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

Masashi TSUCHIDA, Nonmember, Fukuhito OOSHITA, Member, and Michiko INOUE, Senior Member

**SUMMARY** We propose an algorithm for the gathering problem of mobile agents in arbitrary networks (graphs) with Byzantine agents. Our algorithm can make all correct agents meet at a single node in $O(fm)$ time ($f$ is the upper bound of the number of Byzantine agents and $m$ is the number of edges) under the assumption that agents have unique ID and behave synchronously, each node is equipped with an authenticated whiteboard, and $f$ is known to agents. Here, the whiteboard is a node memory where agents can leave information. Since the existing algorithm achieves gathering without a whiteboard in $O(n^6 l)$ time, where $n$ is the number of nodes and $l$ is the length of the longest ID, our algorithm shows an authenticated whiteboard can significantly reduce the time for the gathering problem in Byzantine environments.

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

### 1. Introduction

#### 1.1 Background

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. 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 within a finite time. By achieving gathering, all agents can communicate with each other at the single node.

---

**1.2 Related Works**

The gathering problem has been widely studied in literature [8], [9]. Most studies aim to clarify solvability of the gathering problem in various environments, and, if it is solvable, they aim to clarify costs (e.g., time, number of moves, and memory space) required to achieve gathering. To do this, many studies have been conducted under various environments such that assumptions on synchronization, anonymity, randomized behavior, topology, and presence of node memory (whiteboards) are different. Table 1 summarizes some of the results.

For environments such that no whiteboard exists (i.e., agents cannot leave any information on nodes), many deterministic algorithms to achieve gathering of two agents have been proposed. Note that these algorithms can be easily extended to a case of more than two agents [4]. If agents do not have unique IDs, they cannot achieve gathering for some symmetric graphs. Therefore some works [3]–[5] assume unique IDs and achieve gathering for any graph. Dessmark et al. [3] proposed an algorithm that realizes gathering in $O(n^5 \sqrt{l} + n^{10} l)$ time for any graph, where $n$ is the number of nodes, $l$ is the length of the smaller ID of agents, and $\tau$ is the difference between activation times of agents. Kowalski et al. [4] and Ta-Shma et al. [5] improved the time complexity to $O(n^5 + l^2)$ and $O(n^5 l)$ respectively, which are independent of $\tau$. On the other hand, some works [10]–[12] 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-

| Table 1 | Gathering of synchronous agents with unique IDs in arbitrary graphs ($n$ is the number of nodes, $l$ is the length of the smallest ID of agents, $\tau$ is the maximum difference among activation times of agents, $m$ is the number of edges, $\lambda$ is the length of the longest ID of agents, $f$ is the upper bound of the number of Byzantine agents). |
|--------|--------|-----------------|
| Byzantine | Whiteboard | Time complexity |
| [3] | None | None | $O(n^5 \sqrt{l} + n^{10} l)$ |
| [4] | None | None | $O(n^5 + l^2)$ |
| [5] | None | Non-authenticated | $O(n^5 l)$ |
| [6] | Weak | None | $O(n^5 \lambda)$ |
| Trivial extension of [6] | Weak | Authenticated | $O(n^5 \lambda)$ |
| Proposed algorithm | Weak | Authenticated | $O(fm)$ |

---

Copyright © 2018 The Institute of Electronics, Information and Communication Engineers
efficient gathering algorithms for trees [10], [11] and arbitrary graphs [12].

If a whiteboard exists on each node, the time required for gathering can be significantly reduced. 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 [13], and thus they can achieve gathering by moving to the initial node of the agent with the smallest ID. This trivial algorithm achieves gathering in $O(m)$ time, 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. [14] clarified the relationship between solvability of randomized gathering and termination detection in ring networks with whiteboards.

Recently some works [6], [7] have considered gathering in the presence of Byzantine agents, which can behave arbitrarily. They modeled agents controlled by crackers or corrupted by software errors as Byzantine agents. These works assume agents have unique IDs, behave synchronously, and cannot use whiteboards. They consider two types of Byzantine agents. While a weakly Byzantine agent can make arbitrary behavior except falsifying its ID, a strongly Byzantine agent can make arbitrary behavior including falsifying its ID. Dieudonné et al. [6] proposed algorithms to achieve gathering in arbitrary graphs against weakly Byzantine agents and strongly Byzantine agents, both when the number of nodes $n$ is known and when it is unknown. For weakly Byzantine agents, when $n$ is known, they proposed an algorithm that achieves gathering in $4n^4 \cdot P(n, \lambda)$ time, where $P(n, \lambda)$ is the time required for gathering of two correct agents with unique IDs ($\lambda$ is the length of the smaller given ID) and $\lambda$ is the length of the longest ID among all agents. Since two agents can meet in $P(n, \lambda) = O(n^\lambda)$ time [5], the algorithm achieves gathering in $O(n^\lambda)$ time. For weakly Byzantine agents, when $n$ is unknown, they also proposed a polynomial-time algorithm. However, for strongly Byzantine agents, they proposed only exponential-time algorithms. Bouchard et al. [7] minimized the number of correct agents required to achieve gathering for strongly Byzantine agents, however the time complexity is still exponential.

1.3 Our Contributions

The purpose of this work is to reduce the time required for gathering by using a whiteboard on each node. However, if Byzantine agents can erase all information on whiteboards, correct agents cannot see the information and thus whiteboards are useless. For this reason, we assume that an authentication function is available on the system and this provides authenticated whiteboards. In authenticated whiteboards, each agent is given a dedicated area to write information. In other words, each agent can write information to the dedicated area and cannot write to other areas. Regarding read operations, each agent can read information from all areas on the whiteboards. In addition, we assume, by using the authentication function, each agent can write information with signature that guarantees the writer and the writing node.

No gathering algorithms have been proposed for environments with whiteboards in the presence of Byzantine agents. However, since two agents can meet quickly by using authenticated whiteboards, the time complexity of an algorithm in [6] can be reduced. More specifically, each agent can explore the network in $O(m)$ time by the depth-first search (DFS), and after the first exploration it continues to explore the network in $O(n)$ time for each exploration. By applying this to Dessmark’s algorithm [3], two agents can meet in $P(n, l) = O(nl)$ time. Thus, for weakly Byzantine agents, agents can achieve gathering in $O(n^5 \lambda)$ time.

In this work, we propose a new algorithm to achieve gathering in shorter time. Similarly to [6], we assume agents have unique IDs and behave synchronously. When at most $f$ weakly Byzantine agents exist and $f$ is known to agents, our algorithm achieves gathering in $O(fm)$ time by using authenticated whiteboards. That is, our algorithm significantly reduces the time required for gathering by using authenticated whiteboards. To realize this algorithm, we newly propose a technique to simulate message-passing algorithms by agents. Our algorithm overcomes difficulty of Byzantine agents by simulating a Byzantine-tolerant consensus algorithm [15]. This technique is general and not limited to the gathering problem, and hence it can be applied to other problems of agents.

2. Preliminaries

2.1 A Distributed System and Mobile Agents

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 $A_e : \{(v, u)|u \in N_v\} \rightarrow \{1, 2, \cdots, d(v)\}$ such that $A_e(v, u) \neq A_e(v, w)$ holds for $u \neq w$. We say $A_e(v, u)$ is a port number (or port) of edge $(v, u)$ on node $v$.

Each node does not have a unique ID. Each node has a 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.

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 length of the ID is $O(log k)$ bits. The ID of agent $a_i$ is denoted by $ID_i$. Each agent knows neither $n$ nor $k$. Each agent can move to an adjacent node by choosing an outgoing port. That is, when an agent stays at $v$ and moves via port $p$, it moves to node $w$ such that $p = A_e(v, w)$ holds. When the agent reaches $w$, it gets the port number $A_e(v, w)$, that is, it
finds from which port it came.

Each agent is modeled as a state machine \((S, \delta)\). The first element \(S\) is the set of agent states, where each agent state is determined by values of variables in its memory. The second element \(\delta\) is the state transition function that decides the behavior of an agent. The input of \(\delta\) is the current agent state, the content of the whiteboard in the current node, and the incoming port number. The output of \(\delta\) 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.

Agents move in synchronous rounds. That is, the time required for each correct agent to move to the adjacent node is identical. In the initial configuration, each agent is inactive and stays at an arbitrary node. Some agents spontaneously become active and start the algorithm. When active agent \(a_i\) encounters inactive agent \(a_j\) at some node \(v\), agent \(a_i\) can make \(a_j\) active. In this case, \(a_i\) starts the algorithm before \(a_i\) executes the algorithm at \(v\).

Each agent \(a_i\) can sign a value \(x\) that guarantees its ID \(ID_i\) and its current node \(v\). 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 \(\text{Sign}_{a_i}(x)\) at \(v\) and we denote the output of \(\text{Sign}_{a_i}(x)\) by \((\text{id}, v)\). Each agent \(a_i\) can compute \(\text{Sign}_{a_i}(x)\) for value \(x\) at \(v\), however cannot compute \(\text{Sign}_{a_i}(x)\) for either \(j \neq i\) or \(w \neq v\). Therefore, it is guaranteed that signed value \(\langle x \rangle : (ID_i, v)\) is created by \(a_i\) at \(v\). For signed value \(x = \langle \text{value} \rangle : (id_1, v_1) : (id_2, v_2) : \cdots : (id_j, v_j)\), the output of \(\text{Sign}_{a_i}(x)\) is denoted by \(\langle \text{value} \rangle : (id_1, v_1) : (id_2, v_2) : \cdots : (id_j, v_j) : (ID_i, v)\). Agents can copy the signed value to any whiteboard, but cannot modify it while keeping its validity. In this paper, when an algorithm treats a signed value, it first checks the validity of signatures and ignores the signed value if it includes wrong signatures. We omit this behavior from descriptions, and assume all signatures of every signed value are valid.

Byzantine agents may exist in a distributed system. Each Byzantine agent behaves arbitrarily without being synchronized with other agents. However, each Byzantine agent cannot change its ID. In addition, even if agent \(a_i\) is Byzantine, \(a_i\) cannot compute \(\text{Sign}_{a_i}(x)(j \neq i)\) for value \(x\), and therefore \(a_i\) cannot create \(\langle x \rangle : (ID_j, v)\) for \(j \neq i\). We assume the number of Byzantine agents is at most \(f < k\) and \(f\) is known to each agent.

2.2 The Gathering Problem

The gathering problem is a problem to make all correct agents meet at a single node and declare termination. In the initial configuration, each agent stays at an arbitrary node and multiple agents can stay at a single node. If an agent declares termination, it never works after that.

To evaluate the performance of the algorithm, we consider the time required for all agents to declare termination after some agent starts the algorithm. We assume the time required for a correct agent to move to the adjacent node is one unit time, and we ignore the time required for local computation.


In this section, we explain a Byzantine-tolerant consensus algorithm in [15] that will be used as building blocks in our algorithm.

3.1 A Message-Passing System

The consensus algorithm is proposed in a fully-connected synchronous message-passing system. That is, we assume that processes form a complete network. We assume the time required for all agents to declare termination after some agent starts the algorithm. We assume the number of processes is \(k\) and denote a set of processes by \(P = \{p_1, p_2, \ldots, p_k\}\). Each process has a unique ID, and the ID of \(p_j\) is denoted by \(ID_j\). All processes execute an algorithm in synchronous phases. In the 0-th (or initial) phase, every process computes locally and sends messages (if any). In the \(r\)-th phase \((r > 0)\), every process receives messages, computes locally, and sends messages (if any). If process \(p_i\) sends a message to process \(p_j\) in the \(r\)-th phase, \(p_j\) receives the message at the beginning of \((r + 1)\)-th phase.

Similarly to Sect. 2, each process \(p_j\) has signature function \(\text{Sign}_{p_j}(x)\). The output of \(\text{Sign}_{p_j}(x)\) is denoted by \(\langle x \rangle : (ID_j)\), and only \(p_j\) can compute \(\text{Sign}_{p_j}(x)\).

Some Byzantine processes may exist in the message-passing system. Byzantine processes can behave arbitrarily. But even if \(p_i\) is Byzantine, \(p_i\) cannot compute \(\text{Sign}_{p_j}(x)(j \neq i)\) for value \(x\). We assume the number of Byzantine processes is at most \(f < k\) and \(f\) is known to each process.

3.2 A Byzantine-Tolerant Consensus Algorithm

In this subsection, we explain a Byzantine-tolerant consensus algorithm in [15]. In the consensus algorithm, each process \(p_i\) is given at most one value \(x_i\) as its input. If \(p_i\) is not given an input value, we say \(x_i = \perp\). The goal of the consensus algorithm is to agree on the set of all input values. Of course, some Byzantine processes behave arbitrarily and forge inconsistent input values. However, by the consensus algorithm in [15], all correct agents can agree on the same set \(X \supseteq X_c\), where \(X_c\) is a set of all values input by correct processes.

We show the details of the consensus algorithm. Each process \(p_i\) has one variable \(p_i.W\) to keep a set of input values, and initially \(p_i.W = \emptyset\). The algorithm consists of \(f + 2\) phases (from the 0-th phase to \((f + 1)\)-th phase). After processes terminate, they have the same values in \(W\).

In the 0-th phase, if \(p_i\) is given an input value \(x_i(\neq \perp)\), process \(p_i\) broadcasts \(\text{Sign}_{p_j}(x_i) = \langle x_i \rangle : (ID_j)\) to all processes and adds \(x_i\) to variable \(p_i.W\). If \(p_i\) is not given an input value, it does not do anything.

In the \(r\)-th phase \((1 \leq r \leq f + 1)\), \(p_i\) receives all messages (or signed values) broadcasted in \((r - 1)\)-th phase. After that, for every received message, process \(p_i\) checks its
validity. We say message \( t = \langle x \rangle : id_1 : id_2 : \cdots : id_p \) is valid if and only if \( t \) satisfies all the following conditions.

1. The number \( y \) of signatures in \( t \) is equal to \( r \).
2. All signatures in \( t \) are distinct.
3. Message \( t \) does not contain \( p_i \)’s signature.
4. Value \( x \) is not in \( p_i \). \( W \).

If message \( t = \langle x \rangle : id_1 : id_2 : \cdots : id_p \) is valid, \( p_i \) broadcasts \( Sign(t) = \langle x \rangle : id_1 : id_2 : \cdots : id_q : ID_i \) to all processes (if \( r \leq f \)) and adds \( x \) to variable \( p_i \). \( W \).

For this algorithm, the following theorem holds.

**Theorem 1:** [15] After all processes terminate, all the following holds.

1. For any correct process \( p_i, x_i \in p_i \cdot W \) holds if \( x_i \neq \perp \).
2. For any two correct processes \( p_i \) and \( p_j, p_i \cdot W = p_j \cdot W \) holds.

### 4. Our Algorithm

#### 4.1 Overview

First, we give an overview of our algorithm. When agent \( a_i \) starts the algorithm, \( a_i \) leaves its starting information to a whiteboard at its initial node \( v \). The starting information includes \( ID_i \), and consequently it can notify other agents that \( a_i \) starts at \( v \). After that, \( a_i \) explores the network and collects starting information of all agents. If no Byzantine agent exists, all agents collect the same set of starting information, and thus all agents can meet at a single node by visiting the node where the agent with the smallest ID leaves the starting information.

However, when some Byzantine agent exists, it can write and delete its starting information repeatedly so that only a subset of agents see the information. This implies some agents may obtain a set of starting information different from others and thus may fail to achieve gathering.

To overcome this difficulty, our algorithm makes all correct agents agree on the same set of starting information at each node. That is, letting \( a_i \cdot X_v \) be the set of starting information that \( a_i \) obtains at node \( v \), we guarantee that \( a_i \cdot X_v = a_j \cdot X_v \) holds for any two correct agents \( a_i \) and \( a_j \). In addition, we also guarantee that, if correct agent \( a_i \) starts at \( v \), then \( a_i \cdot X_v \) contains \( a_i \)’s starting information and \( a_i \cdot X_w (w \neq v) \) does not contain \( a_i \)’s starting information. We later explain the details of this procedure.

After that, each agent \( a_i \) can obtain \( a_i \cdot X_{all} = \bigcup_{v \in V} \bigcup_{a: V} a_i \cdot X_v \), and clearly \( a_i \cdot X_{all} = a_j \cdot X_{all} \) holds for any two correct agents \( a_i \) and \( a_j \). Consequently each agent \( a_i \) can compute the same gathering node based on \( a_i \cdot X_{all} \) as follows. First \( a_i \) removes all duplicated starting information from \( a_i \cdot X_{all} \) because a Byzantine agent may leave its starting information at several nodes. After that, \( a_i \) finds the starting information of the agent with the smallest ID and selects the node with the starting information as the gathering node. By this behavior, all correct agents can meet at the same gathering node.

In the rest of this subsection, we explain the way to make all correct agents agree on the same set of starting information at each node. To realize this, our algorithm uses a Byzantine-tolerant consensus algorithm in Sect. 3. At each node, agents simulate the consensus algorithm and then agree on the same set. However, since the consensus algorithm is proposed for synchronous message-passing systems, we need additional synchronization mechanism. We realize this by using the depth-first search (DFS).

#### 4.2 DFS and Round Synchronization

The DFS is a well-known technique to explore a graph. In the DFS, an agent continues to explore a 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 previous node again. By repeating this behavior, each agent can visit all nodes in \( 2m \) unit times, where \( m \) is the number of edges. Note that, since each agent can realize the DFS by using only its dedicated area on whiteboards, Byzantine agents cannot disturb the DFS of correct agents.

To simulate the consensus algorithm, we realize round synchronization of agents by the DFS. More specifically, we guarantee that, before some agent \( a_i \) makes the \( r \)-th visit to \( v \), all agents finish the \((r - 1)\)-th visit to \( v \). To realize this, each agent \( a_i \) executes the following procedure in addition to the DFS.

- If \( a_i \) finds an inactive agent, \( a_i \) makes the agent active.
- Every time \( a_i \) completes a DFS, it waits for the same time as the exploration time. That is, \( a_i \) waits for \( 2m \) unit times after each DFS.

We define the \( r \)-th exploration period of \( a_i \) as the period during which \( a_i \) executes the \( r \)-th DFS exploration, and define the \( r \)-th waiting period of \( a_i \) as the period during which \( a_i \) waits after the \( r \)-th DFS exploration. In addition, we define the \( r \)-th round of \( a_i \) as the period from the beginning of the \( r \)-th exploration period to the end of the \( r \)-th waiting period. As shown in the Fig. 1, before some agent starts the \( r \)-th exploration period, every correct agent completes the \((r - 1)\)-th exploration period.

#### 4.3 Simulation of Consensus Algorithm

In the following, we explain the way to apply the consensus algorithm in Sect. 3. The goal is to make all correct agents agree on the same set of starting information at

![Fig. 1](image-url)
each node. To achieve this, we assume \( k \) virtual processes \( v.p_1, v.p_2, \ldots, v.p_k \) exist at each node \( v \) and form a message-passing system in Sect. 3 (See Fig. 2). When agent \( a_i \) visits node \( v \), it simulates \( v.p_i \)'s behavior of the consensus algorithm.

In the consensus algorithm on node \( v \), each virtual process decides its input value as follows. If \( a_i \) starts the algorithm at \( v \), the input of virtual process \( v.p_i \) is the starting information of \( a_i \). Otherwise, the input of virtual process \( v.p_i \) is not given. Thus, after completion of the consensus algorithm, all virtual processes at \( v \) agree on the same set \( X_v \) of starting information. From the property of the consensus algorithm, \( X_v \) contains starting information of all correct agents that start at \( v \).

Next, we explain how to simulate the behaviors of virtual processes. Each agent \( a_i \) simulates the \( r \)-th phase of virtual process \( v.p_i \) when \( a_i \) visits \( v \) for the first time in the exploration period of \( r \)-th round. Recall that, by the round synchronization, when some correct agent \( a_i \) starts the exploration period of the \( r \)-th round, all correct agents have already completed the exploration period of the \((r - 1)\)-th round. This implies, \( a_i \) can simulate the \( r \)-th phase of virtual process \( v.p_i \) after all virtual processes complete the \((r - 1)\)-th phase.

To simulate \( v.p_i \), \( a_i \) uses variables \( v.wb[ID_i].T \) and \( v.wb[ID_i].W \) in the whiteboard of node \( v \). We denote variable \( var \) in the dedicated area of \( a_i \) by \( v.wb[ID_i].var \). Agent \( a_i \) uses \( v.wb[ID_i].T \) to simulate communications among virtual processes. That is, when \( v.p_i \) sends some messages to other processes, \( a_i \) stores the messages in \( v.wb[ID_i].T \) so that other virtual processes read the messages. Here, to guarantee that the messages are available on only node \( v \), \( a_i \) stores \( Sign_{v.w}(t) \) instead of message \( t \). Agent \( a_i \) uses \( v.wb[ID_i].W \) to memorize variables of \( v.p_i \). By using these variables, \( a_i \) can simulate the \( r \)-th phase of \( v.p_i \) as follows:

1. By reading from all variables \( v.wb[id_i].T \) (for some \( id \), \( a_i \) receives messages that virtual processes have sent to \( v.p_i \) in the \((r - 1)\)-th phase.
2. From \( v.p_i \)'s variables stored in \( v.wb[ID_i].W \) and messages received in 1, agent \( a_i \) simulates local computation of \( v.p_i \)'s \( r \)-th phase.
3. Agent \( a_i \) writes updated variables of \( v.p_i \) to \( v.wb[ID_i].W \). If \( v.p_i \) sends some messages, \( a_i \) writes the messages with signatures to \( v.wb[ID_i].T \).

Note that, since only agent \( a_i \) can update variables \( v.wb[ID_i].T \) and \( v.wb[ID_i].W \), agent \( a_i \) simulates the correct behavior of \( v.p_i \) if \( a_i \) is correct. This implies that the simulated message-passing system contains at most \( f \) Byzantine processes. Consequently (correct) virtual processes can agree on the same set by the consensus algorithm that can tolerate at most \( f \) Byzantine processes. Thus correct agents can agree on the same set of starting information at \( v \).

4.4 Details

The pseudo-code of the algorithm is given in Algorithms 1, 2, and 3. Simply put, functions \( main() \) and \( DFS() \) realize

---

**Algorithm 1 main()**

1: — Variables in the whiteboard of node \( v \)—
2: var \( v.wb[ID_i].T \) and \( v.wb[ID_i].W \)
3: var \( v.wb[ID_i].round \)
4: var \( v.wb[ID_i].from_port \)
5: var \( v.wb[ID_i].unexplored_port \)
6: — Variables of agent \( a_i \)—
7: var \( a_i.node_num = 0 \) // count the number of nodes
8: var \( a_i.all_edge_num = 0 \) // count the number of edges
9: var \( a_i.r = 0 \) // keep the current round
10: var \( a_i.W = \emptyset \) // collect a set of starting information

11: consensus()
12: for \( a_i.r = 1 \) to \( f + 1 \) do
13: \( a_i.node_num = 1 \)
14: \( a_i.all_edge_num = 0 \)
15: DFS(null)
16: \( a_i.all_edge_num \times 2 \)
17: end for
18: Delete duplicated candidate from \( a_i.W \)
19: Move to a node where the minimum candidate in \( a_i.W \) is written
20: Declare termination

**Algorithm 2 DFS(f_port)**

1: make an inactive agent active if such an agent exists at \( v \)
2: if \( v.wb[ID_i].round \neq a_i.r \) then
3: \( v.wb[ID_i].round = a_i.r \)
4: \( v.wb[ID_i].from_port = f_port \)
5: if \( f_port = \) null then
6: \( v.wb[ID_i].unexplored_port = \{1, \ldots, d(o)\} \)
7: else
8: \( v.wb[ID_i].unexplored_port = \{1, \ldots, d(o)\} \setminus \{f_port\} \)
9: end if
10: \( a_i.node_num ++ \)
11: consensus()
12: if \( a_i.r = f + 1 \) then
13: for all candidate in \( v.wb[ID_i].W \) do
14: \( a_i.W = a_i.W \cup \{\text{candidate, } a_i.node_num\} \)
15: end for
16: end if
17: while \( v.wb[ID_i].unexplored_port \neq \emptyset \) do
18: \( x = \min(v.wb[ID_i].unexplored_port) \)
19: \( a_i.all_edge_num + + \)
20: \( v.wb[ID_i].unexplored_port = v.wb[ID_i].unexplored_port \setminus \{x\} \)
21: Go to the next node via port \( x \)
22: DFS(Port number via which \( a_i \) enters the current node)
23: end while
24: Backtrack via port \( v.wb[ID_i].from_port \). If it is null, do not move.
25: else
26: \( v.wb[ID_i].unexplored_port = v.wb[ID_i].unexplored_port \setminus \{f_port\} \)
27: Backtrack via port \( f_port \). If it is null, do not move.
28: end if
the DFS traversal of agent $a_i$. When $a_i$ starts the algorithm, $a_i$ executes $\textit{consensus()}$ once to simulate the 0-th phase of virtual process $v.p_i$. After that, for each node $v$, $a_i$ calls $\textit{consensus()}$ to simulate the $r$-th phase of $v.p_i$ when it visits $v$ for the first time during the $r$-th round.

4.4.1 Function $\textit{main()}$

The main function $\textit{main()}$ of this algorithm is shown in Algorithm 1. Each agent starts the algorithm at an arbitrary node $v$. Recall that, in the initial configuration, every agent is inactive, and some agent spontaneously becomes active and starts the algorithm. We denote by $v.wb[ID_i]$ the dedicated writable area of agent $a_i$ on the whiteboard of node $v$. We assume that, variables on the whiteboard of all agents of all nodes initialize by $null$. Throughout the algorithm, variable $a_i.r$ implies $a_i$ executes the $a_i.r$-th round.

When agent $a_i$ starts the algorithm, it executes function $\textit{consensus()}$ once and writes its starting information to the whiteboard of the current node $v$ (line 12). After that, $a_i$ executes the consensus algorithm to agree on the set of starting information. To do this, $a_i$ explores the network by the DFS and executes the consensus algorithm during the exploration (lines 13 to 18). We explain the details of the consensus algorithm later. After each exploration, $a_i$ waits for the same time as the exploration time to realize round synchronization (line 17).

After $a_i$ completes the $(f + 1)$-th round, it obtains a set of starting information such that all correct agents agree on the set. By using the information, $a_i$ computes the gathering node and moves there (lines 19 to 20). We explain the details of this behavior later.

4.4.2 Function $\textit{DFS()}$

The DFS function $\textit{DFS(f_port)}$ is shown in Algorithm 2. This function realizes exploration of all nodes by the DFS. Function $\textit{DFS(f_port)}$ is defined as a recursive function, and it is called every time an agent visits a node. The parameter $f.port$ denotes the port number through which an agent enters the current node. At the beginning of each exploration period, $\textit{DFS(null)}$ is called in line 16 of function $\textit{main()}. When $a_i$ visits node $v$, if an inactive agent $a_i$ exists, $a_i$ makes $a_i$ active. In this case, agent $a_i$ starts the algorithm before $a_i$ executes the algorithm at $v$. Thus, $a_i$ can read information written by $a_j$ at that time.

At first, $a_i$ checks $v.wb[ID_i].round$ to determine whether $a_i$ visits $v$ for the first time during the current round. If $v.wb[ID_i].round \neq a_i.r$ holds, $a_i$ visits $v$ for the first time. In this case, $a_i$ sets $v.wb[ID_i].round = a_i.r$ (line 3) and memorizes the incoming port by setting $v.wb[ID_i].from_port = f.port$ (line 4). It also initializes $v.wb[ID_i].unexplored_port$, which contains an unexplored port during the current round (lines 5 to 9). Next, $a_i$ increments $a_i.node_num$ (line 10). Note that, since $a_i$ visits nodes in the same order for every round, $a_i$ can use this value as a unique ID of $v$ for $a_i$. Actually $a_i$ later uses this value to recognize the gathering point. After that, agent $a_i$ executes $\textit{consensus()}$ to simulate the $a_i.r$-th phase of the consensus algorithm at node $v$. In lines 12 to 16, $a_i$ collects the results of the consensus algorithm in the $(f + 1)$-th round. After executing $\textit{consensus()}$, agent $a_i$ explores unexplored port in $v.wb[ID_i].unexplored_port$ (lines 17 to 23). Agent $a_i$ recursively calls $\textit{DFS(p)}$, where $p$ is the port number through which $a_i$ enters the next node. Agent $a_i$ also counts the number of edges by $a_i.all_edge_num$, which is used to calculate the duration of the waiting period. If no unexplored port exists, $a_i$ backtracks to the previous node via port $v.wb[ID_i].from_port$ (line 24). If $v.wb[ID_i].from_port = null$ holds, $a_i$ completes the exploration period of the current round. If $v.wb[ID_i].round = a_i.r$ holds, $a_i$ has already visited $v$. In this case, $a_i$ just updates $v.wb[ID_i].unexplored_port$ and backtracks to the previous node via port $f.port$ (lines 26 and 27).

4.4.3 Function $\textit{consensus()}$

Function $\textit{consensus()}$ in Algorithm 3 simulates the consensus algorithm in Sect. 3 by following the strategy in Sect. 4.3. In the 0-th round, $a_i$ simulates the 0-th phase of the consensus algorithm. That is, $a_i$ makes virtual process $v.p_i$ broadcast a signed value $\textit{Sign}_{v_i}(x_i)$ if $v.p_i$ is given an input value $x_i$. Recall that $v.p_i$ is given starting information of $a_i$ as an input if $a_i$ starts at $v$. This means the simulation of the 0-th phase is required only for the initial node of $a_i$. In other words, $a_i$ completes the 0-th round without exploring the network. Specifically, $a_i$ adds $\textit{Sign}_{v_i}(ID_i)$ to $v.wb[ID_i].T$ as its starting information, and adds $ID_i$ to $v.wb[ID_i].W$ (lines 1 to 3).

In the $r$-th round (lines 4 to 11), $a_i$ simulates the $r$-th phase of the consensus algorithm. To realize this, for every node $v$, $a_i$ simulates the $r$-th phase of $v.p_i$ when it visits $v$ for the first time during the round. Specifically, for every message received by $v.p_i$, $a_i$ checks its validity. Note that messages received by $v.p_i$ are stored in $\bigcup_{t \geq 0} v.wb[ID_i].T$. We say message $t = (x) : (id_1, v_1) : (id_2, v_2) : \cdots : (id_p, v_p)$ is valid if and only if $t$ satisfies all the following conditions, where we define $value(t) = x$ and $initial(t) = id_1$.

1. The number $y$ of signatures in $t$ is equal to $r$.
2. All signatures in $t$ are distinct.
3. Message $t$ does not contain $a_i$’s signature.
4. value(t) is not in $v.wb[ID_i].W$.
5. value(t) = initial(t) holds.
6. All the $y$ signatures are given at the current node.

Conditions 1–4 are identical to conditions in Sect. 3. Condition 5 is introduced to assure that value $ID_i$ in messages is originated from $a_i$. Note that, since correct agent $a_i$ can initially add $(ID_i): (ID_i, v) \in v.wb[ID_i].T$, every message $t$ forwarded by correct agents satisfies value(t) = initial(t). This implies condition 5 does not discard messages originated from and forwarded by correct agents, and consequently does not influence the simulation node. Condition 6 is introduced to assure that message $t$ is generated at the current node. If $t$ is valid, $a_i$ adds $Sign_{v,w}(t)$ to $v.wb[ID_i].T$ to simulate broadcast of $Sign_{v,w}(t)$ by virtual process $v.p_i$. At the same time, $a_i$ adds value(t) to $v.wb[ID_i].W$.

In the $(f + 1)$-th round, all agents complete simulating the consensus algorithm. That is, $v.wb[ID_i].W = v.wb[ID_i].W$ holds for any two correct agents $a_i$ and $a_j$. During the $(f + 1)$-th round, $a_i$ collects contents in $v.wb[ID_i].W$ for all $v$ by variable $a_i,W$ (lines 12 to 16 of DFS). Recall that $v.wb[ID_i].W$ includes IDs of agents that start at $v$. When $a_i$ memorizes $candidate \in v.wb[ID_i].W$, $a_i$ memorizes it as a pair $(candidate, a_i.node_num)$ to recognize the node later.

After that, $a_i$ computes the gathering node from the collected information in $a_i,W$ (lines 17 to 18 in main(t)). Since IDs of Byzantine agents may appear more than once in $a_i,W$, $a_i$ deletes all pairs from $a_i,W$ such that $candidate$ is duplicated. Then $a_i$ finds the pair such that $candidate$ is the smallest, and it selects the node of the pair as the gathering node.

Note that the pair includes $candidate$ and $a_i.node_num$. Hence $a_i$ can move to the gathering node by executing the DFS until $a_i.node_num$ becomes the same number as the pair (this procedure is omitted in main(t)).

5. Correctness of Our Algorithm

**Lemma 1:** For any two correct agents $a_i$ and $a_j$, before $a_i$ starts the exploration period of the $(r - 1)$-th round, $a_j$ completes the waiting period of the $(r - 1)$-th round.

**proof:** Immediately after correct agent $a_i$ becomes active, $a_i$ starts the DFS and explores all nodes. During the exploration, if $a_i$ encounters an inactive agent, $a_i$ makes it active. For this reason, all agents become active before $a_i$ completes the exploration period of the first round.

In addition, every correct agent requires exactly $2m$ unit times to complete the exploration period or the waiting period. Thus, every correct agent $a_j$ completes the exploration period of the first round before $a_i$ starts the exploration period of the second round. Similarly, for $r > 2$, every correct agent $a_j$ completes the exploration period of the $(r - 1)$-th round before $a_i$ starts the exploration period of the $(r - 1)$-th round.

**Lemma 2:** For any two correct agents $a_i$ and $a_j$, there exists exactly one node $v$ such that $ID_i \in v.wb[ID_j].W$ holds.

**proof:** First, we show that there exists at least one node $v$ such that $ID_i \in v.wb[ID_j].W$ holds. Let $v$ be the initial node of $a_i$. Then, in the 0-th round, $a_i$ adds its starting information $Sign_{v,w}(ID_i) \in v.wb[ID_j].T$ and adds $ID_i \in v.wb[ID_j].W$. Clearly, if $a_i = a_j$ holds, $ID_j \in v.wb[ID_j].W$ holds. If $a_i \neq a_j$ holds, when $a_j$ visits $v$ in the first round, $a_j$ reads $t = Sign_{v,w}(ID_i)$ from $v.wb[ID_j].T$. Then, since $t$ is valid, $a_j$ adds $ID_i \in v.wb[ID_j].W$. Therefore, $ID_i \in v.wb[ID_j].W$ holds.

Next, we show by contradiction that there exists at most one node $v$ such that $ID_i \in v.wb[ID_j].W$ holds. For contradiction, we assume that, for some distinct nodes $v_1$ and $v_2$, both $ID_i \in v_1.wb[ID_j].W$ and $ID_i \in v_2.wb[ID_j].W$ hold.

From the first part of the proof, when $v$ is the initial node of $a_i$, $ID_i \in v.wb[ID_j].W$ holds. Without loss of generality, we assume $v_1$ is the initial node of $a_i$. Clearly, $v_2$ is not the initial node of $a_i$. To satisfy $ID_i \in v_2.wb[ID_j].W$, $a_j$ must read some valid message $t = (ID_i) : (id_1, v_2) : (id_2, v_2) : \cdots : (id_r, v_2)$ from $v_2.wb[ID_j].T$ for some $a_i$ and add value(t) to $v_2.wb[ID_j].W$. Since message $t$ is valid, value(t) = initial(t) holds and thus $ID_i = id_1$ holds. However, since $v_2$ is not the initial node of $a_i$, $a_i$ never leaves $Sign_{v,w}(ID_i) = (ID_i) : (ID_i, v_2)$ in the whiteboard on $v_2$. This implies no agent can create message $t = (ID_i) : (ID_1, v_2) : (id_2, v_2) : \cdots : (id_r, v_2)$ on $v_2$. This is a contradiction. Therefore, the lemma holds.

**Lemma 3:** After all correct agents complete the $(f + 1)$-th round, for any node $v$ and any two correct agents $a_i$ and $a_j$, $v.wb[ID_i].W = v.wb[ID_j].W$ holds.

**proof:** In the $r$-th round, each correct agent $a_i$ simulates the $r$-th phase of the virtual process $v.p_i$ for every node $v$. That is, $v.p_i$ reads every message $t \in v.wb[ID_i].T$ sent by $v.p_i$ in the previous phase, and if the message is valid, $v.p_i$ sends message $Sign_{v,w}(t)$ (i.e., writes it to $v.wb[ID_i].W$) and adds value(t) to $v.wb[ID_i].W$. Note that we add conditions 5 and 6 to the validity conditions of the original consensus algorithm. Condition 6 assures that every valid message is generated in the consensus algorithm at the current node $v$. This implies every message generated at other nodes does not influence the consensus algorithm at $v$. Condition 5 does not influence the simulation at $v$ because every message originated from and forwarded by correct processes always satisfies condition 5. Condition 6 also does not influence the simulation at $v$ because every message generated at $v$ satisfies condition 6. This implies these two additional conditions do not influence the simulation at $v$. In addition, from Lemma 1, for any correct agent $a_i$, virtual process $v.p_i$ can receive every message sent in the previous phase. Consequently, virtual processes can correctly execute the consensus algorithm in Sect. 3. Therefore, from Theorem 1, we have the lemma.

**Lemma 4:** All correct agents obtain the same gathering node.

**proof:** We consider two correct agents $a_i$ and $a_j$. Each agent $a_i$ collects all starting information from every node during the $(f + 1)$-th round. That is, for every node $v$
and every candidate ∈ v.w[1][1].W, ai adds a 2-tuple (candidate, ai.node_num) to variable ai.W (ai.node_num is used to identify v). Agent aj also collects such information in aj.W. From Lemma 3, v.w[1][1].W = v.w[1][1].W holds for every node v. Therefore, a set of candidate contained in ai.W and aj.W is identical.

To obtain the gathering node, each agent ai deletes elements with duplicated candidate from ai.W, and then computes the node where the smallest candidate in ai.W is written. From Lemma 2, for correct agent ai, there exists an exactly one node v such that 1Dv ∈ v.w[1][1].W. Consequently, there exists at least one element in ai.W such that candidate is unique. Therefore, agent ai can computes the node with the smallest candidate as the gathering node. Since aj.W contains the same set of candidate as ai.W, agent aj can obtain the same node as the gathering node.

Theorem 2: Our algorithm solves the gathering problem in O(fm) unit times.

proof: From Lemma 4, all correct agents obtain the same gathering node. Therefore, all correct agents can meet at a single node and declare termination.

In this algorithm, each agent requires (f + 1) rounds to execute the consensus algorithm. Each round requires 2m unit times for the exploration period, and requires 2m unit times for the waiting period. After the (f + 1)-th round, each agent requires at most 2m unit times to go to the gathering node. Thus, our algorithm requires (2m+2m)×(f+1)+2m = O(fm) unit times.

6. Conclusion

In this paper, we proposed a Byzantine-tolerant gathering algorithm for mobile agents in synchronous networks with authenticated whiteboards. In our algorithm, each agent first writes its starting information to the initial node, and then each agent executes a consensus algorithm so that every correct agent agrees on the same set of starting information. Once correct agents obtain the set, they can calculate the same gathering node. By this algorithm, all correct agents can achieve gathering in O(fm) time, where f is the upper bound of the number of Byzantine agents and m is the number of edges. An important open problem is to develop a Byzantine-tolerant gathering algorithm in asynchronous networks with authenticated whiteboards. Since the consensus algorithm is proven to be unsolvable in asynchronous networks, we must consider other approaches.

References


Masashi Tsuchida received the M.S. degree in Nara Institute of Science and Technology in 2017. He is now Ph.D. course in Nara Institute of Science and Technology.

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 Information Science, 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. Currently, she is a Professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST), a Program Officer of Japan Society for the Promotion of Science (JSPS) Research Center for Science Systems and a member of Science Council of Japan. Her research interests include distributed algorithms, parallel algorithms, graph theory and design and test of digital systems. She is a senior member of IEEE and the Institute of Electronics, Information and Communication Engineers (IEICE), a member of the Information Processing Society of Japan (IPSJ) and Japanese Society for Artificial Intelligence.