Gathering of Mobile Agents in Asynchronous Byzantine Environments with Authenticated Whiteboards *

Masashi Tsuchida, Fukuhito Ooshita, and Michiko Inoue

Nara Institute of Science and Technology, Ikoma, Japan

Abstract. We propose two algorithms for the gathering problem 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. Under these assumptions, the first algorithm achieves gathering without termination in $O(m+fn)$ moves per agent (m is the number of edges and n is the number of nodes). The second algorithm achieves gathering with termination 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.

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 treat massive data communication among them. As a way to mitigate the difficulty, (mobile) agents have attracted a lot of attention [2]. Agents are software programs that can autonomously move from a node to a node and execute various tasks in distributed systems. In systems with agents, nodes do not need to communicate with other nodes because agents themselves can collect and analyze data by moving around the network, which simplifies design of distributed systems. In addition, agents can efficiently execute tasks by cooperating with other agents. Hence many works study algorithms to realize cooperation among multiple agents.

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

However, since agents themselves move on the distributed system and might be affected by several nodes that they visit, some of agents might be cracked

* This work was supported by JSPS KAKENHI Grant Number 26330084.

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 agents, τ is the maximum difference among activation times of agents, m is the number of edges, λ is the length of the longest ID of 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).

	Synchronicity	Graph	Byzantine	Whiteboard	Termination	Time complexity
[8][1]	Synchronous	Arbitrary	Absence	None	Possible	$\tilde{O}(n^5\sqrt{\tau l} + n^{10}l)$
[12][1]	Synchronous	Arbitrary	Absence	None	Possible	$\tilde{O}(n^{15} + l^3)$
[18][1]	Synchronous	Arbitrary	Absence	None	Possible	$\tilde{O}(n^5 l)$
[9]	Synchronous	Arbitrary	Presence	None	Possible	$\tilde{O}(n^9\lambda)$
[19]	Synchronous	Arbitrary	Presence	Authenticated	Possible	$O(f_u m)$
[7][1]	Asynchronous	Infinite lines	Absence	None	Possible	$O((D+\lambda)^3)$
[7][1]	Asynchronous	Rings	Absence	None	Possible	$O((n\lambda))$
[10][1]	Asynchronous	Arbitrary	Absence	None	Possible	$poly(n, l)$
Trivial	Asynchronous	Arbitrary	Absence	Existence	Possible	$O(m)$
Proposed 1	Asynchronous	Arbitrary	Presence	Authenticated	Impossible	$O(m + fn)$
Proposed 2	Asynchronous[2]	Arbitrary	Presence	Authenticated	Possible	$O(m + fn)$

and do not follow the algorithm. We call such agents Byzantine agents. An Byzantine agent is supposed 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 Byzantine agents.

1.1 Related works

The gathering problem has been widely studied in literature [13][16]. Table 1 summarizes some of the results. In this table, we show the number of moves for an agent as the time complexity for asynchronous models. These works aim to clarify solvability of the gathering problem in various environments, and, if it is solvable, they aim to clarify optimal costs (e.g., time, number of moves, and memory space) required to achieve 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 presence of node memory (whiteboard).

For synchronous networks, many deterministic algorithms to achieve gathering have been proposed [1][8][12][18]. If agents do not have unique IDs, they cannot gather in symmetric graphs such as rings because they cannot break symmetry. Therefore, some works [8][12][18] assume unique IDs to achieve gathering for any graph. Dessmark et al. [8] proposed an algorithm that realizes gathering in $\tilde{O}(n^5\sqrt{\tau l} + n^{10}l)$ unit times for any graph, where n is the number of nodes,

[1] This algorithm is originally proposed for a rendezvous problem (i.e., gathering of two agents). However, it can be easily extended to the gathering problem by a technique in [12] and its time complexity is not changed.

[2] Agents on a single node are synchronized.

l is the length of the smallest ID of agents, and τ is the maximum difference between activation times of two agents. Kowalski et al. [12] and Ta-Shma et al. [18] improved the time complexity to $\tilde{O}(n^{15} + l^3)$ and $\tilde{O}(n^5 l)$ respectively, which are independent of τ. On the other hand, some works [4, 5, 11] studied the case that agents have no unique IDs. In this case, gathering is not solvable for some graphs and initial positions of agents. So the works proposed algorithms only for solvable graphs and initial positions. They proposed memory-efficient gathering algorithms for trees [5, 11] and arbitrary graphs [4].

If a whiteboard exists on each node, the time required for gathering can be significantly reduced. Whiteboards are areas prepared on each node, and agents can leave information to them. For example, when agents have unique IDs, they can write their IDs into whiteboards on their initial nodes. Agents can collect all the IDs by traversing the network [14], and they can achieve gathering by moving to the initial node of the agent with the smallest ID. This trivial algorithm achieves gathering in $O(m)$ unit times, where m is the number of edges. On the other hand, when agents have no unique IDs, gathering is not trivial even if they use whiteboards and randomization. Ooshita et al. [15] clarified the relationship between solvability of randomized gathering and termination detection in ring networks with whiteboards.

Recently, some works have considered gathering in the presence of Byzantine agents in synchronous networks [1][9][19]. Byzantine agents can make an arbitrary behavior without following the algorithm due to system errors, cracking, and so on. Dieudonné et al. [9] proposed an algorithm to achieve gathering in Byzantine environments in $\tilde{O}(n^9\lambda)$ unit times, where λ is the length of the longest ID of agents. Bouchard et al. [1] minimized the number of correct agents required to achieve gathering, but the time required for gathering is exponential of the number of nodes and labels of agents. Tsuchida et al. [19] reduced the time complexity to $O(f_u m)$ unit times by using authenticated whiteboards, 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, in which each agent is given a dedicated area to write information. They assumed that agents can sign information to write. In their algorithm, correct agents achieve gathering and declare the termination.

For asynchronous networks, many works consider the gathering problem with additional assumptions. De Marco et al. [7] proposed an algorithm to achieve a gathering of 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 two agents in the initial configuration. In rings, they proposed an algorithm to achieve a gathering in $O(n\lambda)$ moves. Dieudonné et al. [10] considered a gathering problem for arbitrary graphs under the same assumptions as [7]. They realized a gathering in polynomial moves of the number of nodes and the length of the minimum ID of agents.

Das et al. [6] assumed the ability of Byzantine agents different from works in [1][9][19], and they realized the gathering in asynchronous ring and mesh networks with Byzantine agents. In their model, correct agents can distinguish Byzantine 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 gathering in $O(n)$ moves in this model.

Pelc [17] considered the gathering problem with crash faults under a weak synchronization model. Pelc considered a model in which each agent moves at a constant speed, but the moving speed is different. That is, although each agent has the same rate clock, the agent cannot know the number of clocks required for movement of other agents. In this work, some agents may become crashed, that is, they may fail and stop at a node or an edge. Under this assumption, Pelc proposed algorithms to achieve the gathering in polynomial time for two cases: agents stop with or without keeping their memory contents.

In other failure models, Chalopin et al. [3] considered a gathering problem with an asynchronous model in which not agents but edges of the networks become faulty. Chalopin et al. considered the case that some of the edges in the network are dangerous or faulty such that any agent travelling along one of these edges would disappear. They proposed an algorithm to achieve gathering in $O(m(m+k))$ moves in this model and they proved that this cost is optimal, where k is the number of agents.

1.2 Our contributions

In this work, we propose two algorithms to achieve the gathering in asynchronous networks with Byzantine agents. In the first algorithm, we adopt the same model as Tsuchida et al. [19] except synchronicity. That is, Byzantine agents exist in an asynchronous network, and an authenticated whiteboard is equipped on each node. Since most of recent distributed systems are asynchronous, we can apply the proposed algorithm to many systems compared to previous algorithms for synchronous networks. To the best of our knowledge, there are no previous works for asynchronous networks with Byzantine agents. If Byzantine agents do not exist, we can use the trivial algorithm with whiteboards in asynchronous networks. That is, agent can achieve the gathering in $O(m)$ moves by using whiteboards in asynchronous networks. However, this trivial algorithm does not work when Byzantine agents exist. The first algorithm is an algorithm that achieves the gathering without termination by using authenticated whiteboards even if Byzantine agents exist in asynchronous networks. The proposed algorithm realizes the gathering in without termination at most $2m + 4n + 10fn = O(m + fn)$ moves.

The second algorithm realizes gathering with termination by putting additional assumptions. By realizing termination, it is possible to notify the upper layer application of the terminating, which simplifies design of distributed systems. In order to realize this, we assume that agents on the same node are synchronized. This assumption is practical and easy to implement because it is easy to specify the timing when the node executes the algorithm of the agents.

In addition, we assume $f < \lceil \frac{1}{3}k \rceil$ holds and agents know k. Under these assumptions, this algorithm achieves gathering with termination in $O(m + fn)$ moves per agent.

2 Preliminaries

2.1 A 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.

Each node does not have a unique ID. 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 to that area. On the other hand, each agent can read information from all areas (including areas of other agents) in the whiteboard.

2.2 A 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 (Section 3), each agent knows neither n nor k. In the second algorithm (Section 4), each agent knows k but does 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 values of variables in its memory. The second element δ is the state transition function that decides the behavior of an agent. The input of δ is the states of all agents on the current node, 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 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 at the same time. The atomic operation of an agent selected by the adversary is shown below.

- If agent 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 gets 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 move to an edge as a result of the local computation, it leaves v.

– If agent a_j is selected at edge e, a_j arrives at the destination node as an atomic operation. That is, a_j arrive at node.

In the first algorithm (Section 3), we assume that agents operate in an asynchronous manner. To guarantee a progress, we assume that for any agent a, the adversary chooses a infinitely many times. In the second algorithm (Section 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 the node v at the same time.

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

2.3 Signature

Each agent a_i can make a signed information that guarantees its ID ID_i and its current node v by a signature function $Sign_{i,v}()$. That is, any agent identifies an ID of the signed agent and whether it is signed at the current node or not from the signature. We assume a_i can use signature function $Sign_{i,v}()$ at only v. We call the output of signature function a *marker*, and denote a marker signed by a_i at node v by $marker_{i,v}$. 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 stay at v. Any agents can copy the marker and can paste any whiteboard, but cannot modify it while keeping its validity.

In this paper, when algorithms treats a marker, it first checks the validity of signatures and ignores the marker if it includes wrong signatures. We omit this behavior from descriptions, and assume all signatures of every marker are valid.

When a_i creates the signed marker at node v, the marker contains ID_i and information of the 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 is 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 the agent a_j stays at node v, a_j can recognize that $marker_{i,v}$ was created at v, and when a_j stays 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. Each Byzantine agent behaves arbitrarily without following the algorithm. However, each 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, and therefore a_i cannot create $marker_{j,u}(i \neq j$ or $v \neq u)$. In this paper, we assume that each agent do not know number of Byzantine agents exist. We assume f Byzantine agents exist. In the second algorithm, we assume $f < \lceil \frac{1}{3}k \rceil$ holds.

2.5 The gathering problem

We consider two types of gathering problems, gathering without termination and gathering with termination. We say an algorithm solves gathering without termination if all correct agents meet at a single node and continue to stay at the node after a certain point of time. In the second problem, we require agents to declare termination. Once an agent declares termination, it can neither change its state nor move to another node after that. We say an algorithm solves gathering with termination if all correct agents meet at a single node and declare termination at the node. We assume that, in the initial configuration, each agent stays at an arbitrary different node. To evaluate the performance of the algorithm, we consider the maximum number of moves required for an agent to achieve the gathering.

2.6 Procedure DFS

In this subsection, we introduce a procedure depth-first search (DFS) used in our algorithm. The DFS is a well-known technique to explore a graph. In the 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 enters 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, since each agent can realize the DFS by using only its dedicated area on whiteboard, Byzantine agents cannot disturb the DFS of correct agents. In this paper, when algorithms executes DFS, each agent use only its dedicated area on whiteboard. We omit this area on whiteboard.

3 Gathering Algorithm without declaring termination

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

3.1 Our algorithm

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

When agent a_i starts the execution on node v_{start}, a_i creates a marker $makrer_{i,v_{start}}$ indicating that a_i starts from node 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 creates the marker. In this algorithm, all agents share their starting markers and then meet at the node where the agent with the minimum ID creates the starting marker.

To share the starting marker, a_i executes DFS and leaves a copy of the marker to all nodes. When agent a_i sees 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 markers of correct agents and may have some markers of Byzantine agents. After that, a_i selects the 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 the marker $marker_{min,v_{min}}$ is created.

However, if Byzantine agents exist, they may interfere with the gathering in various ways. For example, Byzantine agents might not make 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. By these operations, agents may calculate different gathering nodes. To overcome this problem, in this algorithm, each agent shares information on the starting marker created by the agent with the minimum ID with all agents to get a common marker. If all correct agents get 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 agent share all markers of Byzantine agents, they may move infinite times to share the markers because Byzantine agents can create markers infinite times. To prevent from such interference, each agent also shares an 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 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 the marker $marker_{min,v}$ created by the agent a_{min} with minimum ID. If other agents copy marker $marker_{min,u}(v \neq u)$ and paste it to the node v, a_i can judge that the two markers $marker_{min,v}$ and $marker_{min,u}$ were created by the same ID agent. Since the starting marker has been signed, each agent cannot camouflage the starting marker of other agents. In addition, correct agents create the markers only once when they start the algorithms. Therefore, when there are two starting markers $marker_{min,v}$ and $marker_{min,u}(v \neq u)$ created by single agent a_{min}, a_i can distinguish that a_{min} is a Byzantine agent. When a_i understands 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 paste them to all the nodes so that all other agents also judge 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 among the agents not in the blacklist. By these operations, all agents can select the node with the marker as the common gathering node.

Algorithm 1 Algorithm code of agent a_i. The node v indicates the node which a_i is staying.

1: $marker_{i,v} = Sign_{i,v}(),\ a_i.marker = marker_{i,v},\ a_i.All = \emptyset,\ a_i.state =$ `explore`
2: **while** a_i *is executing* DFS **do**
3: $v.wb[ID_i] = \{a_i.marker\}$
4: $a_i.All = a_i.All \cup \bigcup_{id} v.wb[id]$
5: *Store network topology*
6: *Move to the next node by* DFS
7: **end while**
8: $a_i.t_{min} = null,\ a_i.min = \infty,\ a_i.Byz = \emptyset,\ a_i.T_{Byz} = \emptyset$
9: **while** $True$ **do**
10: $a_i.All = a_i.All \cup \bigcup_{id} v.wb[id]$
11: $min\_tmp = min\{writer(t) : t \in a_i.All \wedge writer(t) \notin a_i.Byz\}$
12: **if** $a_i.min > min\_tmp$ **then**
13: $a_i.state =$ `explore`
14: $a_i.t_{min} = t\ s.t.\ t \in a_i.All \wedge writer(t) == min\_tmp$
15: $a_i.min = min\_tmp$
16: **while** a_i *goes around the network* **do**
17: $v.wb[ID_i] = v.wb[ID_i] \cup \{a_i.t_{min}\}$
18: *Move to the next node*
19: **end while**
20: *Move to the node where* $a_i.t_{min}$ *is created*
21: **else**
22: **if** $\exists x : x \in a_i.All \wedge writer(x) == a_i.min \wedge node\_check(x) == false$ **then**
23: $a_i.state =$ `explore`
24: $a_i.T_{Byz} = \{x, a_i.t_{min}\}$
25: **while** a_i *goes around the network* **do**
26: $v.wb[ID_i] = v.wb[ID_i] \cup a_i.T_{Byz}$
27: *Move to the next node*
28: **end while**
29: $a_i.Byz = a_i.Byz \cup a_i.min$
30: $a_i.min = \infty$
31: **else**
32: $a_i.state =$ `gather`
33: *Stay at the node* v
34: **end if**
35: **end if**
36: **end while**

Since we consider an asynchronous network, agent a_i does not know when other agents write 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 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.

Details of the Algorithm The pseudo-code of the algorithm is given in Algorithm 1. We denote by $v.wb[ID_i]$ the dedicated writable area of agent a_i in the

whiteboard on node v. Agent a_i manages the 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 the 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 an agent with minimum ID except Byzantine agents' ID that a_i has observed so far. 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 Byzantine agent IDs that a_i has confirmed so far. The initial values of these variables 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 creates $marker_{i,v}$. Function $node\_check(marker_{i,v})$ returns true if $marker_{i,v}$ was created on the current node, and otherwise returns false.

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 pseudocode, each agent executes the operations as an atomic operation until it leaves (lines 6, 18, 20 and 27) or it decides to stay (line 33). 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 makes starting marker $marker_{i,v} = Sign_{i,v}()$ and becomes `explore` (line 1). After a_i creates the starting marker, in order to inform other agents about the marker, a_i executes DFS and copies the marker and pastes it to all nodes (line 2 to 7). On every node, a_i adds other agent's marker to $a_i.All$ (line 4). In order 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 with at most $2n$ moves after it finishes DFS.

After a_i finishes DFS, a_i checks the markers collected in $a_i.All$ and calculates a gathering node (lines 9 to 36). First, a_i stores markers of the current node in $a_i.All$ to check 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 11). If $a_i.min > ID_{min}$, a_i executes an update operation of a gathering node (lines 12 to 20). Otherwise, a_i executes a detection operation of Byzantine agents (lines 22 to 30).

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 a_i pastes it to all nodes in order to inform other agents of that minimum ID agent's marker (lines 16 to 19). Note that, since a_i knows the graph topology, it can visit all nodes in at most $2n$ moves. In addition, since a_i visits all nodes, a_i can know at which node $a_i.t_{min}$ was created. 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 determines an agent with the minimum ID as a Byzantine agent.

In detection operation of Byzantine agents, 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, since correct agents create markers only once, 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 a next while-loop. In order 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 decides the current node as a gathering node, a_i changes the $a_i.state$ to `gather` state. After that, if a_i decides to change the gathering node, a_i changes $a_i.state$ to `explore` again (lines 13 and 23).

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

3.2 Correctness of Algorithm 1

Lemma 1. *Correct agents 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 creates 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 DFS, 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 execution of the algorithm to $a_i.All$. 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, since a_i itself is also a correct agent, ID ID_i is never stored in $a_i.Byz$. Therefore, after a_i finishes DFS, $marker_{i,v_{start}} \in a_i.All \wedge writer(marker_{i,v_{start}}) \notin a_i.Byz$ holds, which implies the lemma.

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

Proof. While correct agent a_i executes DFS at the beginning of the algorithm, a_i can observe all markers of correct agents. Therefore, when a_i calculates a

gathering node for the first time, $a_i.min$ becomes the minimum ID among 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 judges $a_i.min$ is an ID of Byzantine agent. Since a_i observes all markers of correct agents during DFS, a_i can observe a marker t satisfying $a_i.min > writer(t)$ only when $writer(t)$ is an ID of Byzantine agent. In addition, after a_i judges $a_i.min$ is an ID of Byzantine agent, a_i never updates $a_i.min$ by using the 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.

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.

In order 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 $marker_{x,v}$. Since $ID_j \notin a_j.Byz$ holds from Lemma 1, $a_j.min \leq x$ holds after a_j observes marker $marker_{x,v}$. Therefore, after the last updates of $a_i.min$ and $a_j.min$, $y = a_j.min \leq x = a_i.min$ holds. This is contradiction. Thus, the lemma holds.

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

Proof. Correct agent a_i finds node v such that the marker was created by $a_i.min$'s agent, and then sets v as a gathering node. From Lemma 2, there is a gathering node. Also, 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. 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. Therefore, all correct agents refer to the same marker $marker_{min,v}$, and calculate the same node v as a gathering node. Moreover, since the time required for agents to move between nodes is also finite, all correct agents can arrive at a gathering node in a finite time. In addition, if correct agent decides not to change gathering node, correct agent 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 $2m + 4n + 10fn$ times.*

Proof. From Lemma 5, all correct agents gather at one node with `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 $2n$ 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 $2n$ moves.

Every time a_i updates $a_i.min$, a_i copies the starting marker and pastes it to all nodes with at most $2n$ moves and moves to a new gathering node with at most $2n$ moves. From Lemma 3, a_i updates $a_i.min$ at most $2f$ times. In addition, when the minimum ID agent is judged as 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 with at most $2n$ moves. Therefore, a_i moves at most $2m + 2n + 2n + 2f(2n + 2n) + f \times 2n = 2m + 4n + 10fn$ times.

4 Gathering algorithm with declaring termination

In this section, we propose an algorithm that solves gathering with termination. To realize the algorithm we add 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, since 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

Overview First, we give an overview of our algorithm. This algorithm achieves gathering with termination 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 `gather` state. After at least $k - f_u$ agents of `gather` state gather at one node v, all correct agents at v terminate. Note that, since the $k - f_u$ agents execute the algorithm in 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 where at least $f_u + 1$ agents have terminated. Thus, all correct agents gather at one node and terminate.

Details of the Algorithm The pseudo-code of the algorithm is given in Algorithm 2. It is basically the same as Algorithm 1, but differences are additional lines 10 to 12 and 21 to 23. 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 6, 26, 28 and 35) or it decides to stay (line 41) or it declare termination (lines 12 and 23). When an agent reads from the whiteboard, it uses the snapshot taken at the beginning of an atomic operation.

Algorithm 2 *main*() Algorithm code of agent a_i. The node v indicates the node which a_i is staying.

```
1: marker_{i,v} = Sign_{i,v}(), a_i.marker = marker_{i,v}, a_i.All = ∅, a_i.state = explore
2: while a_i is executing DFS do
3:    v.wb[ID_i] = {a_i.marker}
4:    a_i.All = a_i.All ∪ ⋃_id v.wb[id]
5:    Store network topology
6:    Move to the next node by DFS
7: end while
8: a_i.t_min = null, A_i.min = ∞, a_i.Byz = ∅, a_i.T_Byz = ∅
9: while true do
10:   if There exist at least k − f_u agents of gather state at node v then
11:      a_i.state = terminate
12:      declare termination
13:   else
14:      a_i.All = a_i.All ∪ ⋃_id v.wb[id]
15:      min_tmp = min{writer(t) : t ∈ a_i.All ∧ writer(t) ∉ a_i.Byz}
16:      if a_i.min > min_tmp then
17:         a_i.state = explore
18:         a_i.t_min = t s.t. t ∈ a_i.All ∧ writer(t) == min_tmp
19:         a_i.min = min_tmp
20:         while a_i goes around the network do
21:            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
22:               a_i.state = terminate
23:               declare termination
24:            end if
25:            v.wb[ID_i] = v.wb[ID_i] ∪ {a_i.t_min}
26:            Move to the next node
27:         end while
28:         Move to the node where a_i.t_min is created
29:      else
30:         if ∃x : x ∈ a_i.All ∧ writer(x) == a_i.min ∧ node_check(x) == false then
31:            a_i.state = explore
32:            a_i.T_Byz = {x, a_i.t_min}
33:            while a_i goes around the network do
34:               v.wb[ID_i] = v.wb[ID_i] ∪ a_i.T_Byz
35:               Move to the next node
36:            end while
37:            a_i.Byz = a_i.Byz ∪ a_i.min
38:            a_i.min = ∞
39:         else
40:            a_i.state = gather
41:            stay at the node v
42:         end if
43:      end if
44:   end if
45: end while
```

In Algorithm 2, agents execute the same operations as Algorithm 1 until at least $k - f_u$ agents of `gather` state gather at its current node v. After at least $k - f_u$ agents of `gather` state gather at node v, correct agents terminate at the node v (lines 10 to 12). If agent a_i sees at least $k - f_u$ agents of `gather` state or at least $f_u + 1$ agents of `terminate` state at node v, a_i terminates at v (lines 21 to 23). Agent a_i executes the above operation while a_i visits all nodes to paste marker $a_i.t_{min}$ for updating the gathering node. Note that a_i does not execute the operation while a_i visits nodes to paste $a_i.t_{Biz}$ for updating the blacklist of Byzantine agents (lines 30 to 38). 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, eventually all the correct agents refer to the starting marker created by the minimum ID agent and gather at the same node with declaring termination.

4.2 Correctness of Algorithm 2

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 10 or 21 as true. Since at most f_u Byzantine agents exist, a does not see $f_u + 1$ agents of `terminate` state at v and thus it never evaluates the latter half of the predicate of line 21 as true. Consequently, when a terminates, it sees at least $k - f_u$ agents of `gather` state at v. From $f_u = \lfloor \frac{k-1}{3} \rfloor$, $k - 2f_u \geq f_u + 1$ agents among at least $k - f_u$ agents are correct. Since all agents on the same node are synchronized, at least $f_u + 1$ correct agents of `gather` state execute lines 10 to 12 (or 21 to 23) at the same time. They also see at least $k - f_u$ agents of `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 2, a correct agent terminates if and only if it evaluates the predicate of line 10 or 21 as true. This implies no correct agent evaluates the predicate as true.

Recall that only lines 10 to 12 and 21 to 23 in Algorithm 2 are added to Algorithm 1. Thus, if predicates of lines 10 and 21 of Algorithm 2 are always false, agents make the same behaviors as Algorithm 1. Consequently, from Theorem 1, all correct agents gather at some 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 10 in Algorithm 2 as true. This is a contradiction.

We define a_f as the correct agent that terminates earliest among 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 10 and 21 as false (Otherwise they terminate). Consequently, all correct agents make 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 correct agent that terminates at v' earliest.

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 of `gather` state at v'. However, since 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 of* `terminate` *state exist at v_f. For any node v' ($v' \neq v_f$), the number of agents of* `terminate` *state at v' is at most f_u.*

Proof. Since $f_u + 1$ 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 DFS. 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 where $f_u + 1$ agents terminate.

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

- Case that, a_f does not considers $id_{v'}$ as 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 exists at v_f, a_f moves to the corresponding node as a new gathering node. Since a must paste a marker of $id_{v'}$ or a smaller ID to all nodes before entering `gather` state, a visits v_f after t_f.

– Case that, a_f considers $id_{v'}$ as 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 terminates. Consequently, v' contains two markers of $id_{v'}$. At time t_f, agent a executes an update operation of the gathering node v' or a detection operation of Byzantine agents. 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 $id_{v'}$ is an ID of a Byzantine agent. Consequently, a pastes two markers of $id_{v'}$ to all nodes and then executes an update operation of a new gathering node. During the update operation, a visits v_f. In the latter case, a has already known $id_{v'}$ 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 an update operation of 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 pastes 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 detection operation of Byzantine agents. In the former case, after a completes 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 of 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 of 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. Then, a executes an update operation of 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 already terminate. From Lemma 6, a sees at least $f + 1$ agents of `terminate` state and terminate 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 2 achieves a gathering with termination within a finite time. In the algorithm, each agent moves at most $O(m + fn)$.*

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 declare termination, and each agent moves $O(m + fn)$ times.

5 Conclusions

In this work, we have proposed two gathering algorithms for mobile agents in asynchronous Byzantine environments with authenticated whiteboards. Each algorithm achieves the gathering in $O(m + fn)$ moves per an agent. In the Algorithm 1 achieves the gathering without termination. In the Algorithm 2 realizes

termination by putting additional assumptions. The additional assumptions are that agents on a single node are synchronized, each agent knows f and k where f is number of Byzantine agents and k is number of total agents.

References

1. Bouchard, S., Dieudonné, Y., Ducourthial, B.: Byzantine gathering in networks. Distributed Computing 29(6), 435–457 (2016)
2. Cao, J., Das, S.K.: Mobile Agents in Networking and Distributed Computing. Wiley (2012)
3. Chalopin, J., Das, S., Santoro, N.: Rendezvous of mobile agents in unknown graphs with faulty links. In: International Symposium on Distributed Computing. pp. 108–122. Springer (2007)
4. Czyzowicz, J., Kosowski, A., Pelc, A.: How to meet when you forget: log-space rendezvous in arbitrary graphs. Distributed Computing 25(2), 165–178 (2012)
5. Czyzowicz, J., Kosowski, A., Pelc, A.: Time versus space trade-offs for rendezvous in trees. Distributed Computing 27(2), 95–109 (2014)
6. Das, S., Luccio, F.L., Markou, E.: Mobile agents rendezvous in spite of a malicious agent. In: International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics. pp. 211–224. Springer (2015)
7. De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous deterministic rendezvous in graphs. Theoretical Computer Science 355(3), 315–326 (2006)
8. Dessmark, A., Fraigniaud, P., Kowalski, D.R., Pelc, A.: Deterministic rendezvous in graphs. Algorithmica 46(1), 69–96 (2006)
9. Dieudonné, Y., Pelc, A., Peleg, D.: Gathering despite mischief. ACM Transactions on Algorithms (TALG) 11(1), 1:1–28 (2014)
10. Dieudonné, Y., Pelc, A., Villain, V.: How to meet asynchronously at polynomial cost. SIAM Journal on Computing 44(3), 844–867 (2015)
11. Fraigniaud, P., Pelc, A.: Delays induce an exponential memory gap for rendezvous in trees. ACM Transactions on Algorithms (TALG) 9(2), 17:1–24 (March 2013)
12. Kowalski, D.R., Malinowski, A.: How to meet in anonymous network. Theoritical Computer Science 399(1-2), 141–156 (2008)
13. Kranakis, E., Krizanc, D., Markou, E.: The mobile agent rendezvous problem in the ring. Synthesis Lectures on Distributed Computing Theory 1(1), 1–122 (2010)
14. Nakamura, J., Ooshita, F., Kakugawa, H., Masuzawa, T.: A single agent exploration in unknown undirected graphs with whiteboards. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 98(10), 2117–2128 (2015)
15. Ooshita, F., Kawai, S., Kakugawa, H., Masuzawa, T.: Randomized gathering of mobile agents in anonymous unidirectional ring networks. IEEE Transactions on Parallel and Distributed Systems 25(5), 1289–1296 (2014)
16. Pelc, A.: Deterministic rendezvous in networks: A comprehensive survey. Networks 59(3), 331–347 (2012)
17. Pelc, A.: Deterministic gathering with crash faults. Networks (2018)
18. Ta-Shma, A., Zwick, U.: Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. ACM Transactions on Algorithms (TALG) 10(3), 12:1–15 (2014)

19. Tsuchida, M., Ooshita, F., Inoue, M.: Byzantine gathering in networks with authenticated whiteboards. In: International Workshop on Algorithms and Computation. pp. 106–118. Springer (2017)