bound N of n is given to agents and at least $4f^2 + 9f + 4$ agents exist in the network. In summary, the former algorithm requires a small number of non-Byzantine agents but has high time complexity, while the latter algorithm requires a large number of non-Byzantine agents but has low time complexity. In particular, if agents need to achieve the gathering when the number of non-Byzantine agents is $\Omega(f)$, they must only choose the former algorithm with high time complexity.

In this paper, we propose a deterministic gathering algorithm with low time complexity in the existence of $\Omega(f)$ non-Byzantine agents. Since there is a large gap between the assumptions of the number of agents in the above works, it is reasonable to consider an efficient gathering algorithm under the intermediate assumption. The proposed algorithm achieves the gathering with non-simultaneous termination in $O(f \cdot |\Lambda_{good}| \cdot X(N))$ rounds under the assumption that 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. and requires fewer non-Byzantine agents than that of Hirose et al. if n is given to agents, although the guarantees on simultaneous termination and startup delay are not the same. To solve the gathering under these assumptions, we propose a new technique to simulate a consensus algorithm [22] for synchronous Byzantine message-passing systems on agent systems, in which one agent imitates one process. It is known that Byzantine consensus is solvable on a synchronous distributed system with at least $3b + 1$ processes where b is the number of Byzantine processes [23, 24]. However, it is difficult for all agents to simulate synchronous rounds of Byzantine message-passing systems and start the consensus algorithm at the same time. Instead, we construct a group of at least $3f + 1$ agents that realize the above behavior. This technique can be used not only for Byzantine gatherings but also for other problems.

2 Agent Model and Problem

2.1 Model

Agent System 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 $\{1, \dots, d(v)\}$. 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.

Agent We denote by $MA = \{a_1, a_2, \dots, a_k\}$ the set of k agents. Each agent $a_i \in MA$ has a unique ID denoted by $a_i.id \in \mathbb{N}$ and is equipped with an infinite amount of memory. Also, 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 three 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 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.

Note that, if two agents traverse the same edge in different directions at the same time, the agents cannot notice this fact.

Byzantine Agent There are f weakly Byzantine agents in the agent system. Weakly Byzantine agents act arbitrarily apart from an algorithm, except for 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 .

2.2 Gathering Problem

The gathering problem requires all good agents to transition into the terminal state at the same node. This problem allows agents to enter a terminal state at different times. 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.

3 Building Blocks

In this section, we describe two existing algorithms that are used as building blocks to design our proposed algorithm in Section 4.

3.1 Rendezvous Procedure

The proposed algorithm uses a rendezvous procedure, which allows two different agents to meet at the same node in any connected graph composed of at most N nodes if each of the agents 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 ID transformation procedure is proposed by Dessmark et al. [5]. The exploration procedure is based on universal exploration sequences (UXS) and is a corollary of the result of Ta-Shma et al. [7]. We call this rendezvous procedure $\text{REL}(id)$, where id is an ID given as input. If the execution time of the exploration procedure is t_{EX} , the procedure from [5] allows two different agents to meet at the same node in at most $(2\lfloor \log(id) \rfloor + 6)t_{EX}$ rounds. Also, the exploration procedure used in $\text{REL}(id)$ allows an agent to visit all nodes of the same type of graph in at most $t_{EX} = O(N^5 \log(N))$ if the agent knows that the number of nodes is at most N . Thus, the execution time $t_{REL}(id)$ of $\text{REL}(id)$ is $O(N^5 \log(N) \log(id))$ rounds. We have the following lemma about $\text{REL}(id)$.

Lemma 3.1 ([5]). *Let a_i and a_j be two different agents, and l_1 (resp. l_2) be ID that a_i (resp. a_j) has. Assume that a_i starts $\text{REL}(l_1)$ in round r_i , a_j starts $\text{REL}(l_2)$ in round r_j , and $l_1 \neq l_2$ holds. Then, agents a_i and a_j meet at the same node before round $\max(r_i, r_j) + t_{REL}(l_{min})$, where $l_{min} = \min(l_1, l_2)$. Furthermore, a_i (resp. a_j) visits all nodes by round $r_i + t_{REL}(a_i.id)$ (resp. round $r_j + t_{REL}(a_j.id)$).*

For integer $t \geq 0$, we write the procedure of the t -th round of $\text{REL}(id)$ by $\text{REL}(id)(t)$.

3.2 A Parallel Consensus Algorithm in Byzantine Synchronous Message-Passing Systems

The proposed algorithm uses a parallel consensus algorithm in [22] working in Byzantine Synchronous Message-Passing Systems by simulating the algorithm on agents systems, as we describe Section 4. In this section, we summarize the model and the property of the algorithm.

Model A message-passing distributed system, in which processors communicate by sending messages, is modeled by an undirected complete graph with m nodes. The nodes have unique IDs, and these IDs do not necessarily have to be consecutive. The system includes at most b Byzantine nodes, which can act arbitrarily apart from an algorithm. We call all nodes except Byzantine nodes *good* nodes. At the beginning of an execution, each node knows its ID only and knows neither the number m of nodes, the number b of Byzantine nodes, nor the other nodes' IDs. The system is synchronous, that is, nodes repeat synchronous phases. In a phase p , every good node executes local computation, sends messages to some nodes, and then receives the messages that were sent to it in phase p . 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. Every message has the ID of its sender; thus, when a node receives a message, it can obtain the sender's ID. There is no restriction on the actions of Byzantine nodes except for falsifying their IDs to a directly communicating node.

Parallel Byzantine Consensus Problem 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:

Validity 1 If (id, x) is an input pair of every good node and $x \neq \perp$, then the output set of a good node must include (id, x) .

Validity 2 If (id, x) is not an input pair of any good node, then 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.

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

If an algorithm satisfies the above four conditions, we say it satisfies the parallel Byzantine consensus property (in short, PBC property). The PBC property allows that, if (id, x) is included in an input set of a part of good nodes, but not all good nodes, (id, x) may not included in the output set of any good node.

Parallel Byzantine Consensus Algorithm The proposed algorithm uses the parallel Byzantine consensus algorithm in [22]. We call this algorithm $PCONS(S)$, where S is a set given as input. We have the following lemma about $PCONS(S)$.

Lemma 3.2 ([22]). *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.*

4 Byzantine Gathering Algorithm

To achieve the gathering, the proposed algorithm uses a subroutine that is used as a building block. In this section, we first explain the overview of the proposed algorithm. After that, we give an idea and a detailed description of the subroutine, and then we show an algorithm to achieve the gathering. 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 .

4.1 Overview

Here, 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 in Sections 4.2 and 4.3. The underlying idea of the algorithm is made of the following three steps:

- (1) All agents collect all agent IDs by using the rendezvous algorithm REL.
- (2) After collecting all good agents' IDs, every agent decides on an ID that is common to other good agents as a target ID from the collected IDs.
- (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 on a common target ID by choosing the smallest ID of the collected IDs. Therefore, in Step (3), all agents gather at the node where the agent with the smallest ID exists. However, if there is a Byzantine agent, that idea fails. Let us consider the case where 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. Also, even if all 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:

- (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 in Section 3.2.
- (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 3.2. Since the consensus algorithm assumes message-passing systems, agents simulate the system by using the rendezvous algorithm REL. Simply put, agents exchange messages when they meet other agents by REL. In Step (a), agents create a group candidate of at least $3f + 1$ agents such that they can send messages to 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 on 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. If agents do not gather sufficiently, 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 reliable group decide on 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. Furthermore, when each good agent meets the reliable group with a smaller group ID, it accompanies the group. All the group 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 gathering algorithm by Hirose et al. [3] employs another strategy to create a reliable group from collecting IDs instead of using a consensus algorithm. In the algorithm, each good agent searches for one of the agents with the smallest $f + 1$ IDs of the collected IDs to gather at the node where the agent is. Because good agents may be divided into $\Omega(f)$ nodes in this strategy, this strategy requires at least $\Omega(f^2)$ good agents to guarantee that a reliable group is created at any of those nodes. On the other hand, the proposed algorithm uses the strategy such that $\Omega(f)$ good agents make a common ID set, search for a target agent one by one synchronously, and try to gather at the node with any of $\Omega(f)$ agents. Therefore, the algorithm requires $\Omega(f)$ good agents, i.e., the key to the algorithm is the reliable group creation procedure using the consensus algorithm.

4.2 Algorithm to Create a Reliable Group

In this section, we explain an algorithm to create a reliable group, called **MakeReliableGroup**, by assuming that $k = g + f \geq 9f + 8$. Recall that agents know N , but do not know n , k , or f .

4.2.1 Idea of the Algorithm

As mentioned in Section 4.1, 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 Byzantine 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 **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 with a_j . 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 Byzantine 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 **REL**($a_i.id$), a_i starts **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.

4.2.2 Details of the Algorithm

Algorithm 1 shows the behavior of each round of Algorithm **MakeReliableGroup** and executes one of Algorithms 2–5 depending on the current stage. In **MakeReliableGroup**, the procedure **WAIT**() means that an agent stays at the current node for one round. Table 2 summarizes variables used in **MakeReliableGroup**. Variable *stage* keeps the current stage of a_i and its initial value is *CollectID*.

Algorithm 1 MakeReliableGroup

```
1:  $a_i.elapsed \leftarrow a_i.elapsed + 1$ 
2: if  $a_i.stage = CollectID$  then
3:   // While executing CollectIDStage,  $a_i$  executes  $a_i.length \leftarrow 2 \cdot a_i.length$ .
4:   Execute CollectIDStage
5: else if  $a_i.stage = MakeCandidate$  then
6:   // While executing MakeCandidateStage,  $a_i$  executes  $a_i.length \leftarrow 2 \cdot a_i.length$ .
7:   Execute MakeCandidateStage
8: else if  $a_i.stage = AgreeID$  then
9:   // While executing AgreeIDStage,  $a_i$  executes  $a_i.count \leftarrow a_i.count + 1$ 
10:  Execute AgreeIDStage
11: else if  $a_i.stage = MakeGroup$  then
12:  // While executing MakeGroupStage,  $a_i$  executes  $a_i.count \leftarrow a_i.count + 1$ 
13:  Execute MakeGroupStage
14: end if
```

Agent a_i has variable $length$ to keep the length of the current cycle. Agent a_i doubles $length$ at the last of each cycle if $a_i.stage \in \{CollectID, MakeCandidate\}$. The initial value of $length$ is T_{ini} , where T_{ini} is a given positive integer. Agent a_i has variable $elapsed$ to keep the number of rounds from the beginning of the current cycle. Variable $count$ maintains the number of cycles that elapsed after the beginning of the *AgreeID* stage.

The overall flow of *MakeReliableGroup* is shown in Fig. 1. In *MakeReliableGroup*, a_i executes the *CollectID* stage, the *MakeCandidate* stage, the *AgreeID* stage, and the *MakeGroup* stage in this order (Algorithm 1). All good agents have the same initial length of a cycle and double their length every cycle until they start the *AgreeID* stage. However, as we explain later, good agents may transition into the next stage at different cycles. Therefore, good agents have the same length of a cycle until they start the *AgreeID* stage. The following observation shows this fact formally. For simplicity, we denote γ -th cycle of an agent a_i and its length by c_i^γ and $|c_i^\gamma|$, respectively.

Observation 4.1. *Let a_i and a_j be good agents. If a_i and a_j are in the *CollectID* stage or the *MakeCandidate* stage, $|c_i^\gamma| = |c_j^\gamma|$ holds for any γ and they started these cycles at the same time.*

In *MakeReliableGroup*, agents that start the *AgreeID* stage simultaneously compose a group candidate. An agent a_i can check whether another agent belongs to the same group candidate as follows. Assume that a_i stores *AgreeID* in $a_i.stage$ in round r .

Let a_j be an agent that also stores *AgreeID* in $a_j.stage$ in round r . By Observation 4.1, $|c_i^\gamma| = |c_j^\gamma|$ holds for c_i^γ and c_j^γ that include round r . Furthermore, since an agent does not double the length of the cycle in the *AgreeID* and the *MakeGroup* stages, $|c_i^\varepsilon| = |c_j^\varepsilon|$ holds for any $\varepsilon > \gamma$.

Let us consider another agent a_ℓ that stores *AgreeID* in $a_\ell.stage$ in round r' ($r' \neq r$). Since the total number of cycles that a_ℓ has executed in the *CollectID* and the *MakeCandidate* stages is greater or less than that of a_i , the number of updates of $a_\ell.length$ is different from that of $a_i.length$. Thus, $|c_i^\varepsilon| = |c_\ell^\zeta|$ does not hold, where c_ℓ^ζ is the cycle when a_ℓ starts the *AgreeID* stage. Hence, when a_i witnesses a_ℓ at the current node after it starts the *AgreeID* stage, a_i can understand that a_ℓ started the *AgreeID* stage at a different round by observing $a_\ell.length$. In other words, a_i can understand that a_ℓ is a member of the group candidate different from a_i . The following observation summarizes this discussion.

Observation 4.2. *Let a_i and a_j be good agents that have entered the *AgreeID* or the *MakeGroup* stage. If a_i and a_j started the *AgreeID* stage at the same time, $|c_i^\gamma| = |c_j^\gamma|$ holds for any γ , otherwise $|c_i^\gamma| = |c_j^\gamma|$ does not hold.*

Consider the case where a_i starts the *AgreeID* stage faster than a_ℓ like Fig. 1b. Since a_i does not update $a_i.length$ anymore, the length of each cycle of the *AgreeID* and the *MakeGroup* stages is identical. On the other hand, a_ℓ doubles $a_\ell.length$ every cycle until it starts the *AgreeID* stage.

Table 2: Variables of agent a_i .

Variable	Initial value	Explanation
$stage$	$CollectID$	The current stage of a_i . This variable takes one of the following values: $CollectID$, $MakeCandidate$, $AgreeID$, $MakeGroup$.
$length$	T_{ini}	The length of the current cycle.
$elapsed$	0	The number of rounds from the beginning of the current cycle.
$count$	0	The number of cycles from the beginning of the $AgreeID$ stage.
$ready$	$False$	$True$ if and only if a_i has met a certain condition, which represents the decision that a_i is ready to transition into the $AgreeID$ stage in the $MakeCandidate$ stage.
$endMakeCandidate$	$False$	$True$ if and only if a_i has met the condition to transition into the $AgreeID$ stage.
gid	∞	The group ID of the reliable group to which a_i belongs.
R	\emptyset	A set of IDs of agents such that a_i knows they satisfy $ready = True$.
S_p	$\{a_i.id\}$	A set of agent IDs that a_i has collected in the $CollectID$ stage.
S_c	\emptyset	An output of $PCONS(S_p)$.
P_p	\emptyset	A set of IDs of agents such that a_i knows they belong to the same group candidate as a_i .
P_c	\emptyset	An output of $PCONS(P_p)$, which is used as a common ID set. Its elements are ordered in increasing order.

This implies that $a_\ell.length \equiv 0 \pmod{a_i.length}$ holds. That is, when a_ℓ starts a cycle, a_i also starts a cycle. From this discussion and Observations 4.1 and 4.2, we have the following observation.

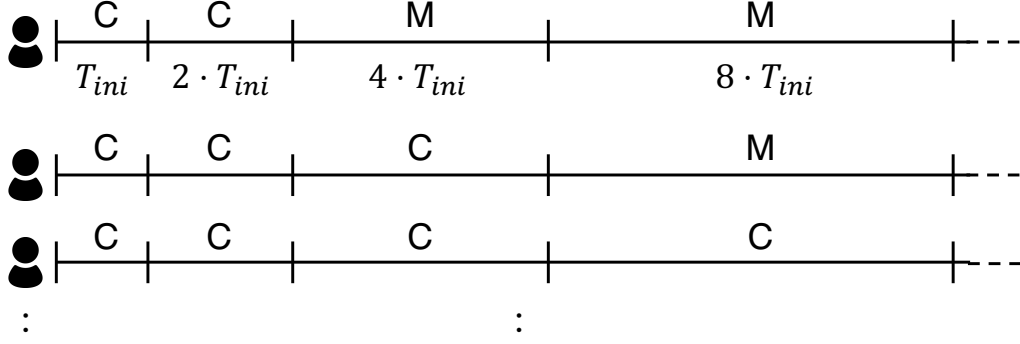
Observation 4.3. *Let a_i and a_j be good agents such that a_j starts the $AgreeID$ stage no earlier than a_i . When a_j starts a cycle, a_i also starts a cycle at the same time.*

CollectID Stage Algorithm 2 is the pseudo-code of the $CollectID$ stage. This stage aims to collect IDs of all good agents. An agent a_i uses variable S_p to record the IDs collected in the $CollectID$ stage.

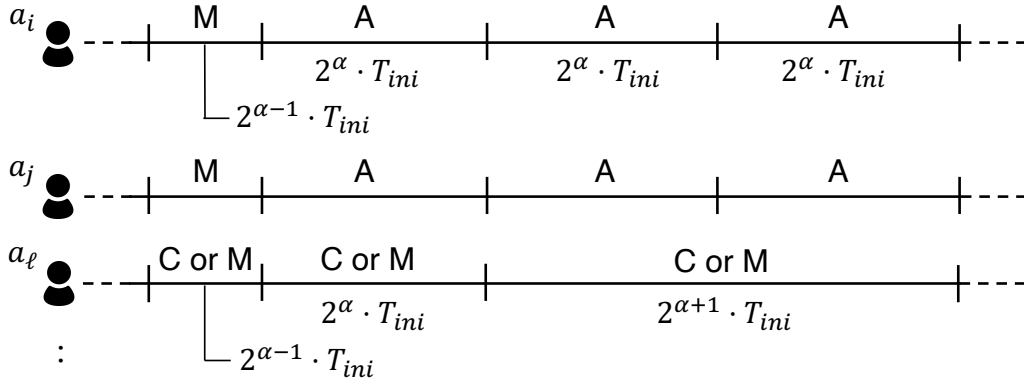
If a_i witnesses an agent a_j with $a_j.ready = True$ at the current node at the beginning of a round, a_i stores $a_j.id$ in variable $a_i.R$ to record IDs of such agents (Line 1 of Algorithm 2). Recall that A_i is a set of agents (including a_i) that stay at the current node of a_i at the beginning of a round. We will explain the details of variable $a_i.ready$ and $a_i.R$ in Section 4.2.2.

If $a_i.length < 2 \cdot (t_{REL}(a_i.id) + 1)$ holds in a cycle, a_i stays at the current node for the cycle. Additionally, a_i updates variables at the last round of the current cycle for the next cycle. To determine whether the current round is the last one of the current cycle, a_i uses variable $elapsed$, which keeps the number of rounds elapsed during the current cycle. If $a_i.length = a_i.elapsed$ holds, a_i understands that the current round is the last one of the current cycle and updates variables $a_i.elapsed$ and $a_i.length$ for the next cycle (Lines 3–6).

If $a_i.length \geq 2 \cdot (t_{REL}(a_i.id) + 1)$ holds in a cycle, a_i collects IDs of all good agents using $REL(a_i.id)$ (Lines 8–18). Agent a_i uses variable S_p to record collected agent IDs. If a_i witnesses a_j at the current node at the beginning of a round, a_i stores $a_j.id$ in $a_i.S_p$ (Line 9). After that, if the current round is not the last round, that is, $a_i.length > a_i.elapsed$ holds, a_i executes $REL(a_i.id)(a_i.elapsed)$ (Lines 10–11). Otherwise, that is, if $a_i.length = a_i.elapsed$ holds, a_i stores $MakeCandidate$ in $a_i.stage$ (to move the $MakeCandidate$ stage in the next cycle) and stays at the current node for one round (Lines 12–17). As we prove later, if $a_i.length \geq 2 \cdot (t_{REL}(a_i.id) + 1)$ holds in the current cycle, a_i meets all



(a) At starting `MakeReliableGroup`



(b) At starting the *AgreeID* stage

Figure 1: The stage flow of Algorithm `MakeReliableGroup`. Notions C, M, and A represent cycles of the *CollectID* stage, *MakeCandidate* stage, and the *AgreeID* stage, respectively.

other good agents by the end of the last round of the current cycle. Therefore, when a_i finishes the last round of that cycle, $a_i.S_p$ includes IDs of all good agents.

MakeCandidate Stage Algorithm 3 is the pseudo-code of the *MakeCandidate* stage. This stage aims to create a group candidate consisting of at least $3f + 1$ good agents. As we explain in Section 4.2.2, good agents estimate f to create a reliable group consists of at least $2f + 1$ agents. However, their estimated values differ by at most f . Thus, `MakeReliableGroup` requires at least $3f + 1$ good agents to allow for that error. In order to describe the *MakeCandidate* stage clearly, we define the group candidate as follows.

Definition 4.1 (Group candidate). *A set GC of agents is a group candidate if and only if all good agents in GC satisfy the following: (1) All the good agents in GC start the *AgreeID* stage at the same time. (2) For any good agent $a_i \in GC$, $a_i.length \geq 4 \cdot (t_{REL}(a_i.id) + 1)$ holds after the beginning of the *MakeCandidate* stage.*

By the behavior of the *CollectID* stage, when an agent a_i starts the *MakeCandidate* stage, the length of a_i 's cycle is at least $4 \cdot (t_{REL}(a_i.id) + 1)$. Furthermore, since the *MakeCandidate* stage does not include the action of reducing the length of a cycle, a_i already satisfies Requirement (2) of Definition 4.1. Also, by Observation 4.1, every good agent in the *MakeCandidate* stage has the same length of a cycle. Therefore, agents that start the *AgreeID* stage at the same time satisfy Requirement (1). Consequently, if at least $3f + 1$ of the good agents start the *AgreeID* stage simultaneously, they achieve the purpose of the *MakeCandidate* stage.

Algorithm 2 CollectIDStage

```
1:  $a_i.R \leftarrow a_i.R \cup \{a_j.id \mid a_j \in A_i \wedge a_j.ready = True\}$ 
2: if  $2 \cdot (t_{REL}(a_i.id) + 1) > a_i.length$  then
3:   if  $a_i.length = a_i.elapsed$  then
4:      $a_i.elapsed \leftarrow 0$ 
5:      $a_i.length \leftarrow 2 \cdot a_i.length$ 
6:   end if
7:   Execute WAIT()
8: else  $\triangleright 2 \cdot (t_{REL}(a_i.id) + 1) \leq a_i.length$ 
9:    $a_i.S_p \leftarrow a_i.S_p \cup \{a_j.id \mid a_j \in A_i\}$ 
10:  if  $a_i.length > a_i.elapsed$  then
11:    Execute REL( $a_i.id$ )( $a_i.elapsed$ )
12:  else
13:     $a_i.elapsed \leftarrow 0$ 
14:     $a_i.length \leftarrow 2 \cdot a_i.length$ 
15:     $a_i.stage \leftarrow MakeCandidate$ 
16:    Execute WAIT()
17:  end if
18: end if
```

Agent a_i executes REL($a_i.id$) every cycle of the *MakeCandidate* stage. First, if a_i witnesses an agent a_j with $a_j.ready = True$ at the current node at the beginning of a round, it stores $a_j.id$ in $a_i.R$ (Line 1 of Algorithm 3). Then, if a_i satisfies either of the following two conditions, a_i stores *True* in *ready* (Lines 2–7).

- (1) Variable $a_i.S_p$ contains at least $(8/9)|a_i.S_p|$ IDs of agents that have started the *MakeCandidate* stage.
- (2) Agent a_i has witnessed at least $(4/9)|a_i.S_p|$ agents with *ready* = *True* from the beginning of *MakeReliableGroup*.

By the behavior of the *CollectID* stage, for a good agent a_i , when the length of a_j 's cycle becomes at least $4 \cdot (t_{REL}(a_j.id) + 1)$, a_j starts the *MakeCandidate* stage. Therefore, a_i can determine Condition (1) by checking whether $a_i.S_p$ contains at least $(8/9)|a_i.S_p|$ IDs, each *id* of which satisfies $a_i.length \geq 4 \cdot (t_{REL}(id) + 1)$. Also, a_i can decide Condition (2) by judging whether $a_i.R$ includes at least $(4/9)|a_i.S_p|$ IDs. Next, if $a_i.R$ contains at least $(6/9)|a_i.S_p|$ IDs, a_i stores *True* in variable *endMakeCandidate* (Lines 8–10). It means that a_i starts the *AgreeID* stage from next cycle. Finally, if the current round is not the last one of the current cycle, that is, $a_i.length > a_i.elapsed$ holds, then a_i executes REL($a_i.id$)($a_i.elapsed$) to inform other agents of the current state of $a_i.ready$ (Lines 11–12). Otherwise, that is, if $a_i.length = a_i.elapsed$ holds, a_i stays at the current node for one round (Lines 13–20). In this case, if $a_i.endMakeCandidate = True$ holds, a_i stores *AgreeID* in $a_i.stage$ before it stays for one round (Lines 16–18).

By the above behavior, at least one group candidate consisting of at least $3f + 1$ good agents is created. The reason is as follows. Assume that agent a_{ini} is the first good agent that stores *True* in *endMakeCandidate*, and let c_{ini}^γ be the cycle in which a_{ini} does so. In this case, the following two situations occur.

- (a) Each good agent that started the *MakeCandidate* stage has witnessed at least $(6/9)|a_{ini}.S_p| - f$ good agents with *ready* = *True* by the beginning of cycle c_{ini}^γ .
- (b) There exists at least one good agent that satisfies Condition (1) (Line 3 of Algorithm 3) by the beginning of cycle c_{ini}^γ (proved in Lemma 4.5).

Let a_ℓ be a good agent that satisfies Condition (1) by the beginning of cycle c_{ini}^γ . By Condition (1) and Situation (b), the following situation occurs:

Algorithm 3 MakeCandidateStage

```
1:  $a_i.R \leftarrow a_i.R \cup \{a_j.id \mid a_j \in A_i \wedge a_j.ready = True\}$ 
2: if  $a_i.elapsed = 1 \wedge (\text{either (1) or (2) holds}) \wedge a_i.ready = False$  then
3:   // (1)  $|\{id \in a_i.S_p \mid a_i.length \geq 4 \cdot (t_{REL}(id) + 1)\}| \geq (8/9)|a_i.S_p|$ 
4:   // (2)  $|a_i.R| \geq (4/9)|a_i.S_p|$ 
5:    $a_i.ready \leftarrow True$ 
6:    $a_i.R \leftarrow a_i.R \cup \{a_i.id\}$ 
7: end if
8: if  $a_i.elapsed = 1 \wedge |a_i.R| \geq (6/9)|a_i.S_p|$  then
9:    $a_i.endMakeCandidate \leftarrow True$ 
10: end if
11: if  $a_i.length > a_i.elapsed$  then
12:   Execute REL( $a_i.id$ )( $a_i.elapsed$ )
13: else
14:    $a_i.elapsed \leftarrow 0$ 
15:    $a_i.length \leftarrow 2 \cdot a_i.length$ 
16:   if  $a_i.endMakeCandidate = True$  then
17:      $a_i.stage \leftarrow AgreeID$ 
18:   end if
19:   Execute WAIT()
20: end if
```

(b') At least $(8/9)|a_\ell.S_p| - f$ good agents have started the *MakeCandidate* stage by cycle c_{ini}^γ .

Here, $(8/9)|a_\ell.S_p| - f \geq (8/9)g - f = (6/9)g + (2/9)g - f \geq (6/9)g + (2/9)(8f + 8) - f = (6/9)g + (7/9)f + 16/9 > (6/9)k$ holds because $k \geq |a_\ell.S_p| \geq g$ holds, which we prove by Corollary 4.1. That is, at least $(6/9)k$ good agents have started the *MakeCandidate* stage by cycle c_{ini}^γ . Here, let a_m be such a good agent. By Situation (a), a_m has witnessed at least $(6/9)|a_{ini}.S_p| - f \geq (6/9)g - f = (4/9)g + (2/9)g - f \geq (4/9)g + (2/9)(8f + 8) - f = (4/9)g + (7/9)f + 16/9 > (4/9)k \geq (4/9)|a_m.S_p|$ agents with *ready* = *True* before cycle c_{ini}^γ . Therefore, since a_m stores *True* in $a_m.ready$ before cycle c_{ini}^γ , a_m witnesses at least $(6/9)k \geq (6/9)|a_m.S_p|$ agents with *ready* = *True* during cycle c_{ini}^γ . Thus, at least $(6/9)k \geq (6/9)(9f + 8) > 6f + 1$ good agents store *True* in *endMakeCandidate* in the first round of either c_{ini}^γ or $c_{ini}^{\gamma+1}$. Hence, at least $3f + 1$ good agents start the *AgreeID* stage at the same time.

AgreeID Stage Algorithm 4 is the pseudo-code of the *AgreeID* stage. This stage aims to obtain a common ID set among the agents in a group candidate by using consensus algorithm PCONS. The common ID set contains at least $f + 1$ agent IDs in the same group candidate. The agents use the common ID set in order to create a reliable group in the *MakeGroup* stage. By the behavior of the *MakeCandidate* stage, we have the following observation.

Observation 4.4. *Let a_i and a_j be good agents. If a_i and a_j belong to the same group candidate, for any γ , they start the first round of c_i^γ and c_j^γ at the same time.*

In order for agents to efficiently create a reliable group, an agent collects IDs in the same group candidate and makes a consensus on a set of the collected IDs. In the *MakeGroup* stage, agents create the reliable group by the method using the group candidate as described in Section 4.1. If agents make a consensus on S_p to make the common ID set, the output may contain agent IDs not in the same group candidate. This approach makes the reliable group creation in the *MakeGroup* stage inefficient. Therefore, in the *AgreeID* stage, agents collect agent IDs in the same group candidate and propose the IDs for consensus. Since the output contains agent IDs in the same group candidate only, this idea can avoid the above problem. An agent uses variable P_p to record the IDs collected in the *AgreeID* stage.

Algorithm 4 AgreeIDStage

```
1: if  $a_i.count = 0$  then
2:    $a_i.P_p \leftarrow a_i.P_p \cup \{a_j.id \mid a_j \in A_i \wedge a_j.length = a_i.length \wedge a_j.stage = AgreeID\}$ 
3: else
4:   Execute  $PCONS(a_i.S_p)(a_i.count)$ 
5:   Execute  $PCONS(a_i.P_p)(a_i.count)$ 
6: end if
7: if  $a_i.length > a_i.elapsed$  then
8:   Execute  $REL(a_i.id)(a_i.elapsed)$ 
9: else
10:   $a_i.elapsed \leftarrow 0$ 
11:   $a_i.count \leftarrow a_i.count + 1$ 
12:  if  $PCONS(a_i.S_p)$  and  $PCONS(a_i.P_p)$  are finished then
13:     $a_i.S_c \leftarrow$  the output of  $PCONS(a_i.S_p)$ 
14:     $a_i.P_c \leftarrow$  the output of  $PCONS(a_i.P_p)$ 
15:     $a_i.stage \leftarrow MakeGroup$ 
16:  end if
17:  Execute  $WAIT()$ 
18: end if
```

In addition, an agent makes a consensus on S_p in this stage. As we explain in Section 4.2.2, the output of the consensus is used in the *MakeGroup* stage.

Hereinafter, we explain the detailed behaviors of the *AgreeID* stage. If an agent a_i executes the first cycle c_i^γ of the *AgreeID* stage, that is, $a_i.count = 0$ holds, a_i collects agent IDs in the same group candidate, say GC (Lines 1–2 of Algorithm 4). To be more precise, if a_i witnesses the agent a_j in GC at the current node in a round of cycle c_i^γ , a_i stores $a_j.id$ in $a_i.P_p$. Agent a_i determines whether a_j belongs to GC by checking $a_j.length$ and $a_j.stage$. Agent a_i meets all good agents in GC and includes their IDs in $a_i.P_p$ by the end of cycle c_i^γ .

If a_i executes the second or later cycle of the *AgreeID* stage, that is, $a_i.count > 0$ holds, then a_i makes a consensus on $a_i.S_p$ and $a_i.P_p$ using $PCONS(a_i.S_p)$ and $PCONS(a_i.P_p)$ to obtain the common ID sets with good agents in GC (Lines 3–6). As we mentioned in Section 4.2.1, agents simulate one phase of the message-passing model by executing REL during one cycle. In Algorithm 4, when an agent executes $PCONS(S)(q)$ except the last round of a cycle, it shares messages of algorithm $PCONS(S)$ with other agents at the current node. If an agent executes $PCONS(S)(q)$ in the last round of a cycle, it computes the q -th phase of $PCONS(S)$ using the collected messages.

To simulate $PCONS(a_i.S_p)$ and $PCONS(a_i.P_p)$, a_i meets the other agents in GC using REL for a cycle. If the current round is not the last round of the current cycle, that is, $a_i.length > a_i.elapsed$ holds, then a_i executes $REL(a_i.id)(a_i.elapsed)$ to meet the other agents in GC in the first cycle, and to execute $PCONS(S)(q)$ with agents in GC in and after the second cycle (Lines 7–8). To make a consensus with good agents in GC , a_i checks to which group candidate an agent belongs, identically with P_p . Otherwise, that is, if $a_i.length = a_i.elapsed$ holds, then a_i first checks the status of $PCONS(a_i.S_p)$ and $PCONS(a_i.P_p)$. If both the consensus instances have finished, a_i stores their outputs in $a_i.S_c$ and $a_i.P_c$, respectively and changes $a_i.stage$ into *MakeGroup* (Lines 12–16). Agent a_i then stays at the current node for one round (Line 17). If GC includes at least $3f + 1$ agents, the execution of $PCONS$ by agents in GC satisfies the PBC property by Observation 4.4. Therefore, all the good agents in GC has identical P_c and S_c . Moreover, since P_p of every good agent contains the IDs of all good agents in GC , P_c also contains them by PBC Validity 1.

MakeGroup Stage Algorithm 5 is the pseudo-code of the *MakeGroup* stage. This stage aims to create a reliable group. An agent a_i has variable *gid* to store its group ID when it becomes a member of a reliable group. In Algorithm *ByzantineGathering*, which we describe later, since agents do not know f , when an agent a_ℓ determines whether a reliable group exists at the current node, a_ℓ uses $(1/8)|a_\ell.S_p|$ instead of f for its decision. However, $|a_\ell.S_p|$ differs from the other good agents

Algorithm 5 MakeGroupStage

```
1: if  $(1/2) \cdot a_i.length \geq a_i.elapsed$  then
2:   Execute REL( $a_i.id$ )( $a_i.elapsed$ )
3: else if  $a_i.length > a_i.elapsed$  then
4:   if  $\exists a_j \in A_i[a_j.id = a_i.P_c[a_i.count \bmod |a_i.P_c|]]$  then
5:     Execute WAIT()
6:   else
7:     Execute REL( $a_i.id$ )( $a_i.elapsed$ )
8:   end if
9: else
10:   $a_i.elapsed \leftarrow 0$ 
11:   $a_i.count \leftarrow a_i.count + 1$ 
12:   $a_i.D \leftarrow \{a_j.id \mid a_j \in A_i \wedge |a_j.S_c| \geq (8/9)|a_j.S_p| \wedge a_j.length = a_i.length \wedge a_j.S_c = a_i.S_c \wedge a_j.stage = MakeGroup\}$ 
13:  if  $|a_i.S_c| \geq (8/9)|a_i.S_p| \wedge |a_i.D| \geq (3/9)|a_i.S_c|$  then
14:     $a_i.gid \leftarrow \min(a_i.D)$ 
15:  end if
16:  Execute WAIT()
17: end if
```

because every good agent possibly met a different number of Byzantine agents in the *CollectID* stage. Therefore, in order to be able to recognize a reliable group even if a good agent has any S_p , we define a reliable group as follows.

Definition 4.2 (Reliable group). *A set RG of agents is a reliable group if and only if RG contains at least $k/8$ good agents and $a_i.gid = a_j.gid$ holds for any two different good agents $a_i, a_j \in RG$.*

Agents behaves differently in the first and second halves of each cycle. If the current round is in the first half of the current cycle, that is, $(1/2) \cdot a_i.length \geq a_i.elapsed$ holds, a_i executes **REL**($a_i.id$)($a_i.elapsed$) to meet agents in the *CollectID* stage and a reliable group that acts for gathering by Algorithm *ByzantineGathering* (Lines 1–2 of Algorithm 5).

If the current round is in the second half of the current cycle, that is, $a_i.length \geq a_i.elapsed > (1/2) \cdot a_i.length$ holds, a_i creates a reliable group using the common ID set of the *AgreeID* stage (Lines 3–16). Let GC be a group candidate of a_i . As we mentioned in Section 4.1, agents in GC decide a target ID base on the common ID set. More concretely, they use variables P_c and $count$ to decide a target ID. Since agents in GC count the number of cycles from the time they started the *AgreeID* stage, they have the same $count$. Therefore, agents in GC decide the same target ID in each cycle of the *MakeGroup* stage. The strategy for the reliable group creation is as follows. Agents in GC decide a target ID from variables P_c and $count$ and searches for the agent with the target ID, say a_{target} , using **REL**. Agent a_{target} stays at the node while the other agents in GC search for a_{target} . If agents in GC fail to create a reliable group in the last round of the current cycle, they decide on a new target ID from the common ID set using $count$ in the first round of the next cycle and repeat the above behavior. Since an agent updates $count$ in the last round 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 by they repeat $f + 1$ times.

Agent a_i stores the common ID set in $a_i.P_c$ and achieves the above behavior as follows. If a_i has the target ID, a_i stays at the node until the last round of the current cycle, otherwise a_i searches for a_{target} using **REL** until the last round of the current cycle (Lines 3–8). Then, a_i updates variables for the next cycle in the last round of the cycle (Lines 9–17).

Hereinafter, we explain the detailed behaviors of the second half of the *MakeGroup* stage. If the current round is not the last round of the current cycle, that is, $a_i.length > a_i.elapsed$ holds, and either of the following is satisfied, then a_i stays at the current node for one round (Lines 4–5).

- (1) a_i has the target ID.

(2) a_i meets a_{target} at the current node.

Agent a_i calculates $a_i.P_c[a_i.count \bmod |a_i.P_c|]$ to determine Conditions (1) and (2). The formula is for a_i to determine the target ID and uses $a_i.P_c$ and $a_i.count$ to determine the same target ID among good agents in GC . Since agents in GC start the *AgreeID* stage at the same time, and they start a cycle at the same time by Observation 4.3, every good agent in GC has the same *count*. Furthermore, if GC contains at least $3f + 1$ agents, every good agent in GC has the same P_c by Lemma 4.10, which we prove later, and calculates the same target ID.

If the current round is not the last one of the current cycle, that is, $a_i.length > a_i.elapsed$ holds, and neither Conditions (1) nor (2) is satisfied, a_i executes $REL(a_i.id)(a_i.elapsed)$ to meet a_{target} (Lines 6–8).

If the current round is the last round of the current cycle, that is, $a_i.length = a_i.elapsed$ holds, a_i creates a reliable group if possible and stays at the current node for one round (Lines 9–17). If a_i witnesses an agent a_j that satisfies all of the following, a_i stores $a_j.id$ in variable D to record such IDs (Line 12).

- (a) $a_j.S_c$ contains at least $(8/9)|a_j.S_p|$ IDs.
- (b) a_j belongs to GC .
- (c) $a_i.S_c$ and $a_j.S_c$ are the same.
- (d) a_j has started the *MakeGroup* stage.

If $a_i.S_c$ contains at least $(8/9)|a_i.S_p|$ IDs and $a_i.D$ contains at least $(3/9)|a_i.S_c|$ IDs, a_i stores the smallest ID in $a_i.D$ in $a_i.gid$ as the group ID (Lines 13–15).

By the above behavior, assuming that a_i stores the group ID in $a_i.gid$ in the last round r of some cycle, we can guarantee that the agents in GC create the reliable group. In round r , there exist at least $(3/9)|a_i.S_c|$ agents that satisfy Conditions (a)–(d) at the current node, and $(3/9)|a_i.S_c| \geq (3/9)(8/9)|a_i.S_p| = (8/27)|a_i.S_p| \geq (8/27)g$ holds by Line 13. If GC contains at least $3f + 1$ agents, every good agent in GC has the same S_c . Furthermore, agents at the current node have the same D . Therefore, at least $(8/27)g - f$ good agents reach the same decision. By $k \geq 9f + 8$ and $g \geq 8f + 8$, $(8/27)g - f = (27/216)g + (37/216)g - f \geq (27/216)g + (37/216)(8f + 8) - f = (27/216)g + (80/216)f + 296/216 > (27/216)(g + f) = k/8$ holds. Hence, in round r , at least $k/8$ good agents, including a_i , store the same group ID and create a reliable group.

4.2.3 Correctness and Complexity Analysis

In this subsection, we prove the correctness and complexity of *MakeReliableGroup*. First, we prove that, when agent a_i executes $REL(a_i.id)$ throughout a cycle, it can meet all agents. Note that, at the beginning of a cycle, a_i determines whether it executes $REL(a_i.id)$ or waits for the whole cycle. If a_i starts $REL(a_i.id)$, it stops the procedure at the middle of a cycle only if a_i satisfies Line 4 of Algorithm 5. Even in this case, a_i executes $REL(a_i.id)$ for at least the half of the cycle. We say “ a_i executes $REL(a_i.id)$ without interruption” if a_i executes $REL(a_i.id)$ throughout a cycle.

Lemma 4.1. *Let a_i be a good agent and c_i^γ be a cycle in which a_i starts $REL(a_i.id)$ at the beginning of the first round of this cycle. If a_i executes $REL(a_i.id)$ until the last round of cycle c_i^γ without interruption, then a_i meets all good agents during cycle c_i^γ .*

Proof. By the behavior of *MakeReliableGroup*, since a_i starts $REL(a_i.id)$ in cycle c_i^γ , a_i satisfies Line 8 of Algorithm 2 no later than the first round of cycle c_i^γ and thus $|c_i^\gamma| \geq 2 \cdot (t_{REL}(a_i.id) + 1)$ holds. Let round r be the first round of cycle c_i^γ . Let a_j be a good agent other than a_i and c_j^ϵ be a cycle of a_j that includes round r . We prove that a_i and a_j meet during cycle c_i^γ . We consider two cases, $|c_i^\gamma| \geq |c_j^\epsilon|$ and $|c_i^\gamma| < |c_j^\epsilon|$.

First, we consider the case $|c_i^\gamma| \geq |c_j^\epsilon|$. In this case, by the behavior of *MakeReliableGroup*, a_j starts the *AgreeID* stage no later than a_i . Therefore, by Observation 4.3, a_i and a_j start cycles c_i^γ and c_j^ϵ at the same time at the beginning of round r . Furthermore, when a_i starts cycle $c_i^{\gamma+1}$,

a_j also starts $c_j^{\varepsilon+\alpha+1}$ at the same time for a non-negative integer α . In other words, a_j finishes all cycles $c_j^\varepsilon, \dots, c_j^{\varepsilon+\alpha}$ by the time a_i finishes cycle c_i^γ . By the behavior of **MakeReliableGroup**, in each cycle, a_j executes $\text{REL}(a_j.id)$ for at least the half of the cycle or stays at the current node. We consider two cases. First, we consider that a_j stays at the current node in every cycle $c_j^\varepsilon, \dots, c_j^{\varepsilon+\alpha}$. In this case, a_j stays at the current node throughout all cycles $c_j^\varepsilon, \dots, c_j^{\varepsilon+\alpha}$, that is, cycle c_i^γ . Since $|c_i^\gamma| \geq 2 \cdot (t_{\text{REL}}(a_i.id) + 1)$ holds, and a_i visits all nodes in $t_{\text{REL}}(a_i.id)$ rounds after the first round of cycle c_i^γ by Lemma 3.1, a_i and a_j meet by the end of cycle c_i^γ . Next, we consider that a_j executes $\text{REL}(a_j.id)$ in some cycle $c_j^{\varepsilon+\beta}$ ($0 \leq \beta \leq \alpha$). In this case, a_j executes $\text{REL}(a_j.id)$ for at least $(1/2) \cdot |c_j^{\varepsilon+\beta}|$ rounds from the first round of cycle $c_j^{\varepsilon+\beta}$. Since a_j satisfies Line 8 of Algorithm 2 no later than the first round of cycle $c_j^{\varepsilon+\beta}$, $|c_j^{\varepsilon+\beta}| \geq 2 \cdot (t_{\text{REL}}(a_j.id) + 1)$ holds. Thus, a_i and a_j execute $\text{REL}(a_i.id)$ and $\text{REL}(a_j.id)$ at the same time for at least $t_{\text{REL}}(\min(a_i.id, a_j.id))$ rounds. Therefore, a_i and a_j meet by the end of cycle c_i^γ by Lemma 3.1.

Next, we consider the case $|c_i^\gamma| < |c_j^\varepsilon|$. By Observation 4.3, when a_j starts cycle $c_j^{\varepsilon+1}$, a_i starts cycle $c_i^{\gamma'}$ ($\gamma' > \gamma$). Thus, the period of cycle c_i^γ is completely included in the period of cycle c_j^ε . By Lemma 3.1, a_i visits all nodes in $t_{\text{REL}}(a_i.id)$ rounds after the first round of cycle c_i^γ . Therefore, since $|c_i^\gamma| \geq 2 \cdot (t_{\text{REL}}(a_i.id) + 1)$ holds, when a_j stays at the current round throughout the first half of cycle c_i^γ , a_i and a_j meet by the end of cycle c_i^γ . Also, when a_j executes $\text{REL}(a_j.id)$ throughout the first half of cycle c_i^γ , a_i and a_j execute $\text{REL}(a_i.id)$ and $\text{REL}(a_j.id)$ at the same time for at least $t_{\text{REL}}(\min(a_i.id, a_j.id))$ rounds. Thus, a_i and a_j meet by the end of cycle c_i^γ by Lemma 3.1. On the other hand, when a_j is in the *MakeGroup* stage, it may interrupt $\text{REL}(a_j.id)$ in the first half of cycle c_i^γ . In this case, a_j stays at the current node throughout the last half of cycle c_i^γ . During the last half of cycle c_i^γ , a_i keeps executing $\text{REL}(a_i.id)$. Thus, from the above discussion, a_i and a_j meet by the end of cycle c_i^γ . Hence, the lemma holds. \square

In the *CollectID* stage, an agent a_i executes $\text{REL}(a_i.id)$ without interruption during the cycle whose length is at least $2 \cdot (t_{\text{REL}}(a_i.id) + 1)$. Since a_i meets all good agents by the end of the cycle, we have the following corollary.

Corollary 4.1. *For good agent a_i , if $a_i.stage \in \{\text{MakeCandidate}, \text{AgreeID}, \text{MakeGroup}\}$ holds, $a_i.S_p$ contains IDs of all good agents and hence $k = g + f \geq |a_i.S_p| \geq g$ holds.*

By $k \geq 9f + 8$, $g \geq 8f + 8$, and Corollary 4.1, we have the following corollary.

Corollary 4.2. *For good agent a_i , if $a_i.stage \in \{\text{MakeCandidate}, \text{AgreeID}, \text{MakeGroup}\}$ holds, $g > (8/9)k \geq (8/9)|a_i.S_p|$ holds.*

Next, we consider the *MakeCandidate* stage. Let a_{max} be the good agent with the largest ID. Regarding this stage, we clarify the following two facts: (1) All good agents finish the *MakeCandidate* stage in some bounded rounds, and (2) At least $(7/18)g$ good agents start the *AgreeID* stage at the same time. We prove (1) with Lemma 4.2 to Lemma 4.4, and (2) with Lemma 4.5 to Corollary 4.3.

Lemma 4.2. *Let c_{max}^γ be the first cycle of the *MakeCandidate* stage of a_{max} . Every good agent a_i executes $a_i.stage \leftarrow \text{AgreeID}$ by the end of cycle $c_{max}^{\gamma+1}$.*

Proof. First, we prove that every good agent a_i executes $a_i.ready \leftarrow \text{True}$ by the end of the first round of cycle c_{max}^γ . Since a_{max} has the largest ID among good agents, a_{max} starts the *MakeCandidate* stage latest among good agents. Therefore, all good agents start the *MakeCandidate* stage by the time a_{max} starts that. We consider two cases. First, we consider the case that a_i is in the *MakeCandidate* stage at the beginning of cycle c_{max}^γ . By Observation 4.1, $a_i.length = a_{max}.length$ holds in cycle c_{max}^γ and a_i starts its cycle at the same time as a_{max} . Since a_{max} satisfies Line 8 of Algorithm 2 before cycle c_{max}^γ , $a_{max}.length \geq 4 \cdot (t_{\text{REL}}(a_{max}.id) + 1)$ holds at the beginning of cycle c_{max}^γ . Therefore, $a_i.length \geq 4 \cdot (t_{\text{REL}}(a_{max}.id) + 1)$ holds at the beginning of cycle c_{max}^γ . Since $a_i.S_p$ contains IDs of all good agents by Corollary 4.1, it contains at least g IDs no larger than $a_{max}.id$. Since $g > (8/9)|a_i.S_p|$ holds by Corollary 4.2, a_i satisfies Line 3 of Algorithm 3 at the beginning of cycle c_{max}^γ . Thus, a_i executes $a_i.ready \leftarrow \text{True}$ by the end of the first round of cycle c_{max}^γ . Next, we consider the case that a_i is in either the *AgreeID* stage or the *MakeGroup* stage at the beginning of

cycle c_{max}^γ . Since a_i has finished the *MakeCandidate* stage before cycle c_{max}^γ , a_i satisfies Line 8 of Algorithm 3, that is, $|a_i.R| \geq (6/9)|a_i.S_p|$ holds. Therefore, a_i satisfies Line 4 of Algorithm 3 and thus a_i executes $a_i.ready \leftarrow \text{True}$ before cycle c_{max}^γ .

Next, we prove that every good agent a_i executes $a_i.stage \leftarrow \text{AgreeID}$ by the end of cycle $c_{max}^{\gamma+1}$. Consider the situation that good agent a_i has not yet executed $a_i.stage \leftarrow \text{AgreeID}$ at the beginning of cycle $c_{max}^{\gamma+1}$. Let a_j be a good agent. First, we consider the case that $a_j.ready = \text{True}$ holds at the beginning of cycle c_{max}^γ . By Lemma 4.1, since a_i executes $\text{REL}(a_i.id)$ throughout cycle c_{max}^γ , a_i meets a_j by the last round of cycle c_{max}^γ . Thus, $a_i.R$ contains $a_j.id$ before cycle $c_{max}^{\gamma+1}$. Next, we consider the case that $a_j.ready = \text{False}$ holds at the beginning of cycle c_{max}^γ . In this case, a_j executes the *MakeCandidate* stage during cycle c_{max}^γ and executes $a_j.ready \leftarrow \text{True}$ by the end of the first round of the cycle. By Observation 4.1, a_i and a_j start their cycles at the same time and have the same length of their cycles. Since $|c_{max}^\gamma| \geq 4 \cdot (t_{\text{REL}}(a_{max}.id) + 1)$ holds, a_i and a_j execute $\text{REL}(a_i.id)$ and $\text{REL}(a_j.id)$ at the same time for at least $t_{\text{REL}}(\min(a_i.id, a_j.id))$ rounds after the first round of cycle c_{max}^γ . Therefore, by Lemma 3.1, a_i meets a_j at least once after a_j executes $a_j.ready \leftarrow \text{True}$. This implies that $a_i.R$ contains $a_j.id$ before cycle $c_{max}^{\gamma+1}$. From these two cases, $a_i.R$ contains at least g IDs before cycle $c_{max}^{\gamma+1}$. Since $g > (8/9)|a_i.S_p|$ holds by Corollary 4.2, a_i satisfies Line 8 of Algorithm 3 at the beginning of cycle $c_{max}^{\gamma+1}$. Since a_{max} doubles $a_{max}.length$ in the last round of cycle c_{max}^γ , $|c_{max}^{\gamma+1}|$ is identical to the length of a cycle of a good agent that executes the *MakeCandidate* stage in cycle $c_{max}^{\gamma+1}$. Therefore, a_i executes $a_i.stage \leftarrow \text{AgreeID}$ in the last round of cycle $c_{max}^{\gamma+1}$. Hence, the lemma holds. \square

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

Lemma 4.3. *For any good agent a_i , $a_i.length$ is less than $32 \cdot (t_{\text{REL}}(a_{max}.id) + 1)$.*

Proof. Let c_{max}^γ be the first cycle of the *MakeCandidate* stage of a_{max} . Each good agent a_i updates $a_i.length$ in the last round of a cycle of the *CollectID* and *MakeCandidate* stages, but not in the *AgreeID* and *MakeGroup* stages. Also, by Lemma 4.2, a_i executes $a_i.stage \leftarrow \text{AgreeID}$ by the end of cycle $c_{max}^{\gamma+1}$. Thus, to calculate the maximum value of $a_i.length$, it is enough to calculate the value of $a_{last}.length$ at the beginning of cycle $c_{max}^{\gamma+2}$ for a good agent a_{last} that starts the *AgreeID* stage latest. Let $length_{last}^\alpha$ be the value of $a_{last}.length$ at the beginning of cycle c_{max}^α for a positive integer α . When a_{last} starts the *MakeCandidate* stage in cycle c_{max}^γ , a_{last} satisfies Line 8 of Algorithm 2 (i.e., $2 \cdot (t_{\text{REL}}(a_{max}.id) + 1) \leq length_{last}^{\gamma-1}$ holds at the beginning of cycle $c_{max}^{\gamma-1}$). On the other hand, since a_{last} does not satisfy Line 8 of Algorithm 2 at the beginning of cycle $c_{max}^{\gamma-2}$, $2 \cdot (t_{\text{REL}}(a_{max}.id) + 1) > length_{last}^{\gamma-2}$ holds. Hence, $2^4 \cdot 2 \cdot (t_{\text{REL}}(a_{max}.id) + 1) = 32 \cdot (t_{\text{REL}}(a_{max}.id) + 1) > length_{last}^{\gamma+2}$ holds at the beginning of cycle $c_{max}^{\gamma+2}$. \square

Lemma 4.4. *All good agents finish the *CollectID* stage in $O(t_{\text{REL}}(a_{max}.id))$ rounds after starting *MakeReliableGroup*. Furthermore, all good agents finish the *MakeCandidate* stage in $O(t_{\text{REL}}(a_{max}.id))$ rounds after starting *MakeReliableGroup*.*

Proof. Let c_{max}^γ be the first cycle of the *MakeCandidate* stage of a_{max} . By Lemma 4.2, all good agents execute $stage \leftarrow \text{AgreeID}$ by the end of cycle $c_{max}^{\gamma+1}$. Let a_{last} be a good agent that finishes the *MakeCandidate* stage latest. To prove this lemma, it is enough to analyze the time t_{mc} that are required for a_{last} to finish the *MakeCandidate* stage. Agent a_{last} repeats a cycle and its length is $a_{last}.length$. Furthermore, a_{last} doubles $a_{last}.length$ in the last round of a cycle of the *CollectID* and *MakeCandidate* stages. By Lemma 4.3, $|c_{max}^{\gamma+2}| < 32 \cdot (t_{\text{REL}}(a_{max}.id) + 1)$ holds. Thus, since $|c_{max}^\alpha|$ is represented as $2^{\alpha-1} \cdot T_{ini}$ for a positive integer α , t_{mc} is at most $T_{ini} + 2 \cdot T_{ini} + 4 \cdot T_{ini} + \dots + 2^{\gamma+1} \cdot T_{ini} = (2^{\gamma+2} - 1) \cdot T_{ini} = |c_{max}^{\gamma+2}| - T_{ini} < 32 \cdot (t_{\text{REL}}(a_{max}.id) + 1) - T_{ini}$. Therefore, a_{last} finishes the *MakeCandidate* stage in $O(t_{\text{REL}}(a_{max}.id))$ rounds after starting *MakeReliableGroup*. Hence, all good agents finish the *MakeCandidate* stage in $O(t_{\text{REL}}(a_{max}.id))$ rounds after starting *MakeReliableGroup*, which implies that they also finish the *CollectID* stage in $O(t_{\text{REL}}(a_{max}.id))$ rounds. \square

From now on, we will prove that at least $(7/18)g$ good agents start the *AgreeID* stage at the same time. First, we focus on the situation where the first good agent becomes ready to transition into the *AgreeID* stage.

Lemma 4.5. *Let a_i be the first good agent that executes $ready \leftarrow True$. Agent a_i executes $a_i.ready \leftarrow True$ by satisfying Line 3 of Algorithm 3.*

Proof. We prove this lemma by contradiction. Let c_i^γ be the cycle in which a_i executes $a_i.ready \leftarrow True$. At the beginning of cycle c_i^γ , $|a_i.R| \leq f$ holds because only Byzantine agents can execute $ready \leftarrow True$ before cycle c_i^γ . On the other hand, a_i should satisfy $|a_i.R| \geq (4/9)|a_i.S_p|$ at the beginning of cycle c_i^γ to execute $a_i.ready = True$ without satisfying Line 3 of Algorithm 3. Since $k \geq |a_i.S_p| \geq g$ holds by Corollary 4.1, $(4/9)|a_i.S_p| \geq (4/9)g \geq (4/9)(8f+8) = (32/9)f + 32/9 > f$ holds by $g \geq 8f+8$. This is a contradiction. \square

By Lemma 4.5, at least one good agent a_i executes $a_i.ready \leftarrow True$ by satisfying Line 3 of Algorithm 3. In the following lemma, we check S_p of good agents that execute the *MakeCandidate* stage at the beginning of the cycle when a_i executes $a_i.ready \leftarrow True$.

Lemma 4.6. *Let a_i be the first good agent that executes $ready \leftarrow True$ and c_i^γ be the cycle in which a_i executes that. Let a_j be a good agent that is in the *MakeCandidate* stage in cycle c_i^γ . At the beginning of cycle c_i^γ , $a_j.S_p$ contains at least $(7/9)|a_j.S_p|$ IDs of good agents that started the *MakeCandidate* stage by the beginning of cycle c_i^γ .*

Proof. By Corollary 4.1, $a_i.S_p$ and $a_j.S_p$ contain IDs of all good agents at the beginning of cycle c_i^γ . Therefore, let f_i (resp. f_j) be the number of IDs of Byzantine agents in $a_i.S_p$ (resp. $a_j.S_p$), and then $|a_i.S_p| = g + f_i$ (resp. $|a_j.S_p| = g + f_j$) holds. By Lemma 4.5, at the beginning of cycle c_i^γ , $a_i.S_p$ contains at least $(8/9)|a_i.S_p|$ IDs of agents that started the *MakeCandidate* stage by the beginning of cycle c_i^γ . Thus, $a_i.S_p$ contains at least $(8/9)|a_i.S_p| - f_i$ IDs of good agents. By $f \geq f_i$, $k \geq 9f+8$, and $g \geq 8f+8$, $(8/9)|a_i.S_p| - f_i = (8/9)(g + f_i) - f_i = (1/9)(8g + 8f_i - 9f_i) = (1/9)(8g - f_i) \geq (1/9)(8g - f) = (1/9)(7g + g - f) \geq (1/9)(7g + 8f + 8 - f) = (1/9)(7g + 7f + 8) > (7/9)(g + f)$ holds. Therefore, by $f \geq f_j$, at the beginning of cycle c_i^γ , $a_j.S_p$ contains at least $(7/9)(g + f) \geq (7/9)(g + f_j) = (7/9)|a_j.S_p|$ IDs of good agents that started the *MakeCandidate* stage by the beginning of cycle c_i^γ . \square

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

Lemma 4.7. *Let a_i be a good agent that executes $endMakeCandidate \leftarrow True$, and c_i^γ be the cycle in which a_i executes that. Let a_j be a good agent that is in the *MakeCandidate* stage in cycle c_i^γ . At the beginning of cycle c_i^γ , $a_j.R$ contains at least $(4/9)|a_j.S_p|$ IDs of good agents.*

Proof. At the beginning of cycle c_i^γ , $a_i.R$ contains at least $(6/9)|a_i.S_p|$ IDs of agents. Thus, $a_i.R$ contains at least $(6/9)|a_i.S_p| - f$ IDs of good agents. Since $k \geq |a_i.S_p| \geq g$ holds by Corollary 4.1, $(6/9)|a_i.S_p| - f \geq (6/9)g - f$ holds. By $g \geq 8f+8$, $(6/9)g - f = (1/9)(4g + 2g - 9f) > (1/9)(4g + 16f - 9f) = (1/9)(4g + 7f) > (4/9)(g + f)$ holds. Since $k \geq |a_j.S_p| \geq g$ holds, $a_j.R$ contains at least $(4/9)(g + f) \geq (4/9)|a_j.S_p|$ IDs of agents at the beginning of cycle c_i^γ . \square

By Lemma 4.2, every good agent starts the *AgreeID* stage by the time a_{max} starts that. In the following lemma, we show that several good agents start the *AgreeID* stage at the same time.

Lemma 4.8. *Let a_{ini} be the first good agent that starts the *AgreeID* stage and c_{ini}^γ be a cycle in which a_{ini} starts the *AgreeID* stage. At least $(7/9)g$ good agents start the *AgreeID* stage in the first round of cycle c_{ini}^γ or cycle $c_{ini}^{\gamma+1}$.*

Proof. First, we prove that at least $(7/9)g$ good agents have started the *MakeCandidate* stage before cycle $c_{ini}^{\gamma-2}$. By the assumption of a_{ini} , every good agent executes the *CollectID* stage or the *MakeCandidate* stage at the beginning of cycle $c_{ini}^{\gamma-1}$. Thus, by Observation 4.1, every good agent has the same *length* and starts its cycle at the same time at the beginning of cycle $c_{ini}^{\gamma-1}$. Since a_{ini} starts the *AgreeID* stage at the beginning of cycle c_{ini}^γ , a_{ini} satisfies Line 8 of Algorithm 3 at the beginning of cycle $c_{ini}^{\gamma-1}$. That is, $|a_{ini}.R| \geq (6/9)|a_{ini}.S_p|$ holds. Since $k \geq |a_{ini}.S_p| \geq g$ holds by Corollary 4.1, $|a_{ini}.R| \geq (6/9)|a_{ini}.S_p| \geq (6/9)g \geq (6/9)(8f+8)$ holds by $g \geq 8f+8$. Since f Byzantine agents exist in the network, at least $(6/9)(8f+8) - f > 1$ good agents execute $ready \leftarrow True$

in cycle $c_{ini}^{\gamma-1}$ or earlier. This implies, by Lemma 4.6, that $a_{ini}.S_p$ contains at least $(7/9)|a_{ini}.S_p|$ IDs of good agents that have started the *MakeCandidate* stage before cycle $c_{ini}^{\gamma-2}$. That is, at least $(7/9)|a_{ini}.S_p| \geq (7/9)g$ good agents have started the *MakeCandidate* stage before cycle $c_{ini}^{\gamma-2}$.

Let A_{em} be a set of good agents that are in the *MakeCandidate* stage in the first round of cycle $c_{ini}^{\gamma-1}$. From the above discussion, $|A_{em}| \geq (7/9)g$ holds. The set A_{em} are divided into the following two sets A_1 and A_2 : A_1 is a set of agents that satisfies Line 8 of Algorithm 3 in the first round of cycle $c_{ini}^{\gamma-1}$, and $A_2 = A_{em} \setminus A_1$. Note that agents in A_1 start the *AgreeID* stage in the first round of cycle c_{ini}^{γ} . Consider an arbitrary agent a_{em} in A_{em} . Since a_{ini} executes $a_{ini}.endMakeCandidate \leftarrow True$ in the first round of cycle $c_{ini}^{\gamma-1}$, $|a_{em}.R|$ contains at least $(4/9)|a_{em}.S_p|$ IDs at the beginning of cycle $c_{ini}^{\gamma-1}$ by Lemma 4.7. Therefore, a_{em} satisfies Line 4 of Algorithm 3 and executes $a_{em}.ready \leftarrow True$ in the first round of cycle $c_{ini}^{\gamma-1}$ if $a_{em}.ready = False$ holds. That is, $a_{em}.ready = True$ holds in the first round of cycle $c_{ini}^{\gamma-1}$. Consider an agent a_i in A_2 . Since a_i meets all good agents by the end of cycle $c_{ini}^{\gamma-1}$ by Lemma 4.1, $|a_i.R|$ contains at least $|A_{em}| \geq (7/9)g$ IDs at the beginning of cycle c_{ini}^{γ} . Since $g > (8/9)k$ by Corollary 4.2, $(7/9)g > (7/9) \cdot (8/9)k = (56/81)k > (6/9)k$ holds. Thus, a_i satisfies Line 8 of Algorithm 3 in the first round of cycle c_{ini}^{γ} and executes $a_i.stage \leftarrow AgreeID$ in the last round of cycle c_{ini}^{γ} . Therefore, agents in A_1 start the *AgreeID* stage at the beginning of c_{ini}^{γ} and agents in A_2 start that at the beginning of $c_{ini}^{\gamma+1}$. Hence, at least $(7/9)g$ good agents start the *AgreeID* stage at the beginning of cycle c_{ini}^{γ} or cycle $c_{ini}^{\gamma+1}$. \square

By Lemma 4.8, we have the following corollary.

Corollary 4.3. *At least $(7/18)g$ good agents start the *AgreeID* stage at the same time.*

Next, we consider the *AgreeID* stage. Regarding this stage, we check P_p of good agents in the *AgreeID* stage and the execution of PCONS for both S_p and P_p .

Lemma 4.9. *Let GC be a group candidate, a_i be a good agent in GC , and c_i^{γ} be a cycle in which a_i starts the *AgreeID* stage. Variable $a_i.P_p$ contains at least IDs of all good agents in GC at the beginning of cycle $c_i^{\gamma+1}$.*

Proof. Since $a_i.count = 0$ holds at the beginning of cycle c_i^{γ} by the behavior of *MakeReliableGroup*, a_i satisfies Line 1 of Algorithm 4 at the beginning of cycle c_i^{γ} and thus starts collecting IDs of agents in GC using $REL(a_i.id)$. All good agents in GC have the same length of a cycle by Observation 4.2. Also, since all good agents in GC start the *AgreeID* stage at the same time by Definition 4.1, they execute $stage \leftarrow AgreeID$ before cycle c_i^{γ} . Therefore, since a_i meets all good agents by the end of cycle c_i^{γ} by Lemma 4.1, $a_i.P_p$ contains at least IDs of all good agents in GC at the beginning of cycle $c_i^{\gamma+1}$. \square

Lemma 4.10. *Let GC be a group candidate, a_i be a good agent in GC , and c_i^{γ} be a cycle in which a_i starts the *AgreeID* stage. If at least $(7/18)g$ good agents belong to GC , all good agents in GC start the *MakeGroup* stage in $O(f)$ cycles after cycle c_i^{γ} . Furthermore, the executions of both $PCONS(a_i.S_p)$ and $PCONS(a_i.P_p)$ satisfy the PBC property.*

Proof. First, we show that all good agents in GC can simulate PCONS. Algorithm *MakeReliableGroup* tries to realize one phase in the Byzantine synchronous message-passing model by one cycle. By Observations 4.2 and 4.4, all good agents in GC have the same length of a cycle and start a cycle at the same time. By the behavior of *MakeReliableGroup*, a_i executes $REL(a_i.id)$ every cycle of the *AgreeID* stage. Also, by Lemma 4.1, a_i meets all good agents in GC by the end of the cycle. Thus, for a message msg that a good agent s in GC send in a phase, its destination agent d receives msg from s by hand when they meet during the phase and d knows the ID of s at that time. This behavior follows the definition of the synchronous message-passing model that PCONS assumes. Hence, all good agents in GC can simulate PCONS.

Next, we prove this lemma. By $g \geq 8f + 8$, $(7/18)g \geq (7/18)(8f + 8) = (56/18)f + 56/18 > 3f$ holds. Since all good agents in GC can simulate PCONS and $(7/18)g > 3f$ holds, by Lemma 3.2, the executions of both $PCONS(a_i.S_p)$ and $PCONS(a_i.P_p)$ satisfy the PBC property and a_i finishes both $PCONS(a_i.S_p)$ and $PCONS(a_i.P_p)$ in $O(f)$ cycles after cycle c_i^{γ} . Thus, a_i executes $a_i.stage \leftarrow$

MakeGroup in the last round of the cycle in which a_i finishes both $\text{PCONS}(a_i.S_p)$ and $\text{PCONS}(a_i.P_p)$. Hence, this lemma holds. \square

For a good agent a_i , at the beginning of the second cycle of the *AgreeID* stage, $a_i.P_p$ contains all IDs of good agents in the same group candidate by Lemma 4.9, but does not contain IDs of the other good agents by the behavior of *MakeReliableGroup*. By Lemma 4.10, since the execution of $\text{PCONS}(a_i.P_p)$ satisfies the PBC property, by Validity 1 and Validity 2 of the PBC property, $a_i.P_c$ contains all IDs of good agents in the same group candidate but not IDs of the other good agents. From this discussion, Lemma 4.10 and Corollary 4.1, we have the following corollary.

Corollary 4.4. *Let GC be a group candidate, a_i be a good agent in GC , and c_i^γ be the first cycle in which all good agents in GC are in the *MakeGroup* stage. If at least $(7/18)g$ good agents belong to GC , all good agents in GC have the same P_c and S_c at the beginning of cycle c_i^γ . For each good agent a_i in GC , $|a_i.S_c| \geq g$ and $|a_i.P_c| \geq (7/18)g$ hold. Also, $a_i.S_c$ contains all IDs of good agents, and $a_i.P_c$ contains all IDs of good agents in GC but not IDs of the other good agents.*

Next, we consider the *MakeGroup* stage. In the following two lemmas, we prove that when there exists at least one group candidate consisting of at least $(7/18)g$ good agents, at least one reliable group is created.

Lemma 4.11. *Let GC be a group candidate, a_i be a good agent in GC , and c_i^γ be a cycle in which all good agents in GC are in the *MakeGroup* stage. If at least $(7/18)g$ good agents belong to GC , a_i stores a group ID in $a_i.gid$ within $f + 1$ cycles after cycle c_i^γ .*

Proof. If a_i has stored a group ID in $a_i.gid$ before cycle c_i^γ , the lemma clearly holds. Therefore, we consider the case where a_i has not stored a group ID in gid before cycle c_i^γ .

First, we prove that all good agents in GC decide at least one same ID of a good agent in GC as a target ID within $f + 1$ cycles after cycle c_i^γ . By Corollary 4.4, since at least $(7/18)g$ good agents belong to GC , for each good agent a_i in GC , $|a_i.P_c| \geq (7/18)g$ holds at the beginning of cycle c_i^γ . Since $(7/18)g \geq (7/18)(8f + 8) > f + 1$ holds by $g \geq 8f + 8$, and agent a_i in GC increments $a_i.count$ by one in the last round of every cycle, a_i uses different $f + 1$ IDs in $a_i.P_c$ as target IDs during $f + 1$ cycles starting from cycle c_i^γ . Hence, since f Byzantine agents exist in the network, the IDs contain at least one ID of a good agent in $a_i.P_c$. Since all good agents in GC start the *AgreeID* stage at the same time by Definition 4.1, and they start a cycle at the same time by Observation 4.2, all good agents in GC have the same *count*. By Corollary 4.4, all good agents in GC have the same P_c , and $a_i.P_c$ contains all IDs of good agents in GC but not IDs of the other good agents. Thus, all good agents in GC decide the same agent ID in P_c of them as a target ID in every cycle from cycle c_i^γ . Hence, all good agents in GC decide at least one same ID of a good agent in GC as a target ID within $f + 1$ cycles after cycle c_i^γ .

Next, let c_i^ε be the first cycle that a good agent in GC has a target ID, a_{gt} be the good agent that has the target ID in cycle c_i^ε , and a_{gm} be the good agent with the largest ID of good agents in GC . We prove that all good agents in GC gather at a single node by the last round of cycle c_i^ε . Agent a_{gt} stays at the current node throughout cycle c_i^ε . By the behavior of *MakeReliableGroup*, an agent searches for an agent with a target ID in the last half of a cycle. Since a_{gm} satisfies Line 8 of Algorithm 2, and all good agents in GC have the same length of a cycle by Observation 4.2, $|c_i^\gamma| \geq 2 \cdot (t_{REL}(a_{gm}.id) + 1)$ holds. On the other hand, by Lemma 3.1, a_i visits all nodes during $t_{REL}(a_i.id)$ rounds. Therefore, a_i meets a_{gt} during the last half of cycle c_i^ε . Hence, all good agents in GC gather at a single node by the last round of cycle c_i^ε .

Finally, we prove that a_i stores a group ID in $a_i.gid$ when all good agents in GC gather at a single node. All good agents in GC have started the *MakeGroup* stage. By Observation 4.2, all good agents in GC have the same *length*. By Corollary 4.4, all good agents in GC have the same S_c and, for each good agent a_i in GC , $|a_i.S_c| \geq g$ holds at the beginning of cycle c_i^γ . Since $g > (8/9)k \geq (8/9)|a_i.S_p|$ holds by Corollary 4.2, $|a_i.S_c| \geq g \geq (8/9)|a_i.S_p|$ holds. For every good agent a_j in GC other than a_i , since $g > (8/9)|a_i.S_p|$ and $|a_j.S_c| = |a_i.S_c| \geq g$ hold, $|a_j.S_c| \geq g \geq (8/9)|a_j.S_p|$ holds. Therefore, $a_i.D$ contains at least IDs of all good agents in GC . Since $(7/18)g = (6/18)g + (1/18)g \geq (6/18)g + (1/18)(8f + 8) = (6/18)g + (8/18)f + 8/18 > (6/18)(g + f) = (3/9)k$ holds by $g \geq 8f + 8$,

$|a_i.D| > (3/9)k \geq (3/9)|a_i.S_c|$ holds. Thus, a_i satisfies Line 13 of Algorithm 5 and thus stores a group ID in $a_i.gid$ when all good agents in GC gather at a single node.

Hence, this lemma holds. \square

Lemma 4.12. *Let a_i be a good agent. When a_i stores a group ID in $a_i.gid$, at least $k/8$ good agents store the same group ID as a_i in their gid .*

Proof. First, we prove that $a_i.D$ includes at least $k/8$ IDs of good agents. By the behavior of **MakeReliableGroup**, a_i satisfies Line 13 of Algorithm 5. Since $k \geq |a_i.S_p| \geq g$ holds by Corollary 4.1, $|a_i.S_c| \geq (8/9)k \geq (8/9)|a_i.S_p| \geq (8/9)g$ holds. Therefore, $|a_i.D| \geq (3/9)|a_i.S_c| \geq (3/9)(8/9)g = (8/27)g$ holds. Since f Byzantine agents exist in the network, at least $(8/27)g - f$ of them are good agents. By $g \geq 8f + 8$, $(8/27)g - f = (27/216)g + (37/216)g - f \geq (27/216)g + (37/216)(8f + 8) - f = (27/216)g + (80/216)f + 296/216 > (27/216)(g + f) = (1/8)(g + f) = k/8$ holds. Hence, $a_i.D$ includes at least $k/8$ IDs of good agents.

Next, we prove that at least $k/8$ good agents in $a_i.D$ store the same group ID as a_i in their gid when a_i stores a group ID in $a_i.gid$. Let a_j be a good agent in $a_i.D$ other than a_i . Since a_i executes $a_i.D \leftarrow a_i.D \cup \{a_j.id\}$ by satisfying Line 12 of Algorithm 5, $|a_j.S_c| \geq (8/9)|a_j.S_p|$, $a_i.length = a_j.length$, $|a_i.S_c| = |a_j.S_c|$, and $a_j.stage = \text{MakeGroup}$ hold. Since a_i and a_j observe the same states of agents at the current node, $a_i.D = a_j.D$ holds. Since $|a_i.D| \geq (3/9)|a_i.S_c|$, $a_i.D = a_j.D$, and $|a_i.S_c| = |a_j.S_c|$ hold, $|a_j.D| \geq (3/9)|a_j.S_c|$ holds. Therefore, a_j satisfies Line 13 of Algorithm 5 and thus stores the same group ID in $a_j.gid$.

Hence, this lemma holds. \square

Finally, we prove the complexity of **MakeReliableGroup**.

Theorem 4.1. *Let n be the number of nodes, k be the number of agents, g be the number of good agents, f be the number of weakly Byzantine agents, 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 1 makes good agents create at least one reliable group in $O(f \cdot t_{REL}(a_{max}.id))$ rounds.*

Proof. By Lemma 4.4, all good agents finish the *MakeCandidate* stage in $O(t_{REL}(a_{max}.id))$ rounds after starting **MakeReliableGroup**. By Lemma 4.8 and Corollary 4.3, in $O(t_{REL}(a_{max}.id))$ rounds after starting **MakeReliableGroup**, there is at least one round that at least $(7/18)g$ good agents start the *AgreeID* stage at the same time. Let GC be a group candidate of at least $(7/18)g$ good agents, c_{gc}^ζ be a cycle that all good agents in GC start the *AgreeID* stage, and c_{gc}^η be the first cycle in which all good agents in GC are in the *MakeGroup* stage. By Lemma 4.10, all good agents in GC have started the *MakeGroup* stage in $O(f)$ cycles after cycle c_{gc}^ζ . By Lemma 4.11, a good agent in GC stores a group ID in its gid within $f + 1$ cycles after cycle c_i^η . Also, by Lemma 4.12, when a good agent in GC stores a group ID in its gid , at least $k/8$ good agents store the same group ID as the agent in their gid . That is, a reliable group is created within $f + 1$ cycles after cycle c_i^η . Therefore, since the maximum length of the cycles is at most $32 \cdot (t_{REL}(a_{max}.id) + 1)$ by Lemma 4.3, a reliable group is created in $32 \cdot (t_{REL}(a_{max}.id) + 1) \cdot O(f) + 32 \cdot (t_{REL}(a_{max}.id) + 1)(f + 1) = O(f \cdot t_{REL}(a_{max}.id))$ rounds after starting cycle c_{gc}^ζ . Hence, since the first round of cycle c_{gc}^ζ is within $O(t_{REL}(a_{max}.id))$ rounds after starting **MakeReliableGroup**, a reliable group is created in $O(t_{REL}(a_{max}.id)) + O(f \cdot t_{REL}(a_{max}.id)) = O(f \cdot t_{REL}(a_{max}.id))$ after starting **MakeReliableGroup**. Hence, this theorem holds. \square

4.3 Gathering Algorithm

In this section, we propose an algorithm that solves the gathering problem by assuming that $k = g + f \geq 9f + 8$. The proposed algorithm uses **MakeReliableGroup** described in Section 4.2 to create at least one reliable group. Recall that agents know N , but do not know n, k, f or F .

Algorithm 6 ByzantineGathering(N) for agent a_i

```
1: if  $a_i.stage = CollectID$  then
2:   Execute MakeReliableGroup
3: else  $\triangleright a_i.stage \in \{MakeCandidate, AgreeID, MakeGroup\}$ 
4:    $a_i.S_{gid} \leftarrow \{x \mid \exists A_{rg} \subset A_i[|A_{rg}| \geq (1/8)|a_i.S_p| \wedge \forall a_j \in A_{rg} : a_j.gid = x]\}$ 
5:   if  $a_i.S_{gid} \neq \emptyset$  then
6:      $a_i.minGID \leftarrow \min(a_i.S_{gid})$ 
7:   end if
8:   if  $a_i.S_{gid} \neq \emptyset \wedge a_i.gid > a_i.minGID$  then
9:      $a_i.S_{rg} \leftarrow \{id \mid \exists a_j \in A_i[a_j.gid = a_i.minGID \wedge a_j.id = id]\}$ 
10:    Execute FOLLOW( $a_i.S_{rg}$ )
11:   else if  $a_i.gid \neq \infty$  then
12:      $a_i.elapsed \leftarrow a_i.elapsed + 1$ 
13:     if  $a_i.length = a_i.elapsed$  then
14:       Execute TERMINATE()
15:     end if
16:     Execute REL( $a_i.gid$ )( $a_i.elapsed$ )
17:   else
18:     Execute MakeReliableGroup
19:   end if
20: end if
```

4.3.1 Description of the Algorithm

Algorithm 6 shows the behavior of each round of Algorithm **ByzantineGathering**. The proposed algorithm aims to make all good agents transition into the terminal state at the same node using a reliable group. In **ByzantineGathering**, we introduce procedures **TERMINATE()** and **FOLLOW(S_{rg})**. Procedure **TERMINATE()** means that an agent transitions into the terminal state. Procedure **FOLLOW(S_{rg})** means that an agent a_i executes the following two actions: (1) When a majority of agents in S_{rg} move to some node, a_i also moves to the node. (2) When a majority of agents in S_{rg} execute **TERMINATE()** or have entered a terminal state, a_i executes **TERMINATE()**. Agent a_i can refer to the variables used in **MakeReliableGroup** at any time.

First, we provide the overall execution of **ByzantineGathering**. 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 a reliable group in round r , a good agent a_ℓ in the reliable group executes **REL($a_\ell.gid$)** for $a_\ell.length$ rounds after round $r + 1$. Then, when a good agent meets the reliable group with a smaller group ID, it accompanies the group. By Lemma 4.12, when a good agent stores a group ID in its gid , at least $k/8$ good agents store the same group ID in their gid . Thus, a reliable group contains at least $k/8$ good agents. Since $k \geq |a_i.S_p| \geq g$ holds by Corollary 4.1, $k/8 \geq (1/8)|a_i.S_p| \geq g/8$ holds. Therefore, if a reliable group contains at least one good agent and exists at the node with a_i , a_i can recognize the group. On the other hand, by $g \geq 8f + 8$, $(1/8)|a_i.S_p| \geq g/8 \geq f + 1$ holds and thus only f Byzantine agents are not enough to create a reliable group. Thus, if a group of only f Byzantine agents exists at the node with a_i , a_i does not recognize the group as a reliable group. Summarizing the above, we have the following observation.

Observation 4.5. *If a good agent meets a reliable group consisting of at least $k/8$ good agents with the same group ID, the agent recognizes the group as a reliable group, otherwise, the agent does not recognize the group as a reliable group.*

Furthermore, a reliable group contains at least $k/8$ good agents, by $k \geq 9f + 8$, $k/8 \geq (9/8)f + 1 > f + 1$ holds. That is, a reliable group contains more than $f + 1$ good agents. Thus, good agents are the majority of agents in a reliable group even if the group contains f Byzantine agents. Therefore, when a good agent meets a reliable group, it can trust the group. By the behavior of **MakeReliableGroup**,

round r is the last round of some cycle of the *MakeGroup* stage, and thus round $r + 1$ is the first round of the next cycle of that. All good agents in the same reliable group have the same length of their cycles, and thus the execution of $\text{REL}(a_\ell.\text{gid})$ can be regarded as the execution of an additional cycle of the *MakeGroup* stage by a_ℓ . The following observation summarizes this discussion.

Observation 4.6. *When a good agent a_i belongs to a reliable group, the start round and length of the execution of $\text{REL}(a_i.\text{gid})$ are the same as those of the last cycle in the *MakeGroup* stage of a_i .*

Therefore, by Lemma 4.1, when a_ℓ executes $\text{REL}(a_\ell.\text{gid})$ for $a_\ell.\text{length}$ rounds after round $r + 1$, a_ℓ meets all good agents not in the same reliable group. Consequently, all good agents accompany the reliable group with the smallest ID and achieve the gathering.

Hereinafter, we explain the detailed behaviors of *ByzantineGathering*. At the beginning of every round, an agent a_i determines its action depending on the states of a_i and other agents at the current node.

If $a_i.\text{stage} = \text{CollectID}$ holds, it executes *MakeReliableGroup* until it collects IDs of all good agents (Lines 1–2 of Algorithm 6).

If $a_i.\text{stage} \in \{\text{MakeCandidate}, \text{AgreeID}, \text{MakeGroup}\}$ holds, a_i determines whether the current node contains a reliable group or not. First, if a_i finds groups with at least $(1/8)|a_i.S_p|$ agents, it stores the group IDs to variable $a_i.S_{gid}$ (Line 4). After that, if $a_i.S_{gid}$ contains at least one group ID, a_i calculates the smallest group ID among $a_i.S_{gid}$ and stores the smallest ID in variable minGID (Line 5–7). Initially, $a_i.S_{gid} = \emptyset$ and $a_i.\text{minGID} = \infty$ hold.

Consider the case where $a_i.S_{gid}$ contains at least one group ID and $a_i.\text{minGID}$ is smaller than $a_i.\text{gid}$ (Line 8). This implies one of the following two conditions.

- (1) Agent a_i is not a member of a reliable group and meets a reliable group at the current node.
- (2) Agent a_i is a member of a reliable group and meets a reliable group with a group ID smaller than $a_i.\text{gid}$ at the current node.

In this case, a_i finds agents with $\text{gid} = a_i.\text{minGID}$ and stores their IDs in variable $a_i.S_{rg}$. Then, a_i follows the action of the majority of agents in $a_i.S_{rg}$ (Lines 8–10).

If a_i is a member of a reliable group with the smallest group ID at the current node, it executes $\text{REL}(a_i.\text{gid})$ to meet other good agents and then transitions into a terminal state (Lines 11–16). Note that, if a_i meets a reliable group with a group ID smaller than $a_i.\text{gid}$ during the execution, it follows the group as in the previous paragraph. When a_i executes $\text{REL}(a_i.\text{gid})$ for $a_i.\text{length}$ rounds, by Observation 4.6, the execution and a cycle of the *MakeGroup* stage in *MakeReliableGroup* start and finish at the same time. Thus, by Lemma 4.1, a_i meets all good agents.

If a_i does not satisfy the above conditions (i.e., the current node does not contain a reliable group), it executes *MakeReliableGroup* for one round (Lines 17–19).

4.3.2 Correctness and Complexity Analysis

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

By the behavior of *ByzantineGathering*, an agent executes *MakeReliableGroup* until the current node contains a reliable group. By Theorem 4.1, agents create at least one reliable group after starting *ByzantineGathering*. Therefore, we prove that, after a reliable group is created, all good agents gather at a single node using a reliable group and transition into a terminal state.

Let r_{ini} be the round in which the first reliable group is created, SRG_{ini} be a set of the reliable groups created in round r_{ini} , and RG_{min} be the reliable group with the smallest group ID among SRG_{ini} . Let A_{ec} be a set of good agents that execute the *CollectID* stage in round $r_{ini} + 1$. Let A_{fc} be a set of good agents that have finished the *CollectID* stage and do not belong to a reliable group in round $r_{ini} + 1$. For a good agent $a_i \in A_{ec} \cup A_{fc}$, let c_i^* be a cycle of a_i that includes round $r_{ini} + 1$ and round r_i^* be the last round of cycle c_i^* . For a good agent a_j in some reliable group of SRG_{ini} , let round r_j^* be the last round of the execution of $\text{REL}(a_j.\text{gid})$. Let $r^* = \max(\{r_i^* \mid a_i \text{ is a good agent in } MA\})$. By the behavior of *MakeReliableGroup*, since an agent extends the length of its cycle in the *CollectID* and *MakeCandidate* stages but not in the *AgreeID* and *MakeGroup* stages,

good agents in the *CollectID* and *MakeCandidate* stages have the longest cycles. By Observation 4.6, when a good agent a_j belongs to a reliable group in SRG_{ini} , the length of the execution of $REL(a_j.gid)$ is the same as that of the last cycle in the *MakeGroup* stage of a_j . Thus, by Observation 4.3, when good agents in the *CollectID* and *MakeCandidate* stages start a cycle, a_j and good agents in the *AgreeID* and *MakeGroup* stages also start executing $REL(a_j.gid)$ and a cycle. Therefore, the period of a cycle of good agents in the *CollectID* and *MakeCandidate* stages includes the period of a cycle of good agents in the *AgreeID* and *MakeGroup* stages and the execution of $REL(a_j.gid)$. By Observation 4.1, good agents in the *CollectID* and *MakeCandidate* stages have the same length of their cycles. Hence, if A_{ec} includes at least one good agent, r^* is the last round of the cycle of an agent in A_{ec} . From this discussion, we have the following observation.

Observation 4.7. *If A_{ec} includes at least one good agent, r^* is the last round of cycle c_i^* for some agent $a_i \in A_{ec}$.*

In the following lemma, we prove that a good agent a_i in A_{fc} and RG_{min} meet if a_i and good agents in RG_{min} never interrupt *MakeReliableGroup* and *REL* with the group ID in any round before they meet.

Lemma 4.13. *Assume that a good agent a_i in A_{fc} executes *MakeReliableGroup* until the last round of cycle c_i^* , and every good agent a_j in RG_{min} executes $REL(a_j.gid)$ for $a_j.length$ rounds without interruption after round $r_{ini} + 1$. In this case, a_i and RG_{min} meet before the last round of cycle c_i^* .*

Proof. First, we consider the case that the first round of cycle c_i^* is round $r_{ini} + 1$. In this case, by the behavior of *MakeReliableGroup*, a_i executes $REL(a_i.id)$ for at least the first half of cycle c_i^* . Since a_i and a_j satisfy Line 8 of Algorithm 2 before round r_{ini} , $|c_i^*| \geq 2 \cdot (t_{REL}(a_i.id) + 1)$ and $a_j.length \geq 2 \cdot (t_{REL}(a_j.id) + 1)$ hold. Since $a_j.id \geq a_j.gid$ holds by the behavior of *MakeReliableGroup*, $a_j.length \geq 2 \cdot (t_{REL}(a_j.gid) + 1)$ holds. Thus, a_i and a_j execute $REL(a_i.id)$ and $REL(a_j.gid)$ at the same time for at least $t_{REL}(\min(a_i.id, a_j.gid))$ rounds. Also, by Definition 4.2, all good agents in RG_{min} have the same *gid* and *length*. Thus, they behave identically during the execution of $REL(a_j.gid)$. Therefore, by Lemma 3.1, a_i meets RG_{min} in at least $t_{REL}(\min(a_i.id, a_j.gid))$ rounds after round $r_{ini} + 1$, that is, before the last round of cycle c_i^* .

Next, we consider the case that the first round of cycle c_i^* is not round $r_{ini} + 1$. By Observations 4.3 and 4.6, if $|c_i^*| \leq a_j.length$ holds, when a_j starts executing $REL(a_j.gid)$, a_i also starts cycle c_i^* . That is, this case does not apply if $|c_i^*| \leq a_j.length$ holds. Thus, this case implies that $|c_i^*| > a_j.length$ holds. By Observations 4.3 and 4.6, the execution of $REL(a_j.gid)$ is completely included in the period of cycle c_i^* . Therefore, by Lemma 4.1, a_j meets a_i by the last round of execution of $REL(a_j.gid)$. Since all good agents in RG_{min} behave identically during the execution of $REL(a_j.gid)$, a_i and RG_{min} meet before the last round of the execution of $REL(a_j.gid)$. \square

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

Lemma 4.14. *Assume that at least one reliable group exists at a node v in round r' . Let RG' be the reliable group with the smallest group ID at v in round r' . Then, all good agents not in RG' at v follow RG' in round r' .*

Proof. Let a_i be a good agent not in RG' at v in round r' . Let C' be a set of all group IDs of reliable groups at v in round r' . By Observation 4.5, a_i recognizes all reliable groups at v . Thus, a_i executes $a_i.S_{gid} \leftarrow C'$, and then a_i executes $a_i.minGID \leftarrow \min(a_i.S_{gid})$. This implies that a_i executes $a_i.minGID \leftarrow \min(C')$. By the assumption of this lemma, since $a_i.S_{gid} \neq \emptyset$ and $a_i.gid > a_i.minGID$ hold in round r'_i , a_i calculates $a_i.S_{rg}$ and executes *FOLLOW*($a_i.S_{rg}$). Note that $a_i.S_{rg}$ contains all good agents in RG' . Recall that all good agents in RG' make the same behavior and the number of good agents in RG' is at least $f + 1$ by Definition 4.2, Lemma 4.12, and $k \geq 9f + 8$. Hence, a_i follows RG' . \square

In the following lemma, we prove that agents do not create a reliable group between round $r_{ini} + 1$ and round r^* inclusive.

Lemma 4.15. *Assume that every good agent a_j in RG_{min} executes $REL(a_j.gid)$ for $a_j.length$ rounds without interruption after round $r_{ini} + 1$. No reliable group is created between round $r_{ini} + 1$ and round r^* inclusive.*

Proof. We prove this lemma by contradiction. Assume that a reliable group is created in round r' ($r^* \geq r' \geq r_{ini} + 1$) and no reliable group is created between round $r_{ini} + 1$ and round $r' - 1$ inclusive. By Observation 4.7, if A_{ec} includes at least one good agent, an agent in A_{ec} starts the *MakeCandidate* stage after round r^* and hence it cannot participate in the creation of a reliable group. Thus, it is sufficient to consider only agents in A_{fc} . By the behavior of *ByzantineGathering*, each agent $a_i \in A_{fc}$ behaves according to *MakeReliableGroup* until the current node contains a reliable group. On the other hand, by Lemma 4.14, if a_i meets a reliable group, it follows the reliable group and hence cannot participate in the creation of a reliable group. Therefore, to derive a contradiction, it is enough to prove that a_i meets some reliable group before round r^* . However, if a_i continues *MakeReliableGroup* without meeting a reliable group, by Lemma 4.13, a_i and RG_{min} meet before the last round of cycle c_i^* . Since round r' is the last round of cycle c_i^* or later by the behavior of *MakeReliableGroup*, and $r^* \geq r'$ holds, a_i meets RG_{min} before round r^* . This is a contradiction. Hence, this lemma holds. \square

In the following lemma, we prove that good agents in RG_{min} never interrupt *REL*.

Lemma 4.16. *Every good agent a_i in RG_{min} executes $REL(a_i.gid)$ for $a_i.length$ rounds without interruption after round $r_{ini} + 1$ and then transitions into a terminal state.*

Proof. By the behavior of *ByzantineGathering*, a_i interrupts $REL(a_i.gid)$ if it meets a reliable group with a smaller group ID. However, by Lemma 4.15, no reliable group is created between round $r_{ini} + 1$ and round r^* inclusive, and hence the group ID of RG_{min} is the smallest until round r^* . This implies that, since $r^* \geq r_{ini} + a_i.length$ holds, a_i executes $REL(a_i.gid)$ for $a_i.length$ rounds without interruption. \square

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

Lemma 4.17. *Let RG be a reliable group in $SRG_{ini} \setminus \{RG_{min}\}$. Then, all good agents in RG meet a reliable group with a group ID smaller than RG before round r^* .*

Proof. Let a_i be an arbitrary good agent in RG_{min} . By Lemma 4.16, a_i executes $REL(a_i.gid)$ for $a_i.length$ rounds without interruption after round $r_{ini} + 1$.

For contradiction, assume that some agent a_j in RG never meets a reliable group with a group ID smaller than RG . In this case, a_j executes $REL(a_j.gid)$ for $a_j.length$ rounds without interruption after round $r_{ini} + 1$. Also, $a_i.id \geq a_i.gid$ and $a_j.id \geq a_j.gid$ hold. Thus, since a_i and a_j satisfy Line 8 of Algorithm 2, $a_i.length \geq 2 \cdot (t_{REL}(a_i.id) + 1) \geq 2 \cdot (t_{REL}(a_i.gid) + 1)$ and $a_j.length \geq 2 \cdot (t_{REL}(a_j.id) + 1) \geq 2 \cdot (t_{REL}(a_j.gid) + 1)$ hold. Therefore, since $a_i.gid < a_j.gid$ holds, by the behavior of *ByzantineGathering*, a_i and a_j execute $REL(a_i.gid)$ and $REL(a_j.gid)$ at the same time for at least $t_{REL}(a_i.gid)$ rounds after round $r_{ini} + 1$. By Lemma 3.1, a_i and a_j meet in $t_{REL}(a_i.gid)$ rounds after round $r_{ini} + 1$. Since $r^* > r_{ini} + t_{REL}(a_i.gid)$ holds and all good agents in RG_{min} stay at the same node, a_j meets RG_{min} . This is a contradiction. \square

Lemma 4.18. *All good agents in A_{fc} and all good agents in reliable groups in SRG_{ini} gather at the node with RG_{min} and transition into a terminal state by round r^* .*

Proof. By Lemma 4.16, every good agent a_i in RG_{min} executes $REL(a_i.gid)$ without interruption and transitions into a terminal state before round r^* . In addition, if a good agent meets RG_{min} , it follows RG_{min} after that by Lemma 4.14. Hence, it is sufficient to prove that all good agents in A_{fc} and all good agents in reliable groups other than RG_{min} meet RG_{min} before round r^* .

Consider a reliable group RG in $SRG_{ini} \setminus \{RG_{min}\}$. By Lemma 4.17, agents in RG meet a reliable group RG' with a smaller group ID before round r^* . If RG is RG_{min} , agents in RG meet

RG_{min} . Otherwise, good agents in RG follow RG' by Lemma 4.14. Similarly good agents in RG' meet another reliable group RG'' with a smaller group ID before r^* . Hence, good agents in RG also meet RG'' . By repeating this discussion, eventually good agents in RG meet RG_{min} before round r^* .

Lastly consider a good agent a_i in A_{fc} . As long as a_i does not meet a reliable group, it continues **MakeReliableGroup**. In this case, by Lemma 4.13, a_i meets RG_{min} before round r^* . Otherwise, if a_i meets a reliable group, it follows the reliable group by Lemma 4.14. In this case, similarly to the previous paragraph, a_i meets RG_{min} before round r^* . \square

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

Theorem 4.2. *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 6 solves the gathering problem in $O(f \cdot t_{REL}(a_{max}.id))$ rounds.*

Proof. By the behavior of **ByzantineGathering**, an agent executes **MakeReliableGroup** until the current node contains a reliable group. By Theorem 4.1, a reliable group is created in $O(f \cdot t_{REL}(a_{max}))$ rounds after starting **MakeReliableGroup**. Thus, round r_{ini} is in $O(f \cdot t_{REL}(a_{max}))$ rounds after starting **MakeReliableGroup**.

First, we consider agents in A_{fc} or some reliable group of $SRG_{ini} \setminus \{RG_{min}\}$. By Lemma 4.18, all good agents in A_{fc} or some reliable group of $SRG_{ini} \setminus \{RG_{min}\}$ gather at the node with RG_{min} and transition into a terminal state by round r^* .

Next, we consider agents in A_{ec} . Let a_ℓ be a good agent in A_{ec} . By Observation 4.7, round r^* is the last round of cycle c_ℓ^* . Therefore, from discussion in the previous paragraph, when a_ℓ starts the *MakeCandidate* stage, all good agents in A_{fc} or some reliable group of SRG_{ini} have gathered with RG_{min} and entered a terminal state. Also, since a_ℓ satisfies Line 8 of Algorithm 2, by the behavior of **MakeReliableGroup**, a_ℓ executes $REL(a_\ell.id)$ for at least $t_{REL}(a_\ell.id)$ rounds in the first cycle of the *MakeCandidate* stage. Therefore, by Lemma 3.1, a_ℓ visits all nodes in the first cycle of the *MakeCandidate* stage and thus meets RG_{min} during the cycle. By Lemma 4.14, a_ℓ follows RG_{min} at that time and hence transitions into a terminal state by the last round of the first cycle of the *MakeCandidate*.

Finally, we analyze the time complexity of **ByzantineGathering**. From the above discussion, the time required to achieve the gathering depends on whether A_{ec} contains at least one good agent or not. Thus, since a_{max} finishes the *CollectID* stage the latest, we consider the two cases, one where A_{ec} does not contain a_{max} and the other where A_{ec} contains a_{max} . In the former case, since $a_i.length < 32 \cdot (t_{REL}(a_{max}.id) + 1)$ holds for any good agent a_i by Lemma 4.3, $r^* \leq r_{ini} + 32 \cdot (t_{REL}(a_{max}.id) + 1) + 1$ holds. Thus, good agents achieve the gathering in $r_{ini} + a_i.length + 1 < O(f \cdot t_{REL}(a_{max})) + 32 \cdot (t_{REL}(a_{max}.id) + 1) + 1 = O(f \cdot t_{REL}(a_{max}))$ rounds. In the latter case, r^* is the last round of cycle c_{max}^* by Observation 4.7. Also, a_{max} finishes the *CollectID* stage in $O(t_{REL}(a_{max}.id))$ rounds after starting **MakeReliableGroup** by Lemma 4.4. Thus, since $a_{max}.length < 32 \cdot (t_{REL}(a_{max}.id) + 1)$ holds, good agents achieve the gathering in $O(t_{REL}(a_{max}.id)) + 32 \cdot (t_{REL}(a_{max}.id) + 1) = O(t_{REL}(a_{max}.id))$ rounds.

Hence, all good agents gather at the same node and transition into a terminal state in $O(f \cdot t_{REL}(a_{max}))$ rounds after starting **ByzantineGathering**. \square

5 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. In 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] 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. Springer, 2019.
- [2] Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Transactions on Algorithms*, 11(1):1–28, 2014.
- [3] Jion Hirose, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue. Gathering with a strong team in weakly byzantine environments. In *ICDCN ’21: International Conference on Distributed Computing and Networking*, pages 26–35. ACM, 2021.
- [4] Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55. Springer Science & Business Media, 2006.
- [5] Anders Dessmark, Pierre Fraigniaud, Dariusz R. Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.
- [6] Dariusz R. Kowalski and Adam Malinowski. How to meet in anonymous network. *Theoretical Computer Science*, 399(1-2):141–156, 2008.
- [7] Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms*, 10(3):12:1–12:15, 2014.
- [8] Avery Miller and Andrzej Pelc. Time versus cost tradeoffs for deterministic rendezvous in networks. *Distributed Computing*, 29(1):51–64, 2016.
- [9] Pierre Fraigniaud and Andrzej Pelc. Delays induce an exponential memory gap for rendezvous in trees. *ACM Transactions on Algorithms*, 9(2):17:1–17:24, 2013.
- [10] 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.
- [11] Pierre Fraigniaud and Andrzej Pelc. Deterministic rendezvous in trees with little memory. In *Distributed Computing, 22nd International Symposium, DISC 2008*, pages 242–256, 2008.
- [12] Yoann Dieudonné and Andrzej Pelc. Anonymous meeting in networks. *Algorithmica*, 74(2):908–946, 2016.
- [13] Evangelos Bampas, Jurek Czyzowicz, Leszek Gasieniec, David Ilcinkas, and Arnaud Labourel. Almost optimal asynchronous rendezvous in infinite multidimensional grids. In *Distributed Computing, 24th International Symposium, DISC 2010*, pages 297–311, 2010.
- [14] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, 2006.

- [15] Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM Journal on Computing*, 44(3):844–867, 2015.
- [16] Samuel Guilbault and Andrzej Pelc. Gathering asynchronous oblivious agents with local vision in regular bipartite graphs. *Theoretical Computer Science*, 509:86–96, 2013.
- [17] Sébastien Bouchard, Yoann Dieudonné, and Bertrand Ducourthial. Byzantine gathering in networks. *Distributed Computing*, 29(6):435–457, 2016.
- [18] Sébastien Bouchard, Yoann Dieudonné, and Anissa Lamani. Byzantine gathering in polynomial time. *Distributed Computing*, 2022.
- [19] Avery Miller and Ullash Saha. Fast byzantine gathering with visibility in graphs. In *Algorithms for Sensor Systems - 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2020*. Springer, 2020.
- [20] Masashi Tsuchida, Fukuhito Ooshita, and Michiko Inoue. Byzantine-tolerant gathering of mobile agents in arbitrary networks with authenticated whiteboards. *IEICE Transactions on Information and Systems*, 101-D(3):602–610, 2018.
- [21] Masashi Tsuchida, Fukuhito Ooshita, and Michiko Inoue. Byzantine-tolerant gathering of mobile agents in asynchronous arbitrary networks with authenticated whiteboards. *IEICE Transactions on Information and Systems*, 103-D(7):1672–1682, 2020.
- [22] Pankaj Khanchandani and Roger Wattenhofer. Byzantine agreement with unknown participants and failures. In *35th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2021*, pages 952–961. IEEE, 2021.
- [23] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.
- [24] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.