PAPER

Byzantine-Tolerant Gathering of Mobile Agents in Asynchronous Arbitrary Networks with Authenticated Whiteboards*

Masashi TSUCHIDA[†a)], *Nonmember*, **Fukuhito OOSHITA**[†b)], *Member*, *and* **Michiko INOUE**[†c)], *Fellow*

SUMMARY We propose two algorithms for the gathering of k mobile agents in asynchronous Byzantine environments. For both algorithms, we assume that graph topology is arbitrary, each node is equipped with an authenticated whiteboard, agents have unique IDs, and at most f weakly Byzantine agents exist. Here, a weakly Byzantine agent can make arbitrary behavior except falsifying its ID. Under these assumptions, the first algorithm achieves a gathering without termination detection in $O(m + fn)$ moves per agent (m is the number of edges and n is the number of nodes). The second algorithm achieves a gathering with termination detection in $O(m + fn)$ moves per agent by additionally assuming that agents on the same node are synchronized, $f < \lceil \frac{1}{3}k \rceil$ holds, and agents know k. To the best of our knowledge, this is the first work to address the gathering problem of mobile agents for arbitrary topology networks in asynchronous Byzantine environments.

key words: *mobile agent, gathering problem, Byzantine fault*

1. Introduction

Distributed systems, which are composed of multiple computers (nodes) that can communicate with each other, have become larger in scale recently. This makes it complicated to design distributed systems because developers must maintain a huge number of nodes and handle the massive amount of data communication among them. As a way to mitigate this difficulty, (mobile) agents have attracted a lot of attention [2]. Agents are software programs that can autonomously move from node to node and execute various tasks in distributed systems. In systems with agents, nodes do not need to communicate with other nodes because the agents themselves can collect and analyze data by moving around the network, which simplifies the design of distributed systems. In addition, agents can efficiently execute tasks by cooperating with other agents. Hence, many studies have investigated algorithms to realize cooperation among multiple agents.

The gathering problem is a fundamental task needed to realize cooperation among multiple agents. The goal of the gathering problem is to make all agents meet at a single node. When a gathering, all the agents can communicate with each other at the single node.

However, because agents themselves move on the distributed system and might be affected by several nodes, some agents might be cracked so that they do not follow the algorithm. We call such agents Byzantine agents. A Byzantine agent is assumed to execute arbitrary operations without following an algorithm. In this paper, we propose two algorithms that can make all correct agents meet at a single node regardless of the behavior of the Byzantine agents.

1.1 Related Work

The gathering problem has been widely studied in the literature [10], [11]. Table 1 summarizes some of the results. In this table, the time complexity in the asynchronous model is the largest number of moves required for a gathering by agents. The aim of these studies is to clarify the solvability of the gathering problem in various environments, and, if it is solvable, their aim is to clarify the optimal costs (e.g., time, number of moves, and memory space) required to achieve a gathering. To clarify solvability and optimal costs, many studies have been conducted under various environments with different assumptions on synchronization, anonymity, randomized behavior, topology, and the presence of node memory (whiteboards).

For synchronous networks, many deterministic algorithms to achieve a gathering have been proposed [3]–[5], [12]. If agents do not have unique IDs, they cannot gather in symmetric graphs such as rings because they cannot break symmetry. Therefore, some methods [3]–[5] assume unique IDs to achieve a gathering for any graph. Dessmark et al. [3] proposed an algorithm that realizes a gathering in $\tilde{O}(n^5 \sqrt{\tau l} + n^{10} l)$ unit times for any graph, where n is the number of nodes, l is the length of the smallest ID of agents, and τ is the maximum difference between the activation times of two agents. Kowalski et al. [4] and Ta-Shma et al. [5] improved the time complexity to $\tilde{O}(n^{15} + l^3)$ and $\tilde{O}(n^5 l)$, respectively, independently of τ. In contrast, some studies [13]–[15] studied the case in which agents have no unique IDs. In this case, gathering is not solvable for some graphs and initial positions of agents. Hence, these studies proposed algorithms only for solvable graphs and initial positions. They proposed memory-efficient gathering algorithms for trees [14], [15] and arbitrary graphs [13].

If a whiteboard exists on each node, the time required

Manuscript received November 22, 2019.
Manuscript revised March 6, 2020.
Manuscript publicized April 15, 2020.

[†]The authors are with Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan.

*This work was supported in part by JSPS KAKENHI Grant Number 18K11167 and JST SICORP Grant Number JPMJSC1806. The conference version of this work is published in The 6th Edition of The International Conference on Networked systems [1].

a) E-mail: tsuchida.masashi.td8@is.naist.jp
b) E-mail: f-oosita@is.naist.jp
c) E-mail: kounoe@is.naist.jp

DOI: 10.1587/transinf.2019EDP7311

Table 1 Gathering of agents with unique IDs in graphs (n is the number of nodes, l is the length of the smallest ID of the agents, τ is the maximum difference among the activation times of the agents, m is the number of edges, λ is the length of the longest ID of the agents, f_u is the upper bound of the number of Byzantine agents, D is the diameter of the graph, f is the number of Byzantine agents, k is the number of agents).

	Synchro-nicity	Graph	Knowledge	Byzantine	Whiteboard	Termination detection	Time complexity	Other assumptions
[3]†	Sync.	Arbitrary	–	Absence	None	Achieved	$\tilde{O}(n^5\sqrt{\tau l}+n^{10}l)$	–
[4]†	Sync.	Arbitrary	–	Absence	None	Achieved	$\tilde{O}(n^{15}+l^3)$	–
[5]†	Sync.	Arbitrary	–	Absence	None	Achieved	$\tilde{O}(n^5 l)$	–
[6]	Sync.	Arbitrary	n	Presence	None	Achieved	$\tilde{O}(n^9\lambda)$	–
[7]	Sync.	Arbitrary	f_u	Presence	Authenticated	Achieved	$O(f_u m)$	–
[8]†	Async.	Infinite lines	–	Absence	None	Achieved	$O((D+\lambda)^3)$	–
[8]†	Async.	Rings	–	Absence	None	Achieved	$O(n\lambda)$	–
[9]†	Async.	Arbitrary	–	Absence	None	Achieved	$poly(n,l)$	–
Trivial	Async.	Arbitrary	–	Absence	Existence	Achieved	$O(m)$	–
Proposed 1	Async.	Arbitrary	–	Presence	Authenticated	Unachieved	$O(m+fn)$	–
Proposed 2	Async.	Arbitrary	k	Presence	Authenticated	Achieved	$O(m+fn)$	$f<\lceil\frac{1}{3}k\rceil$ holds and agents on a single node are sycnronized

for a gathering can be significantly reduced. Whiteboards are areas prepared on each node at which agents can leave information. For example, when agents have unique IDs, they can write their IDs onto the whiteboards on their initial nodes. Agents can collect all the IDs by traversing the network [16], and they can achieve a gathering by moving to the initial node of the agent with the smallest ID. This trivial algorithm achieves a gathering in $O(m)$ unit times, where m is the number of edges. In contrast, when agents have no unique IDs, gathering is not trivial, even if they use whiteboards and randomization. Ooshita et al. [17] clarified the relationship between the solvability of randomized gathering and termination detection in rings with whiteboards.

Recently, some studies have considered gathering in the presence of Byzantine agents in synchronous networks [6], [7], [12]. Byzantine agents can perform an arbitrary behavior that does not follow the algorithm because of system errors, cracking, and similar reasons. In literature, weakly and strongly Byzantine agents are considered. Weakly Byzantine agents can behave arbitrarily except falsifying their IDs, and strongly Byzantine agents can behave arbitrarily (including falsifying their IDs). Dieudonné et al. [6] proposed an algorithm to achieve a gathering in $\tilde{O}(n^9\lambda)$ unit times in environments with weakly Byzantine agents, where λ is the length of the longest agent ID. For environments with strongly Byzantine agents, Bouchard et al. [12] minimized the number of correct agents required to achieve a gathering, but the time required for a gathering is exponential with respect to the number of nodes and labels of agents.

Tsuchida et al. [7] focused on environments with weakly Byzantine agents and reduced the time complexity to $O(f_u m)$ unit times using an authenticated whiteboards and signature function, where f_u is the upper bound of the number of Byzantine agents and m is the number of edges. They used authenticated whiteboards for each node, on which each agent is given a dedicated area to write information with its signature. This is because, if the whiteboard is not equipped with such authentication function, Byzantine agents can delete all the contents written by correct agents and so the whiteboard is useless. They also used a signature function, which allows an agent to create signed information that guarantees its ID and its current node. That is, any agent can identify the ID of the signed agent and whether it has been signed at the current node or not from the signature. These assumptions are practical because we can use digital signatures to guarantee publishers of the signatures. Note that, even if each node does not publish its ID to agents (i.e., nodes are anonymous from the viewpoint of agents), the node can also verify whether signed information has been signed there by using the some degital signature. We also assume the authenticated whiteboard and the signature function in this paper.

For asynchronous networks, many studies consider the gathering problem with additional assumptions. De Marco et al. [8] proposed an algorithm to achieve a gathering for two agents in asynchronous networks without considering Byzantine agents. They considered infinite lines and rings under the assumption that agents have unique IDs and can meet inside an edge. In infinite lines, their algorithm can achieve a gathering in $O((D+\lambda)^3)$ moves, where D is the distance between the two agents in the initial configuration. In rings, they proposed an algorithm to achieve a gathering in $O(n\lambda)$ moves. Dieudonné et al. [9] considered the gathering problem for arbitrary graphs under the same assumptions as [8]. They realized gathering in polynomial moves with respect to the number of nodes and the minimum length of the agent ID.

Das et al. [18] assumed Byzantine agent capabilities that differ from [6], [7], [12], and they realized gathering in asynchronous ring and mesh networks with Byzantine agents. In their model, correct agents can distinguish Byzan-

†This algorithm was originally proposed for the rendezvous problem (i.e., a gathering of two agents). However, it can be easily extended to the gathering problem by a technique in [4] without changing its time complexity.

tine agents. In addition, correct agents and Byzantine agents can neither meet on the same node nor pass each other on edges. Das et al. proposed an algorithm to achieve a gathering in $O(n)$ moves in this model.

Pelc [19] considered the gathering problem with crash faults under a weak synchronization model. A model was considered in which each agent moves at a constant but different speed. That is, although each agent has the same rate clock, the agent cannot know the number of clocks required for the movement of other agents. In this study, some agents may crash, that is, they may fail and stop at a node or an edge. Under this assumption, Pelc proposed algorithms to achieve a gathering in polynomial time for two cases in which agents do or do not retain their memory contents when they stop.

In other failure models, Chalopin et al. [20] considered a gathering problem with an asynchronous model in which no agents but the edges of the networks become faulty. They considered the case in which some of the edges in the network are dangerous or faulty such that any agent that travels along one of these edges disappears. They proposed an algorithm to achieve a gathering in $O(m(m + k))$ moves, where k is the number of agents, and proved that this cost is optimal.

1.2 Our Contributions

In this work, we propose two algorithms to achieve a gathering in asynchronous networks with weakly Byzantine agents. In the first algorithm, we adopt the same model as Tsuchida et al. [7] but without synchronicity. That is, Byzantine agents exist in an asynchronous network, and an authenticated whiteboard is attached to each node. Because most recent distributed systems are asynchronous, the proposed algorithm is suitable for more systems than previous algorithms for synchronous networks. To the best of our knowledge, there are no previous methods for asynchronous networks with Byzantine agents. If Byzantine agents do not exist, we can use a trivial algorithm with whiteboards in asynchronous networks. That is, agents can achieve a gathering in $O(m)$ moves using whiteboards in an asynchronous network. However, this trivial algorithm does not work when Byzantine agents exist. The first algorithm for asynchronous networks achieves a gathering without termination detection in at most $2m+3n-3+8f(n-1) = O(m+fn)$ moves per agent using authenticated whiteboards even if Byzantine agents exist, where f is the number of Byzantine agents. By definition, this algorithm also works in synchronous environments, and achieves a gathering earlier than the algorithm in [7]. However, the algorithm in [7] achieves a gathering with termination detection. This means the first algorithm reduces the time complexity by sacrificing termination detection.

The second algorithm realizes gathering with termination by making additional assumptions. By realizing termination detection, it is possible to notify the upper-layer application of the termination, which simplifies the design of distributed systems. To realize this, we assume that agents on the same node are synchronized. This assumption is practical and easy to implement because, in many mobile agent systems, each node can control the activation times of agents on the node. In addition, we assume $f < \lceil \frac{1}{3}k \rceil$ holds and agents know k. Under these assumptions, this algorithm achieves a gathering with termination detection in $O(m + fn)$ moves per agent. Compared with the first algorithm, this algorithm realizes termination detection without additional moves. When we apply the second algorithm to synchronous networks, the algorithm achieves a gathering with termination detection earlier than the algorithm in [7]. This means that, by making additional assumptions ($f < \lceil \frac{1}{3}k \rceil$ and each agent knows k), we can improve the time complexity for a gathering with termination detection in synchronous networks.

2. Preliminaries

2.1 Distributed System

A distributed system is modeled by a connected undirected graph $G = (V, E)$, where V is a set of nodes and E is a set of edges. The number of nodes is denoted by $n = |V|$. When $(u, v) \in E$ holds, u and v are adjacent. A set of adjacent nodes of node v is denoted by $N_v = \{u|(u, v) \in E\}$. The degree of node v is defined as $d(v) = |N_v|$. Each edge is labeled locally by function $\lambda_v : \{(v, u)|u \in N_v\} \to \{1, 2, \cdots, d(v)\}$ such that $\lambda_v(v, u) \neq \lambda_v(v, w)$ holds for $u \neq w$. We say $\lambda_v(v, u)$ is a port number (or port) of edge (v, u) on node v.

Nodes are anonymous, that is, they do not have IDs. Each node has an (authenticated) whiteboard where agents can leave information. Each agent is assigned a dedicated writable area in the whiteboard, and the agent can write information only in that area. In contrast, each agent can read the information from all areas (including areas of other agents) on the whiteboard.

2.2 Mobile Agent

Multiple agents exist in a distributed system. The number of agents is denoted by k, and a set of agents is denoted by $A = \{a_1, a_2, \cdots, a_k\}$. Each agent has a unique ID, and the ID of agent a_i is denoted by ID_i. In the first algorithm (Sect. 3), the agents know neither n nor k. In the second algorithm (Sect. 4), the agents know k but do not know n.

Each agent is modeled as a state machine (S, δ). The first element S is a set of agent states, where each agent state is determined by the values of the variables in its memory. The second element δ is the state transition function that determines the behavior of an agent. The input of δ is the states of all agents on the current node, the current node's degree, the content of the whiteboard in the current node, and the incoming port number. The output of δ is the next agent state, the next content of the whiteboard, whether the agent stays or leaves, and the outgoing port number if the agent leaves.

We assume the activations of agents are scheduled by an adversary. The adversary chooses one or more agents at one time, and each selected agent executes an atomic operation simultaneously. The two atomic operations of an agent selected by the adversary are detailed below. Note that two types of atomic operations exist depending on where an agent is selected by the adversary.

- If agent a_i is selected at node v, a_i executes the following operations as an atomic operation. First, a_i takes a snapshot, that is, a_i obtains the states of all agents at v and contents of the whiteboard at v. After that, a_i changes its own state and the content of the dedicated writable area in the whiteboard at v. Moreover, if a_i decides to move to an edge as a result of the local computation, it leaves v and moves to the edge.
- If agent a_j is selected at edge e, a_j arrives at the destination node as an atomic operation.

From the above definition, when an agent moves to an adjacent node, it moves to an edge connecting to the adjacent node in the first atomic operation and then moves from the edge to the adjacent node in the second atomic operation. Since the adversary decides when an agent executes the atomic operations, it can decide the time required for an agent to move from a node to its adjacent node. This implies that, while an agent moves from an node to its adjacent node, another agent can visit multiple nodes.

In the first algorithm (Sect. 3), we assume that agents operate in an asynchronous manner. To guarantee progress, we assume that for any agent a, the adversary chooses a infinitely many times. In the second algorithm (Sect. 4), we assume that agents on the same node are synchronized. That is, in addition to the above assumption, we assume that, if the adversary selects an agent a at a node v, it selects all agents at node v simultaneously.

In the initial configuration, each agent is located at an arbitrary different node. We assume that each agent performs an operation on its starting node earlier than other agents. That is, we assume that the adversary selects all agents at the same time at the beginning of an execution.

2.3 Signature

Each agent a_i can create signed information that guarantees its ID ID_i and its current node v by a signature function $Sign_{i,v}()$. That is, any agent can identify the ID of the signed agent and whether it has been signed at the current node or not from the signature. No nodes publish their IDs because nodes are anonymous. We assume a_i can use signature function $Sign_{i,v}()$ at v only. We call the output of signature function a *marker*, and $marker_{i,v}$ denotes a marker that a_i signed at node v. The marker's signature cannot be counterfeited, that is, an agent a_i can use a signature function $Sign_{i,v}()$ at v but cannot compute $Sign_{j,u}()$ for either $i \neq j$ or $v \neq u$ when a_i is located at v. Agents can copy a marker and paste it to any whiteboard, but cannot modify it without destroying its validity. In this paper, when an agent processes a marker, it first checks the validity of the signatures and ignores the marker if the marker includes the wrong signatures. We omit this behavior from descriptions, and assume all the signatures of every marker are valid.

When a_i creates a signed marker at node v, the marker contains ID_i and the information of node v. That is, when an agent finds a signed marker, it can identify 1) the ID of the agent that created it and 2) whether it was created at the current node or not. Therefore, it is guaranteed that signed marker $marker_{i,v}$ is created by a_i at v. When agent a_j is located at node v, a_j can recognize that $marker_{i,v}$ was created at v, and when a_j is located at node $u(\neq v)$, a_j can recognize that it was created at another node.

2.4 Byzantine Agents

Byzantine agents may exist in a distributed system. The number of Byzantine agents is denoted by f. Each Byzantine agent behaves arbitrarily without following the algorithm. However, a Byzantine agent cannot change its ID. In addition, even if agent a_i is Byzantine, a_i cannot compute $Sign_{j,u}()(i \neq j$ or $v \neq u)$ at node v, therefore a_i cannot create $marker_{j,u}(i \neq j$ or $v \neq u)$. Note that, each agent including Byzantine agent has a unique ID. In this paper, we assume that the agents do not know the number of Byzantine agents. In the second algorithm, we assume $f < \lceil \frac{1}{3}k \rceil$ holds.

2.5 Gathering Problem

The goal of the gathering problem is that all correct agents gather at one node. We consider gathering problem without termination detection and gathering problem with termination detection. We say an algorithm solves gathering problem without termination detection if all correct agents meet at a single node and continue to stay at that node after a certain point of time. Note that, when an algorithm solves gathering without termination, agents may not detect completion of gathering and may move again after waiting for an unexpected period. In the second property, we require agents to termination. Once an agent terminates, it can neither change its state nor move to another node. We say an algorithm solves gathering with termination if each correct agent has locally terminates and all correct agents exist on the same node after a certain point of time. To evaluate the performance of the algorithm, we consider the maximum number of moves required for any agent to achieve a gathering.

2.6 DFS Procedure

In this subsection, we review the procedure depth-first search (DFS) used in our algorithm. DFS is a well-known technique for exploring a graph [21]. In a DFS, an agent continues to explore an unexplored port as long as it visits a new node. If the agent visits an already visited node, it backtracks to the previous node and explores another unexplored port. If no unexplored port exists, the agent backtracks to the node from which it entered the current node for the first time. By repeating this behavior, each agent can visit all nodes in $2m$ moves, where m is the number of edges. Note that because each agent can realize DFS using only its dedicated area on the whiteboard, Byzantine agents cannot disturb the DFS of correct agents. We omit a detailed explanation for how to use these areas for DFS.

3. Gathering algorithm without termination detection

In this section, we propose an algorithm that solves gathering without termination detection. Here, we assume agents operate in an asynchronous manner. In addition, f Byzantine agents exist and all agents do not know n, k, or f.

3.1 Our Algorithm

3.1.1 Overview

First, we give an overview of our algorithm. This algorithm achieves a gathering of all correct agents in an asynchronous network even if Byzantine agents exist. The basic strategy of the algorithm is as follows.

When agent a_i starts on node v_{start}, a_i creates a marker $marker_{i,v_{start}}$ indicating that a_i is starting from v_{start}. We call this marker a starting marker. This marker contains information on the ID of the agent and the node where a_i created the marker. In this algorithm, all agents share their starting markers and meet at the node where the agent with the minimum ID created the starting marker.

To share the starting marker, a_i executes a DFS and leaves a copy of the marker at all nodes. When agent a_i sees the other agents' markers, a_i stores the markers to its own local variable. After agent a_i finishes the DFS and returns to v_{start}, a_i has all the markers of the correct agents and may have some markers of Byzantine agents. After that, a_i selects marker $marker_{min,v_{min}}$, which was made by the agent a_{min} with the minimum ID. If Byzantine agents do not exist, agent a_i can achieve a gathering by moving to node v_{min}, where marker $marker_{min,v_{min}}$ was created.

However, if Byzantine agents exist, they may interfere with a gathering in various ways. For example, Byzantine agents might not write their own starting markers, they might write and delete starting markers so that only some correct agents can see the markers, or they might create multiple starting markers. Because of these operations, agents may calculate different gathering nodes. To overcome this problem, in this algorithm, each agent shares the information about the starting marker created by the agent with the minimum ID with all agents to obtain a common marker. If all correct agents obtain a common marker of the minimum ID agent, they can calculate the same gathering node. However, while agents share the markers, Byzantine agents may make new markers to interfere with sharing. If agents share all the markers of the Byzantine agents, they may move infinite times to share the markers because Byzantine agents can create markers infinite times. To prevent such interference, each agent also shares a blacklist. The blacklist is a list of Byzantine agents' IDs. If the markers and the blacklists are shared, correct agents can identify the common marker that is created by the agent with the minimum ID from among the agents not in the blacklist.

We explain how agents identify Byzantine agents. When a_i calculates a gathering node and moves to that node for the first time, a_i refers to marker $marker_{min,v}$ created by the agent a_{min} with the minimum ID. If other agents copy marker $marker_{min,u}(v \neq u)$ and paste it to node v, a_i can determine that the two markers $marker_{min,v}$ and $marker_{min,u}$ were created by the same agent. Because the starting marker has been signed, an agent cannot camouflage the starting marker of other agents. In addition, correct agents create markers only once when they start the algorithm. Therefore, when there are two starting markers $marker_{min,v}$ and $marker_{min,u}(v \neq u)$ created by a single agent a_{min}, a_i can determine that a_{min} is a Byzantine agent. Note that a_i can confirm it only when a_i stays on node u or v. When a_i discovers that a_{min} is a Byzantine agent, a_i adds ID_{min} to the blacklist and shares ID_{min} with all agents as a member of the blacklist. To share ID_{min}, agent a_i shares two starting markers created by the Byzantine agent a_{min}. That is, a_i copies a_{min}'s two markers and pastes them to all the nodes so that all other agents can also determine that a_{min} is a Byzantine agent. After that, all correct agents ignore all markers of a_{min} and identify the marker created by the agent with the minimum ID from among the agents not in the blacklist. By these operations, all agents can select the node with the marker as the common gathering node.

Because we consider an asynchronous network, agent a_i does not know when other agents write a starting marker on the whiteboard. For this reason, after a_i moves to the gathering node, a_i continues to monitor the whiteboard and check for the presence of new markers. When a_i finds a new agent with the minimum ID or Byzantine agents, a_i repeats the above operation.

3.1.2 Details of the Algorithm

The pseudo-code of the algorithm is given in Algorithm 1. Variable $v.wb[ID_i]$ denotes a variable that is the writable area of agent a_i on the whiteboard on node v. Agent a_i manages local variables $a_i.All$, $a_i.state$, $a_i.min$, $a_i.t_{min}$, and $a_i.Byz$. Variable $a_i.All$ stores all markers observed by a_i. Variable $a_i.state$ stores `explore` or `gather`. When $a_i.state$ = `gather` holds, a_i arrives at the current gathering node and waits for other agents. When $a_i.state$ = `explore` holds, a_i is currently computing the gathering node or moving to the node. Variable $a_i.t_{min}$ stores the marker created by the agent with the minimum ID, excluding the Byzantine agents' IDs that a_i has observed. Variable $a_i.min$ stores the ID of the agent that created $a_i.t_{min}$. Variable $a_i.Byz$ is a blacklist, that is, it stores all Byzantine agent IDs that a_i has confirmed. The initial values are $a_i.All = \emptyset$, $a_i.state$ = `explore`, $a_i.min = \infty$, $a_i.t_{min} = null$, and $a_i.Byz = \emptyset$. In addition, function $writer(marker_{i,v})$ returns i, that is, the ID of the agent that created $marker_{i,v}$. Func-

Algorithm 1 Gathering procedure without termination detection

```
1: // Algorithm of agent $a_i$. Node $v$ indicates the node at which $a_i$ is located.
2: $a_i.marker \leftarrow Sign_{i,v}()$, $a_i.All \leftarrow \emptyset$, $a_i.state \leftarrow$ explore
   // $a_i.marker = marker_{i,v}$ in an explanation of the algorithm
3: $a_i.All \leftarrow first\_move(a_i.marker, a_i.All)$
4: $a_i.t_{min} \leftarrow null$, $a_i.min \leftarrow \infty$, $a_i.Byz \leftarrow \emptyset$, $a_i.T_{Byz} \leftarrow \emptyset$
5: while $True$ do
6:     $a_i.All \leftarrow a_i.All \cup \bigcup_{id} v.wb[id]$
7:     $min\_tmp \leftarrow min\{writer(t) : t \in a_i.All \wedge writer(t) \notin a_i.Byz\}$
8:     if $a_i.min > min\_tmp$ then
9:         $a_i.state \leftarrow$ explore
10:        $a_i.t_{min} \leftarrow t$ s.t. $t \in a_i.All \wedge writer(t) == min\_tmp$
11:        $a_i.min \leftarrow min\_tmp$
12:        while $a_i$ traverses the network do
13:            $v.wb[ID_i] \leftarrow v.wb[ID_i] \cup \{a_i.t_{min}\}$
14:            Move to the next node
15:        end while
16:        Move to the node where $a_i.t_{min}$ was created
17:    else
18:        if $\exists x : x \in a_i.All \wedge writer(x) == a_i.min \wedge node\_check(x) == false$ then
19:            $a_i.state \leftarrow$ explore
20:            $a_i.T_{Byz} \leftarrow \{x, a_i.t_{min}\}$
21:            while $a_i$ traverses the network do
22:                $v.wb[ID_i] \leftarrow v.wb[ID_i] \cup a_i.T_{Byz}$
23:                Move to the next node
24:            end while
25:            $a_i.Byz \leftarrow a_i.Byz \cup \{a_i.min\}$
26:            $a_i.min \leftarrow \infty$
27:        else
28:            $a_i.state \leftarrow$ gather
29:            Stay at node $v$
30:        end if
31:    end if
32: end while
```

Algorithm 2 Function: $first\_move(marker, marker\_set)$.

```
1: $a_i.dfs\_marker \leftarrow marker$, $a_i.marker\_set \leftarrow marker\_set$
2: while $a_i$ is executing DFS do
3:     Activate inactive agents if such agents exist at $v$
4:     $v.wb[ID_i] \leftarrow \{a_i.dfs\_marker\}$
5:     $a_i.marker\_set \leftarrow a_i.marker\_set \cup \bigcup_{id} v.wb[id]$
6:     Store the network topology
7:     Move to the next node by DFS
8:     return ($a_i.marker\_set$)
9: end while
```

tion $node\_check(marker_{i,v})$ returns true if $marker_{i,v}$ was created on the current node, and otherwise returns false. Function $first\_move(marker_{i,v}, marker\_set)$ executes a DFS and copies the marker and pastes it to all nodes. We show the pseudo-code of function $first\_move(marker_{i,v}, marker\_set)$ in Algorithm 2. This function eventually returns the set of markers that the agent observed during DFS. We explain the behavior of this function later.

Recall that, in an atomic operation, an agent obtains the snapshot, updates its state and the whiteboard, and then, possibly leaves the node. In Algorithm 1, each agent executes the operations as an atomic operation until it leaves (lines 14, 16, and 23) or decides to stay (line 29). Similarly, each agent executes the operations as an atomic operation until line 7 of Algorithm 2. When an agent reads from the whiteboard, it uses the snapshot taken at the beginning of an atomic operation.

When a_i starts the algorithm, it creates starting marker $marker_{i,v} = Sign_{i,v}()$ and enters the explore state (line 2). After a_i creates the starting marker, to inform other agents about the marker, a_i executes function $first\_move(marker_{i,v}, marker\_set)$. First, a_i stores function arguments in local variables ($a_i.dfs\_marker$ and $a_i.marker\_set$). Then, when a_i visits node v, if an inactive agent a_j exists, a_i activates a_j. In this case, agent a_j starts the algorithm before a_i executes the algorithm at v. Thus, a_i can read information written by a_j at this time. On every node, a_i adds the other agent's markers stored at that node to $a_i.marker\_set$ (line 5 of Algorithm 2). To obtain the network topology, a_i memorizes the connection relation between all nodes and all edges during the DFS. Consequently, a_i can traverse the network in at most $2n$ moves after it finishes DFS. Finally, this function returns $a_i.marker\_set$ and ends the operation.

After a_i finishes the DFS, a_i checks the markers collected in $a_i.All$ and calculates the gathering node (lines 6 to 33). First, a_i stores the markers of the current node in $a_i.All$ to check for new markers. After that, a_i selects the ID ID_{min} such that $ID_{min} = min\{writer(t) : t \in a_i.All \wedge writer(t) \notin a_i.Byz\}$ holds (line 7). If $a_i.min > ID_{min}$, a_i executes an update operation of a gathering node (lines 8 to 16). Otherwise, a_i executes an operation to detect Byzantine agents (lines 17 to 31).

In the update operation of a gathering node, a_i calculates a new gathering node. In this step, a_i stores marker t satisfying $writer(t) == min\{writer(t) : t \in a_i.All \wedge writer(t) \notin a_i.Byz\}$ to $a_i.t_{min}$ and stores $writer(a_i.t_{min})$ to $a_i.min$. After that, a_i copies $a_i.t_{min}$ and pastes it to all nodes to inform other agents of the marker of the minimum ID agent (lines 12 to 15). Note that, because a_i knows the graph topology, it can visit all nodes in at most $2n$ moves. In addition, because a_i visits all nodes, a_i knows at which node $a_i.t_{min}$ was created. That is, a_i executes $node\_check(a_i.t_{min})$ at each visited node and memorizes the node where the function returns true (recall that $node\_check(a_i.t_{min})$ returns true only if $a_i.t_{min}$ is created on the current node). Therefore, after a_i copies $a_i.t_{min}$ and a_i pastes it to all nodes, a_i can move to the node where $a_i.t_{min}$ was created. If there are two or more markers created by an agent with the minimum ID, a_i refers to one of the markers and calculates a gathering node. Then, in the detection operation of the next while-loop, a_i identifies an agent with the minimum ID as a Byzantine agent.

In the Byzantine agent detection operation, a_i determines whether the minimum ID agent is a Byzantine agent. If there is a marker x that satisfies $x \in a_i.All \wedge writer(x) == a_i.min \wedge node\_check(x) == false$, a_i determines that $writer(x)$ is a Byzantine agent. This is because correct agents create markers only once. Hence, only Byzantine agents can create markers on two nodes. In this case, a_i informs other agents of the ID of the Byzantine agent and executes the update operation in the next while-loop. To realize this, a_i copies the starting markers of the Byzantine agent and pastes them to all nodes, and then a_i initializes $a_i.min = \infty$.

Finally, if a_i executes local computation and determines that the current node is a gathering node, a_i changes $a_i.state$ to `gather`. After that, if a_i decides to change the gathering node, a_i changes $a_i.state$ to `explore` again (lines 9 and 19).

By repeating the above operation, all the correct agents eventually refer to the starting markers created by the same minimum ID agent and gather at the same node.

3.2 Correctness of the First Algorithm

Lemma 1: Correct agent a_i never adds correct agent a_j's ID ID_j to $a_i.Byz$.

Proof : Correct agent a_j creates a starting marker $marker_{j,v} = Sign_{j,v}()$ only once when it starts the algorithm at node v. In addition, Byzantine agents cannot create or modify the signed marker of a_j. Therefore, there is no marker $marker_{j,u} = Sign_{j,u}()$ $(v \neq u)$.

Recall that a_i adds ID_b to $a_i.Byz$ only when agent a_i confirms that agent a_b created starting markers $marker_{b,v}$ and $marker_{b,u}$ $(v \neq u)$. Thus, a_i never adds correct agent a_j's ID ID_b to $a_i.Byz$. □

Lemma 2: For any correct agent a_i, after a_i finishes function $first\_move(marker_{i,v}, marker\_set)$, there exists at least one marker $marker_{x,v}$ that satisfies $marker_{x,v} \in a_i.All \wedge writer(marker_{x,v}) \notin a_i.Byz$.

Proof : Correct agent a_i stores all of the starting markers observed during the execution of function $first\_move(marker_{i,v}, marker\_set)$ to $a_i.marker\_set$. These markers also include a_i's marker $marker_{i,v_{start}}$. From Lemma 1, correct agent a_i never adds correct agents' IDs to $a_i.Byz$. In addition, because a_i itself is also a correct agent, ID ID_i is never stored in $a_i.Byz$. Therefore, after a_i finishes $first\_move(marker_{i,v}, marker\_set)$, $marker_{i,v_{start}} \in a_i.All \wedge writer(marker_{i,v_{start}}) \notin a_i.Byz$ holds, which implies the lemma. □

From Lemma 2, correct agent a_i selects a marker created by an agent with the minimum ID from among markers that satisfy $marker_{i,v_{start}} \in a_i.All \wedge writer(marker_{i,v_{start}}) \notin a_i.Byz$. After that, a_i determines the node where the selected marker was created as a gathering node.

Lemma 3: After correct agent a_i has calculated a gathering node for the first time, a_i updates $a_i.min$ at most $2f$ times.

Proof : While correct agent a_i executes $first\_move(marker_{i,v}, marker\_set)$ at the beginning of the algorithm, a_i can observe all the markers of correct agents. Therefore, when a_i calculates a gathering node for the first time, $a_i.min$ becomes the minimum ID of the correct agents or a smaller ID of some Byzantine agent. After that, a_i updates $a_i.min$ when a_i observes a marker t satisfying $a_i.min > writer(t)$ or when a_i determines that $a_i.min$ is an ID of Byzantine agent. Because a_i observes all the markers of correct agents during $first\_move(marker_{i,v}, marker\_set)$, a_i can observe a marker t satisfying $a_i.min > writer(t)$ only when $writer(t)$ is an ID of a Byzantine agent. In addition, after a_i determines that $a_i.min$ is an ID of a Byzantine agent, a_i never updates $a_i.min$ using that ID. Thus, a_i updates $a_i.min$ at most twice per Byzantine agent, which implies a_i updates $a_i.min$ at most $2f$ times. □

From Lemma 3, a_i updates $a_i.min$ at most $2f$ times. That is, there is the last update at which a_i calculates the value of $a_i.min$.

Lemma 4: For any correct agents a_i and a_j, after the last updates of $a_i.min$ and $a_j.min$, $a_i.min$ and $a_j.min$ are equal.

Proof : We prove this lemma by contradiction. Assume that, after the last updates of $a_i.min$ and $a_j.min$, $a_i.min \neq a_j.min$ holds for correct agents a_i and a_j. Without loss of generality, we assume that $a_i.min = x < a_j.min = y$ holds.

To satisfy $a_i.min < a_j.min$, a_i and a_j should observe different markers $marker_{x,v}$ and $marker_{y,u}$. When a_i regards $marker_{x,v}$ as the marker created by the minimum ID agent, the agent copies the marker $marker_{x,v}$ and pastes it to all the nodes. After that, a_j observes the copied marker $marker_{x,v}$. Because we assume that $a_i.min = x < a_j.min = y$ holds, a_j should update $a_j.min$ to x. Therefore, a_j should update $a_j.min$ after the last update. This is a contradiction. Thus, the lemma holds. □

Lemma 5: All correct agents gather at one node with the `gather` state within a finite time.

Proof : Correct agent a_i finds node v such that the marker was created by agent with ID $a_i.min$, and then sets v as a gathering node. From Lemma 2, there is a gathering node. In addition, from Lemma 3, after correct agent a_i calculates the gathering node for the first time, a_i updates $a_i.min$ at most $2f$ times. In addition, from Lemma 4, for any correct agents a_i and a_j, after the last updates of $a_i.min$ and $a_j.min$, $a_i.min$ and $a_j.min$ are equal. Therefore, all correct agents refer to the same marker $marker_{min,v}$, and calculate the same node v to be the gathering node. Moreover, because the time required for agents to move between nodes is also finite, all correct agents can arrive at gathering node v in a finite time. Because a_i updates the value of $a_i.min$ at most 2_f times, there are configurations in which $a_i.min$ is not updated within a finite time. In addition, if a_i does not change the value of $a_i.min$, a_i becomes `gather` state. This implies all agents gather at v with `gather` state within a finite time. □

Theorem 1: Algorithm 1 solves gathering with termination. In the algorithm, each agent moves at most $2m + 3n - 3 + 8f(n - 1)$ times.

Proof : From Lemma 5, all correct agents gather at one node with the `gather` state within finite time. Let us consider the number of moves required for the gathering. Correct agent a_i first visits all nodes by DFS, which requires $2m$

moves. After that, when a_i calculates the gathering node for the first time, it copies and pastes the marker created by the minimum ID agent to all nodes and moves to the gathering node. In this movement, a_i can copy the starting marker and paste it to all nodes with at most $2n - 2$ moves because the agent knows the graph topology while executing DFS. After a_i copies and pastes the marker, a_i moves to a gathering node with at most $n - 1$ moves.

Every time a_i updates $a_i.min$, a_i copies the starting marker and pastes it to all nodes with at most $2n - 2$ moves and moves to a new gathering node with at most $n-1$ moves. From Lemma 3, a_i updates $a_i.min$ at most $2f$ times. In addition, when the minimum ID agent is determined to be a Byzantine agent, a_i informs the other agents about the ID of that Byzantine agent by copying and pasting that starting marker to all nodes using at most $2n - 2$ moves. Therefore, a_i moves at most $2m + 2n - 2 + n - 1 + 2f((2n - 2) + (n - 1)) + f \times (2n - 2) = 2m + 3n - 3 + 8f(n - 1)$ times. □

It seems that we cannot bound the space complexity because Byzantine agent a_b can put an arbitrary number of markers in $v.wb[ID_b]$ on some node v. However, we can easily modify the algorithm to bound the space complexity. As the first modification, for every agent a_i and every node v, we forbid a_i to write more than two markers with the same creator into $v.wb[ID_i]$. We can realize this by preparing two variables for each creator instead of $v.wb[ID_i]$. Since correct agents write at most two markers with the same creator, such variables are sufficient to execute the algorithm. Since the number of creators is at most k, each node requires an $O(k \cdot |marker| + |auth|)$-bit memory for a writable area of one agent (note that DFS requires a smaller memory), where $|marker|$ is the maximum number of bits for signed markers and $|auth|$ is the maximum number of bits required for authentication of agents. Hence, each node requires an $O(k^2 \cdot |marker| + k \cdot |auth|)$-bit memory in total. As the second modification, we forbid correct agent a_i to store more than two markers with the same creator into $a_i.marker\_set$ and $a_i.All$. Recall that, if agent a_i knows that a_j creates two markers, a_i regards a_j as a Byzantine agent. Consequently a_i does not have to bring more than two markers with the same creator. Hence, agent a_i requires an $O(k \cdot |marker|)$-bit memory for $a_i.marker\_set$ and $a_i.All$. In addition, a_i requires an $O(m + n)$-bit memory to store the graph topology. Since other variables require a smaller memory, a_i requires an $O(k \cdot |marker| + m + n)$-bit memory.

4. Gathering Algorithm with Termination Detection

In this section, we propose an algorithm that solves gathering with termination detection. To realize the algorithm, we add the assumptions that agents on a single node are synchronized, $f < \lceil \frac{1}{3}k \rceil$ holds, and agents know k. In addition, we define $f_u = \lfloor \frac{k-1}{3} \rfloor$. Note that because f_u is the maximum integer less than $\lceil \frac{1}{3}k \rceil$, f_u is an upper bound of f.

4.1 Our Algorithm

4.1.1 Overview

First, we present an overview of our algorithm. This algorithm achieves a gathering with termination detection in asynchronous networks even if Byzantine agents exist. Agents execute the same operations as Algorithm 1 until $k - f_u$ agents gather at the same node and enter the `gather` state. After at least $k - f_u$ agents in the `gather` state gather at one node v, all correct agents at v terminate. Note that, because the $k - f_u$ agents execute the algorithm synchronously at v and at most f_u Byzantine agents exist, at least $k - 2f_u \geq f_u + 1$ correct agents terminate at v from $f_u = \lfloor \frac{k-1}{3} \rfloor$. As we show in Lemma 8, correct agents that have not terminated yet eventually visit v. When correct agents visit v, they can see that at least $f_u + 1$ agents have terminated, and then they also terminate at v. In addition, we show in Lemma 7 that there is only one node v at which at least $f_u + 1$ agents have terminated. Thus, all correct agents have gathered at one node and terminated.

4.1.2 Details of the Algorithm

The pseudo-code of the algorithm is given in Algorithm 3. It is basically the same as Algorithm 1, but has additional lines 6 to 8 and 17 to 19. Recall that, in an atomic operation, an agent obtains the snapshot, updates its state and the whiteboard, and then, possibly leaves the node. In the pseudo-code, each agent executes the operations as an atomic operation until it leaves (lines 22, 24, and 31), decides to stay (line 37), or terminate (lines 8 and 19). Similarly, each agent executes the operations as an atomic operation until line 7 of Algorithm 2. When an agent reads from the whiteboard, it uses the snapshot taken at the beginning of an atomic operation.

In Algorithm 3, agents execute the same operations as Algorithm 1 until at least $k - f_u$ agents in the `gather` state gather at its current node v. After at least $k - f_u$ agents in the `gather` state gather at node v, correct agents terminate at node v (lines 6 to 8). If agent a_i sees at least $k - f_u$ agents in the `gather` state or at least $f_u + 1$ agents in the `terminate` state at node v, a_i terminates at v (lines 17 to 19). Agent a_i executes the above operation while a_i visits all nodes to paste marker $a_i.t_{min}$ to update the gathering node. Note that a_i does not execute the operation while a_i visits nodes to paste $a_i.T_{Biz}$ to update the blacklist of Byzantine agents (lines 26 to 34). This is because a_i executes an update operation of the gathering node after an update operation of the blacklist.

By repeating the above operation, all the correct agents eventually refer to the starting marker created by the minimum ID agent and gather at the same node while terminating.

Algorithm 3 Gathering procedure with termination detection

```
1: //Algorithm of agent $a_i$. Node $v$ indicates the node at which $a_i$ is located.
2: $a_i.marker \leftarrow Sign_{i,v}()$, $a_i.All \leftarrow \emptyset$, $a_i.state \leftarrow$ explore
   // $a_i.marker = marker_{i,v}$ in an explanation of the algorithm
3: $a_i.All \leftarrow first\_move(a_i.marker, a_i.All)$
4: $a_i.t_{min} \leftarrow null$, $A_i.min \leftarrow \infty$, $a_i.Byz \leftarrow \emptyset$, $a_i.T_{Byz} \leftarrow \emptyset$
5: while true do
6:   if There exist at least $k - f_u$ agents of gather state at node $v$ then
7:     $a_i.state \leftarrow$ terminate
8:     terminate
9:   else
10:    $a_i.All \leftarrow a_i.All \cup \bigcup_{id} v.wb[id]$
11:    $min\_tmp \leftarrow min\{writer(t) : t \in a_i.All \wedge writer(t) \notin a_i.Byz\}$
12:    if $a_i.min > min\_tmp$ then
13:      $a_i.state \leftarrow$ explore
14:      $a_i.t_{min} \leftarrow t$ s.t. $t \in a_i.All \wedge writer(t) == min\_tmp$
15:      $a_i.min \leftarrow min\_tmp$
16:      while $a_i$ traverses the network do
17:        if There are at least $k - f_u$ agents of gather state or
             at least $f_u + 1$ agents of terminate state at node $v$
           then
18:          $a_i.state \leftarrow$ terminate
19:          terminate
20:        end if
21:        $v.wb[ID_i] \leftarrow v.wb[ID_i] \cup \{a_i.t_{min}\}$
22:        Move to the next node
23:      end while
24:      Move to the node where $a_i.t_{min}$ was created
25:    else
26:      if $\exists x : x \in a_i.All \wedge writer(x) == a_i.min \wedge node\_check(x) == false$ then
27:        $a_i.state \leftarrow$ explore
28:        $a_i.T_{Byz} \leftarrow \{x, a_i.t_{min}\}$
29:        while $a_i$ traverses the network do
30:          $v.wb[ID_i] \leftarrow v.wb[ID_i] \cup a_i.T_{Byz}$
31:          Move to the next node
32:        end while
33:        $a_i.Byz \leftarrow a_i.Byz \cup a_i.min$
34:        $a_i.min \leftarrow \infty$
35:      else
36:        $a_i.state \leftarrow$ gather
37:        stay at node $v$
38:      end if
39:    end if
40:  end if
41: end while
```

4.2 Correctness of the Second Algorithm

Lemma 6: If a correct agent of `terminate` state exists at a node v, at least $f_u + 1$ correct agents of `terminate` state exist at v.

Proof : Assume that at least one correct agent of `terminate` state exists at v. Note that each agent enters `terminate` state when it terminates. Let a be the correct agent that terminates earliest at v. To terminate the algorithm, a must evaluate the predicate of line 6 or 17 as true. Because at most f_u Byzantine agents exist, a does not see $f_u + 1$ agents in the `terminate` state at v and thus, it never evaluates the latter half of the predicate of line 17 as true. Consequently, when a terminates, it sees at least $k - f_u$ agents in the `gather` state at v. From $f_u = \lfloor \frac{k-1}{3} \rfloor$, $k - 2f_u \geq f_u + 1$ agents among the $k - f_u$ agents are correct. Because all agents on the same node are synchronized, at least $f_u + 1$ correct agents in the `gather` state execute lines 6 to 8 (or 17 to 19) at the same time. They also see at least $k - f_u$ agents in the `gather` state, and thus terminate at v. Therefore, the lemma holds. □

Lemma 7: At least one correct agent eventually terminates.

Proof : We prove this lemma by contradiction. Assume that no correct agent terminates. In Algorithm 3, a correct agent terminates if and only if it evaluates the predicate of line 6 or 17 as true. This implies no correct agent evaluates the predicate as true.

Recall that only lines 6 to 8 and 17 to 19 in Algorithm 3 have been added to Algorithm 1. Thus, if predicates of lines 6 and 17 of Algorithm 3 are always false, agents perform the same behaviors as those of Algorithm 1. Consequently, from Theorem 1, all correct agents gather at the same node within a finite time. Therefore, at least $k - f_u$ correct agents enter `gather` state at the node and they evaluate the predicate of line 6 in Algorithm 3 as true. This is a contradiction. □

We define a_f as the correct agent that terminates earliest of all agents. Let t_f be the time at which a_f terminates and v_f be the node where a_f terminates.

Lemma 8: Each agent moves at most $O(m + fn)$ times before time t_f.

Proof : Before time t_f, correct agents always evaluate predicates of lines 6 and 17 as false (otherwise they terminate). Consequently, all correct agents perform the same behaviors as Algorithm 1. From Theorem 1, each agent moves at most $O(m + fn)$. □

Lemma 9: No correct agent terminates at node v' $(v' \neq v_f)$.

Proof : We prove this lemma by contradiction. Assume that some correct agent terminates at v' $(v' \neq v_f)$. Let a_s be the earliest correct agent that terminates at v'.

From Lemma 6, at least $f_u + 1$ correct agents terminate at node v_f. When a_s terminates at v', it sees at least $k - f_u$ agents in the `gather` state at v'. However, because at least $f_u + 1$ agents have already terminated at v_f, at least $k - f_u$ agents cannot gather at v'. This is a contradiction, and thus the lemma holds. □

Corollary 1: After time t_f, $f_u + 1$ agents in the `terminate` state exist at v_f. For any node v' $(v' \neq v_f)$, the number of agents in the `terminate` state at v' is at most f_u.

Proof : Because $f_u + 1$ correct agents terminate at the same time as a_f from Lemma 6, the first part clearly holds. From Lemma 9, no correct agent terminates at v' $(v' \neq v_f)$. Thus, the second part holds. □

Lemma 10: Each correct agent not in v_f at time t_f terminates at v_f after moving $O(m)$ times.

Proof : Let a be a correct agent not in v_f at time t_f. Let id_{v_f} be the ID of the agent that starts the algorithm from node v_f.

We consider three cases depending on the status of agent a at time t_f: 1) a considers v_f as a gathering node, 2) a considers a node other than v_f as a gathering node, and 3) a has not finished the $first\_move(marker_{i,v}, marker\_set)$. In the first case, $a.min = id_{v_f}$ holds. Agent a visits node v_f within a finite time because, from Corollary 1, v_f is the unique node at which $f_u + 1$ agents terminate.

In the second case, we assume that a considers v' ($v' \neq v$) to be the gathering node. In addition, we define id'_{v_f} to be the ID of the agent that starts the algorithm from node v'. Here, there are two subcases: $id_{v_f} > id_{v'}$ or $id_{v_f} < id_{v'}$. When $id_{v_f} > id_{v'}$ holds, we consider two cases.

- Agent a_f does not consider $id_{v'}$ to be an ID of a Byzantine agent (i.e., $id_{v'} \notin a_f.Byz$) at time t_f. In this case, a marker of $id_{v'}$ or smaller ID must not exist at v_f at time t_f because, if such a marker existed at v_f, a_f would move to the corresponding node as the new gathering node. Because a must paste a marker of $id_{v'}$ or a smaller ID to all nodes before entering the `gather` state, a visits v_f after t_f.
- Agent a_f considers $id_{v'}$ to be an ID of a Byzantine agent (i.e., $id_{v'} \in a_f.Byz$) at time t_f. In this case, a_f has pasted two markers of $id_{v'}$ to all nodes before it terminated. Consequently, v' contains two markers of $id_{v'}$. At time t_f, agent a executes an update operation of gathering node v' or a Byzantine agent detection operation. In the former case, after a completes pasting a marker of $id_{v'}$ to all nodes, it moves to v'. Then, at v', a understands that $id_{v'}$ is an ID of a Byzantine agent. Consequently, a pastes two markers of $id_{v'}$ to all nodes and then executes the update operation for a new gathering node. During the update operation, a visits v_f. In the latter case, a already knows $id_{v'}$ is an ID of a Byzantine agent, and explores the network to paste two markers of $id_{v'}$. Hence, after a arrives at v', it executes the update operation for a new gathering node. During the update operation, a visits v_f.

When $id_{v_f} < id_{v'}$ holds, the marker of id_{v_f} must exist in v' at time t_f because a_f pasted a marker of id_{v_f} to all nodes before terminating at v_f. At time t_f, agent a executes an update operation of the gathering node v' or a Byzantine agent detection operation. In the former case, after a has completed pasting a marker of $id_{v'}$ to all nodes, a moves to v'. Then, at v', a finds a marker of id_{v_f} or a smaller ID and executes an update operation for the gathering node. During the update operation, a visits a_f. In the latter case, a knows $id_{v'}$ is an ID of a Byzantine agent, and explores the network to paste two markers of $id_{v'}$. Hence, after a arrives at v', a executes an update operation for a new gathering node. During the update operation, a visits v_f.

In the third case, a eventually finishes DFS and goes back to the starting node of a in function $first\_move(marker_{i,v}, marker\_set)$. Then, a executes an update operation for the gathering node. During the update operation, a visits v_f.

For all cases, when a visits v_f after time t_f, correct agent a_f has already terminated. From Lemma 6, a sees at least $f + 1$ agents in the `terminate` state and terminates there. In addition, from the above cases, a terminates at v_f before it explores the network twice. Hence, the lemma holds. □

Theorem 2: Algorithm 3 achieves a gathering with termination within a finite time. In the algorithm, each agent moves at most $O(m + fn)$ times.

Proof : From Lemma 8, each agent moves $O(m+fn)$ times before time t_f. After time t_f, from Lemma 10, each correct agent not in v_f at time t_f terminates at v_f after moving $O(m)$ times. Therefore, all correct agents gather at v_f and terminate, and each agent moves $O(m + fn)$ times. □

Since the behavior of agents in Algorithm 3 is the same as in Algorithm 1 except for termination, the memory requirement of Algorithm 3 can be discussed similarly to Algorithm 1.

4.3 Remarks

In this section, we have assumed that terminated agents do not move to other nodes. However, the terminated agents may execute other algorithms and hence want to move to other nodes. In this subsection, we describe a method to deal with this situation.

Algorithm 3 requires terminated agents to stay at the gathering node because other agents determine the gathering node by the number of terminated agents at a node. To treat this, we propose a method that uses a signed marker instead of a terminated agent. If an agent terminates at the gathering node, the agent makes a signed marker and puts it at the node. After that, if agents executing Algorithm 3 see this marker at a node, they recognize that an agent that created the marker stays at the node with `terminate` state.

Of course Byzantine agents can also create their signed markers, and they can put and delete their signed markers at any node. In addition, a Byzantine agent can put its marker on multiple nodes. This is equivalent to the situation that the Byzantine agent exists on multiple nodes at the same time, however this does not disrupt the algorithm. In asynchronous environments, agents cannot recognize whether agents on other nodes execute operations at the same time or not. Hence we can assume that agents on different nodes execute operations at different times. In this case, immediately before agents on a node execute operations, a Byzantine agent can visit the node. Since Algorithm 3 works correctly for such an execution, it also works correctly even if a Byzantine agent puts its marker on multiple nodes.

5. Conclusions

In this work, we proposed two gathering algorithms for mobile agents in asynchronous Byzantine environments with authenticated whiteboards. Each algorithm achieves a gathering in $O(m + fn)$ moves per agent. Algorithm 1 achieves a gathering without termination detection. Algorithm 3 realizes termination detection by making additional assumptions. The additional assumptions are that agents on a single node are synchronized and each agent knows f and k, where f is the number of Byzantine agents and k is the number of total agents.

Open problems are as follows: 1) whether it is possible to terminate for the same assumptions of Algorithm 1 and 2) whether it is possible to relax the additional assumptions used in Algorithm 3.

References

[1] M. Tsuchida, F. Ooshita, and M. Inoue, "Gathering of mobile agents in asynchronous Byzantine environments with authenticated whiteboards," The international conference on networked systems (NETYS2018), pp.85–99, Springer, 2018.
[2] J. Cao and S.K. Das, "Mobile Agents in Networking and Distributed Computing," Wiley, 2012.
[3] A. Dessmark, P. Fraigniaud, D.R. Kowalski, and A. Pelc, "Deterministic rendezvous in graphs," Algorithmica, vol.46, no.1, pp.69–96, 2006.
[4] D.R. Kowalski and A. Malinowski, "How to meet in anonymous network," Theoritical Computer Science, vol.399, no.1-2, pp.141–156, 2008.
[5] A. Ta-Shma and U. Zwick, "Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences," ACM Transactions on Algorithms (TALG), vol.10, no.3, pp.12:1–15, 2014.
[6] Y. Dieudonné, A. Pelc, and D. Peleg, "Gathering despite mischief," ACM Transactions on Algorithms (TALG), vol.11, no.1, pp.1:1–28, 2014.
[7] M. Tsuchida, F. Ooshita, and M. Inoue, "Byzantine-tolerant gathering of mobile agents in arbitrary networks with authenticated whiteboards," IEICE TRANSACTIONS on Information and Systems, vol.E101-D, no.3, pp.602–610, 2018.
[8] G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro, "Asynchronous deterministic rendezvous in graphs," Theoretical Computer Science, vol.355, no.3, pp.315–326, 2006.
[9] Y. Dieudonné, A. Pelc, and V. Villain, "How to meet asynchronously at polynomial cost," SIAM Journal on Computing, vol.44, no.3, pp.844–867, 2015.
[10] E. Kranakis, D. Krizanc, and E. Markou, "The mobile agent rendezvous problem in the ring," Synthesis Lectures on Distributed Computing Theory, vol.1, no.1, pp.1–122, 2010.
[11] A. Pelc, "Deterministic rendezvous in networks: A comprehensive survey," Networks, vol.59, no.3, pp.331–347, 2012.
[12] S. Bouchard, Y. Dieudonné, and B. Ducourthial, "Byzantine gathering in networks," Distributed Computing, vol.29, no.6, pp.435–457, 2016.
[13] J. Czyzowicz, A. Kosowski, and A. Pelc, "How to meet when you forget: log-space rendezvous in arbitrary graphs," Distributed Computing, vol.25, no.2, pp.165–178, 2012.
[14] J. Czyzowicz, A. Kosowski, and A. Pelc, "Time versus space trade-offs for rendezvous in trees," Distributed Computing, vol.27, no.2, pp.95–109, 2014.
[15] P. Fraigniaud and A. Pelc, "Delays induce an exponential memory gap for rendezvous in trees," ACM Transactions on Algorithms (TALG), pp.224–232, 2010.
[16] Y. Sudo, D. Baba, J. Nakamura, F. Ooshita, H. Kakugawa, and T. Masuzawa, "A single agent exploration in unknown undirected graphs with whiteboards," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E98-A, no.10, pp.2117–2128, 2015.
[17] F. Ooshita, S. Kawai, H. Kakugawa, and T. Masuzawa, "Randomized gathering of mobile agents in anonymous unidirectional ring networks," IEEE Transactions on Parallel and Distributed Systems, vol.25, no.5, pp.1289–1296, 2014.
[18] S. Das, F.L. Luccio, and E. Markou, "Mobile agents rendezvous in spite of a malicious agent," International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, vol.9536, pp.211–224, Springer, 2015.
[19] A. Pelc, "Deterministic gathering with crash faults," Networks, vol.72, no.2, pp.182–199, 2018.
[20] J. Chalopin, S. Das, and N. Santoro, "Rendezvous of mobile agents in unknown graphs with faulty links," International Symposium on Distributed Computing, pp.108–122, Springer, 2007.
[21] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to algorithms, MIT press, 2009.

Masashi Tsuchida received the B.E. degree from Osaka Prefecture University College of Technology in 2015, and received M.E. and D.E. degrees from Nara Institute of Science and Technology in 2017 and 2020, respectively. His research interests include distributed algorithms.

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

Michiko Inoue received her B.E., M.E, and Ph.D. degrees in computer science from Osaka University in 1987, 1989, and 1995 respectively. She worked at Fujitsu Laboratories Ltd. from 1989 to 1991. She is a Professor of Graduate School of Science and Technology, Nara Institute of Science and Technology (NAIST). Her research interests include distributed computing and dependability of LSI. She is a member of Science Council of Japan, IPSJ, and JSAI, and a senior member of IEEE.