Self-Stabilizing Weakly Byzantine Perpetual Gathering of Mobile Agents

Jion Hirose
Department of Electronics and Information Engineering, National Institute of Technology, Ishikawa College, Ishikawa, Japan

Ryota Eguchi
Graduate School of Science and Technology, Nara Institute of Science and Technology, Nara, Japan

Yuichi Sudo
Faculty of Computer and Information Sciences, Hosei University, Tokyo, Japan

Abstract

We study the *Byzantine* gathering problem involving k mobile agents with unique identifiers (IDs), f of which are Byzantine. These agents start the execution of a common algorithm from (possibly different) nodes in an n-node network, potentially starting at different times. Once started, the agents operate in synchronous rounds. We focus on *weakly* Byzantine environments, where Byzantine agents can behave arbitrarily but cannot falsify their IDs. The goal is for all *non-Byzantine* agents to eventually terminate at a single node simultaneously.

In this paper, we first prove two impossibility results: (1) for any number of non-Byzantine agents, no algorithm can solve this problem without global knowledge of the network size or the number of agents, and (2) no self-stabilizing algorithm exists if $k \leq 2f$ even with n, k, f, and the length Λ_g of the largest ID among IDs of non-Byzantine agents, where the self-stabilizing algorithm enables agents to gather starting from arbitrary (inconsistent) initial states. Next, based on these results, we introduce a *perpetual gathering* problem and propose a self-stabilizing algorithm for this problem. This problem requires that all non-Byzantine agents always be co-located from a certain time onwards. If the agents know Λ_g and upper bounds N, K, F on n, k, f, the proposed algorithm works in $O(K \cdot F \cdot \Lambda_g \cdot X(N))$ rounds, where $X(n)$ is the time required to visit all nodes in a n-nodes network. Our results indicate that while no algorithm can solve the original self-stabilizing gathering problem for any k and f even with *exact* global knowledge of the network size and the number of agents, the self-stabilizing perpetual gathering problem can always be solved with just upper bounds on this knowledge.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed algorithms, Byzantine environments, Gathering

Digital Object Identifier 10.4230/LIPIcs.SAND.2025.13

Related Version *Full Version*: `https://arxiv.org/abs/2504.08271` [10]

Funding *Jion Hirose*: JSPS KAKENHI Grant Number JP25K03101.
Ryota Eguchi: JSPS KAKENHI Grant Number JP22H03569.
Yuichi Sudo: JST FOREST Program JPMJFR226U and JSPS KAKENHI Grant Numbers JP20KK0232, JP25K03078, and JP25K03079.

1 Introduction

1.1 Background

Mobile agents, or simply agents, are software programs that can autonomously traverse a network, modeled as an undirected graph. We focus on the gathering problem which requires multiple agents, initially scattered throughout the network, to eventually terminate at a single node simultaneously. This problem is crucial for information exchange and enabling more complex behavior, and has been studied extensively [20].

Table 1 A summary of synchronous Byzantine or self-stabilizing gathering algorithms, assuming that agents have unique IDs. Here, n is the number of nodes, k is the total number of agents, K is a given upper bound of k, f is the number of Byzantine agents, F is a given upper bound of f, and Λ_g is the length of the largest ID among IDs of non-Byzantine agents. Symbols `REN`, `GAT`, `PGAT`, and GK represent rendezvous, gathering, perpetual gathering, and global knowledge, respectively.

Problem	`REN`	`GAT`			`PGAT`	
Byzantine	No	No	Yes		Yes	Yes
Self stabilizing	Yes	Yes	No	Yes	No	Yes
With GK	trivial	Possible [18]	Possible [8]	**Impossible for $k \leq 2f$** (Result 2)	trivial	**Possible** (Result 3)
	-	k	n or F	even with n, k, f, and Λ_g	-	N, K, F, Λ_g
Without GK	Possible [18]	Impossible [18]	**Impossible** (Result 1)	Impossible [18]	Possible [8]	-

In the literature, *transient* and *Byzantine* faults are considered in the gathering problem. The transient fault model involves temporary memory corruption, erroneous initialization, or other similar issues in faulty agents. To overcome transient faults, Ooshita, Datta, and Masuzawa [18] proposed a self-stabilizing algorithm that enables agents to gather from arbitrary (inconsistent) initial states, specifically addressing the gathering problem for two agents ($k = 2$), known as the *rendezvous* problem. They also claim that the algorithm can be extended to a self-stabilizing algorithm for cases where $k \geq 2$, if k is given to agents as global knowledge. In the Byzantine fault model, some agents incur Byzantine faults, which are called Byzantine agents, and they can behave arbitrarily against their algorithms; thus, this fault model includes other agent faults. In this paper, we assume that Byzantine agents cannot falsify their identifiers (IDs), which is a common assumption in many existing works. Byzantine agents under this restriction are sometimes referred to as *weakly* Byzantine agents in the literature, but we simply refer to them as Byzantine agents throughout this paper for simplicity. To tolerate Byzantine faults, many gathering algorithms [8, 11, 12] have been proposed. These algorithms require that agents initially know the number n of nodes or a given upper bound N on n. In particular, Dieudonné, Pelc, and Peleg [8] proposed an algorithm for the Byzantine environment if agents know n.

Both algorithms in [8] and [18] share a common characteristic: they require global knowledge of the network size or the number of agents to solve the problem. The authors in [18] show that without k, no self-stabilizing algorithm can solve the gathering problem. In the Byzantine fault model, it is unclear if the problem is solvable without n or F. Additionally, although these faults have received much attention so far, to the best of our knowledge, no studies consider both fault models *simultaneously* for agent systems.

1.2 Our Contribution

In this paper, we investigate the following questions: (1) is the gathering problem solvable in Byzantine environments *without* global knowledge of the network size or the number of (Byzantine) agents? (2) is there a *self-stabilizing* algorithm for the gathering problem? (3) if not, is there a relaxed variant of the gathering problem that allows the existence of the self-stabilizing algorithm?

For the first two questions, we demonstrate the following impossibility results for the gathering problem in Byzantine environments: (i) for any number of non-Byzantine agents, no algorithm can solve this problem *without* any global knowledge of the network size or the number of agents, and (ii) when $k \leq 2f$, no self-stabilizing algorithm exists for this problem, even if agents know the number n of nodes, the number k of agents, the number f of Byzantine agents, and the length Λ_g of the largest ID among IDs of non-Byzantine agents. Note that the impossibility proof for (i) is based on the scenario where only one non-Byzantine agent exists in the network, the agent should claim the termination of the algorithm. This assumption is common in fault-tolerant gathering problems (e.g., [8, 19]).

Based on these impossibilities, we propose a relaxed variant of the gathering problem, called *perpetual gathering* problem, in Byzantine environments. This problem requires that all non-Byzantine agents always be co-located from a certain time t onwards. Unlike the original gathering problem, this does not require the agents to stay at a single node permanently from some time; they may move but must do so together along the same edge after time t. The term "perpetual gathering" is inspired by the famous "perpetual exploration" problem, where an agent repeatedly visits all nodes in the graph (cf. [6]).

For the perpetual gathering problem, if the agents know Λ_g and upper bounds N, K, F on n, k, f, we propose a self-stabilizing algorithm for any k and f with time complexity of $O(K \cdot F \cdot \Lambda_g \cdot X(N))$, where $X(n)$ represents the time required to visit all nodes in a n-nodes network. For example, $X(n) = O(n)$ [13] for a cycle, a clique, and a tree. For an arbitrary graph, $X(n) = n^5 \log n$ [21]. When each node is equipped with a local memory (i.e., a whiteboard), $X(n) = O(m + nD)$, where m is the number of edges and D is the diameter of the graph [22].

Our results indicate that while no algorithm can solve the original self-stabilizing gathering problem for any k and f, even with *exact* global knowledge of the network size and the number of agents, the self-stabilizing perpetual gathering problem can always be solved with just upper bounds on this knowledge. Note that the (non-self-stabilizing) perpetual gathering problem without global knowledge in Byzantine environments can be solved using an algorithm proposed by Dieudonné et al. [8]. If the termination detection mechanism is removed from this algorithm, global knowledge becomes unnecessary. In this case, agents continue to move, but after a certain time, all non-Byzantine agents meet at the same node. This can be interpreted as solving the perpetual gathering problem without global knowledge. We summarize our contribution and existing results in Table 1.

1.3 Related Work

Many researchers considered rendezvous and gathering problems in various scenarios, such as synchrony, anonymity, network topology, randomness, and so on, and clarified the solvability. They also proposed various solutions to reduce various costs (e.g., time complexity, the number of moves, memory capacity, etc.). In this paper, we focus on deterministic solutions for rendezvous and gathering problems in the scenario where agents have unique IDs and move synchronously; Pelc [20] extensively surveyed these solutions, including those for anonymous agents and asynchronous environments. A comprehensive survey of the randomized solutions can also be found in the book by Alpern and Gal [1].

In such a scenario, the majority of the results are for fault-free settings. These results exploited agent IDs to break the symmetry. Several studies [7, 14, 23] proposed algorithms to solve rendezvous problems for arbitrary graphs if agents do not know the number of nodes n and start with a delay of at most τ rounds. Dessmark et al. [7] presented the first deterministic algorithm, with polynomial time in n, τ, and λ, where λ is the length of the

smallest ID among agent IDs. Kowalski and Malinowski [14] proposed an algorithm whose time complexity is independent of τ and is polynomial in n and λ. Ta-Shma and Zwick [23] provided an algorithm with lower time complexity that is polynomial in n and λ. Miller and Pelc [15] investigated tradeoffs between the number of rounds and the total number of edge traversals until the meeting.

Elouasbi and Pelc [9] introduced a new problem (called gathering with detection), which requires agents to declare the termination simultaneously when the meeting is perceived only through communication. They proposed an algorithm for this problem on arbitrary graphs in polynomial time in n and λ if agents use beeps as their communication method and do not know n. Bouchard et al. [4] considered a weaker model where each agent only detects the number of agents located at the same node in a round. They provided two types of algorithms to achieve the gathering with detection on arbitrary graphs in this model: one in polynomial time in N and λ when agents know the upper bound N of n, and the other in exponential time in n when agents do not know n. Molla et al. [17] proposed a faster algorithm to solve the gathering with detection on arbitrary graphs. Their algorithm is faster than the existing algorithms for even just gathering if agents do not know n and start simultaneously, with polynomial time in n.

Recently, the gathering under various types of agent faults has been studied, particularly the gathering problem in the presence of Byzantine agents. This problem assumes that the network topology is arbitrary, there are k agents, and f of them are Byzantine. Studies addressing these problems considered two types of Byzantine agents: strongly and weakly Byzantine. Strongly Byzantine agents can ignore the algorithm and behave arbitrarily, while weakly Byzantine agents can do the same except falsify their own IDs. Dieudonné et al. [8] designed the first gathering algorithms to tolerate Byzantine agents. They proposed algorithms to solve the gathering for weakly Byzantine agents in polynomial time in n and Λ_g, where Λ_g is the length of the largest ID among IDs of non-faulty agents, both if agents know n and $k \geq f + 1$ holds, and if agents know the upper bound F of f and $k \geq 2F + 2$ holds. These algorithms match the lower bounds on the number of non-faulty agents required. They also provided algorithms to solve the gathering for strongly Byzantine agents in exponential time in n and Λ_g, both if agents know n and F and $k \geq 3F + 1$ holds, and if agents know F and $k \geq 5F + 2$ holds. However, the lower bounds in these cases are $F + 1$ and $F + 2$, respectively, which means that the required numbers of non-faulty agents required by these algorithms do not match these lower bounds. Bouchard et al. [2] improved the upper bounds to $k \geq 2F + 1$ if agents know n and F, and to $k \geq 2F + 2$ if agents know F. Bouchard et al. [3] provided an algorithm that achieves the gathering in the presence of strongly Byzantine agents in polynomial time in N and λ if agents have global knowledge of $O(\log \log \log N)$ and $k \geq 5f^2 + 7f + 2$ holds. Hirose et al. [11] shown a gathering algorithm that tolerates weakly Byzantine agents if agents know N and $k \geq 4f^2 + 9f + 4$ holds, with time complexity smaller than [8] and polynomial in N, f, and Λ_a, where Λ_a is the length of the largest ID among agent IDs. Additionally, Hirose et al. [12] proposed a similar algorithm if agents know N and $k \geq 9f + 8$ holds, with time complexity greater than [11] but smaller than [8]. Tsuchida et al. [24] reduced the time complexity of the gathering algorithm to tolerate weakly Byzantine agents by assuming authenticated whiteboards, where each node has dedicated memory for each agent to leave information if agents know F and $k \geq F$ holds. Their algorithm works in polynomial time in F and m, where m is the number of edges. Tsuchida et al. [25] showed an algorithm to solve the gathering problem in asynchronous environments with weakly Byzantine agents by using authenticated whiteboards. Miller and Saha [16] considered the case where agents can obtain information in the subgraph induced by nodes within a radius of

the network from their current node. Their proposed algorithm solves the gathering problem in the presence of strongly Byzantine agents in polynomial time in k and n if $k \geq 2f + 1$ holds.

Several studies looked at other types of agent faults. Chalopin et al. [5] studied delay faults, where faulty agents remain at the current node for a finite number of rounds regardless of their plans to move. Pelc [19] considered the gathering problem with crash faults, where some agents suddenly become immobile at the node in some round, for systems where each agent moves at constant speed but their speeds differ. Ooshita et al. [18] proposed a self-stabilizing algorithm to achieve the rendezvous when agents start with arbitrary states of their memory. This self-stabilizing algorithm guarantees that agents eventually meet at a single node even if their memory experiences any kind of corruption, i.e., it counteracts transient faults in agent memory.

2 Preliminaries

2.1 Model

The system is represented as a simple, connected, and undirected graph $G = (V, E)$ with $n = |V|$ nodes. Nodes have no ID. Each node $v \in V$ assigns unique port numbers in $\{1, \ldots, d(v)\}$ to its edges, where $d(v)$ is the degree of v. Port numbers are local, meaning edges (v, u) and (u, v) for nodes v and u may have different numbers.

There are k agents in the system, denoted by set A. Each agent $a \in A$ has a unique ID $a.id$ but knows only its own ID. We denote "an agent with an ID in an ID set S_{id}" as "an agent in S_{id}" for brevity. Agents have unbounded memory but cannot leave any information on nodes or edges. Among the k agents, exactly f are Byzantine, which means they can act arbitrarily but cannot falsify their IDs; thus, when a Byzantine agent b communicates with another agent a, a can correctly identify b's ID. These agents are commonly referred to as *weakly Byzantine agents* and have been studied in the literature, including [8, 11, 12]. We call all the non-Byzantine agents *good* agents. We denote the length of the largest ID in IDs of good agents as Λ_g.

An algorithm $\mathcal{A}$ for an agent a is defined as $\mathcal{A}(a.id, n', k', f', \Lambda'_g) = (S, \delta, s_{ini}, S_{ter})$, where S is a set of states, δ is a function for transitioning its state, $s_{ini} \in S$ is the special state, and S_{ter} is a set of terminal states. Algorithm $\mathcal{A}$ takes five arguments, which are assigned integer values. The last four arguments correspond to n, k, f, and Λ_g, and each value is assumed to be assigned if required by the algorithm. We refer to the arguments n', k', f' and Λ'_g as *global knowledge*, and we assume that each of these arguments takes the same value for all agents in the system. In the above tuple, only the function δ depends on $a.id$, which means that the states of all agents are identical. States are represented by a tuple of variables of a; if its variables contain associative arrays, we refer to these simply as arrays. State s_{ini} represents one where all variables of a are initialized. A subset S_{ter} will be defined later.

A *configuration* in the system is defined as a combination of the locations, the states, and the IDs of all agents in A. We define C_{all} as the set of all possible configurations in which at least one good agent is not in a dormant state. An agent in a dormant state, or simply a dormant agent, transitions into s_{ini} when (a) the adversary wakes up the dormant agent, or (b) a non-dormant agent visits the node with the dormant agent. Non-dormant agents execute a common algorithm in synchronous and discrete time steps, called *rounds*. We let C_{ini} be a set of configurations in C_{all} where each agent is either dormant or in s_{ini}.

In each round r, each non-dormant agent a_i at v performs the following operations sequentially:

(1) It acquires $d(v)$, the port number p_{in} through which it arrived at v in $r-1$ (if it stayed in $r-1$, $p_{in} = \perp$), and states of agents (including a_i) at v. We define MG_i as the set of a_i and agents that a_i observes at v in r.
(2) It updates its next state s' using δ with $d(v)$, p_{in}, the states of all agents in MG_i, and the arguments of $\mathcal{A}$, deciding whether to stay or leave, and determining the outgoing port p_{out}. If it decides to stay, $p_{out} = \perp$.
(3) It either stays at v or moves to the destination node via p_{out} by $r+1$.

Note that in Operation (1), every agent in MG_i has to share its states with others before any agent in MG_i computes δ, but Byzantine agents in MG_i can falsify the values of their variables to others. We assume that all agents in MG_i observe the same state from any Byzantine agent in MG_i, i.e., Byzantine agents share the same (possibly falsified) states with all co-located agents. This assumption is known as *shouting* and has been adopted in previous studies on Byzantine gathering (e.g., [2, 3, 8, 11, 12]). Without this assumption, a Byzantine agent could send different states to different co-located agents. This makes it significantly harder for co-located good agents to detect inconsistencies or coordinate their actions. Note also that in Operation (3), if two agents pass the same edge in opposite directions simultaneously, they do not notice this fact. If an agent enters a state in S_{ter} at node v in round r, δ outputs $p_{out} = \perp$ and $s' \in S_{ter}$ in every round from r onwards. Since the terminal states depend on the global knowledge, we sometimes denote S_{ter} by $S_{ter}(n', k', f', \Lambda'_g)$ if needed. An execution starting from $c_0 \in C_{all}$ is a sequence $c_0, c_1, c_2, \ldots$, such that each configuration at round r is determined (deterministically) by applying the procedures described above to all the good agents and determining the behavior of the Byzantine agents by the adversary.

2.2 Problems

Gathering problem

This problem requires all good agents to eventually enter a state in S_{ter} at a single node simultaneously.

▶ **Definition 1.** *For a graph $G = (V, E)$ and $C^* \subseteq C_{all}$, an algorithm $\mathcal{A}_g$ solves the gathering problem in G from C^* if all good agents, executing $\mathcal{A}_g$ in G, eventually enter terminal states at the same node simultaneously, starting from any initial configuration of C^*. We say an algorithm $\mathcal{A}_g$ solves a gathering problem (`GAT`) if for any $G = (V, E)$ and C_{ini}, it solves the problem in G from any of C_{ini}. Similarly, we say an algorithm $\mathcal{A}_{ssg}$ solves a self-stabilizing gathering problem (`SS-GAT`) if for any $G = (V, E)$ and C_{all}, it solves the problem in G from any of C_{all}.*

Perpetual gathering problem

This problem requires all good agents to always be co-located from a certain round r onwards. Unlike the gathering problem, this problem does not require that these agents stay at a single node permanently; they may move but must do so together along the same edge after round r.

▶ **Definition 2.** *For a graph $G = (V, E)$ and $C^* \subseteq C_{all}$, an algorithm $\mathcal{A}_{pg}$ solves the perpetual gathering problem (`PGAT`) in G from C^* if all good agents, executing $\mathcal{A}_{pg}$ in G, always stay at a single node from a certain round onwards, starting from any initial configuration of C^*. We say an algorithm $\mathcal{A}_{sspg}$ solves a self-stabilizing gathering problem (`SS-PGAT`) if for any $G = (V, E)$ and C_{all}, the algorithm solves the perpetual gathering problem in G from any of C_{all}.*

Global knowledge and rounds

▶ **Definition 3.** *For a problem* $P \in \{$`GAT, PGAT, SS-GAT, SS-PGAT`$\}$*, we say an algorithm* $\mathcal{A}$ *solves* `P` *with global knowledge* n, k, f *and* Λ_g *if* $\mathcal{A}(id, n', k', f', \Lambda'_g)$ *solves* `P` *with the assumption* $(n', k', f', \Lambda'_g) = (n, k, f, \Lambda_g)$*. Similarly, we say* $\mathcal{A}$ *solves* `P` *with global knowledge* N, K, F *and* Λ_g *if* $\mathcal{A}(id, n', k', f', \Lambda'_g)$ *solves* `P` *with the assumption* $(n', k', f', \Lambda'_g) = (N, K, F, \Lambda_g)$ *for all* N, K, F *such that* $n \leq N, k \leq K, f \leq F$*. Additionally, we say an algorithm* $\mathcal{A}$ *solves* `P` *without the global knowledge, if* $\mathcal{A}(id, n', k', f', \Lambda'_g)$ *solves* `P` *for any values of* (n', k', f', Λ'_g)*.*

We evaluate the time complexity of each algorithm by the number of rounds required for any execution to reach the target configuration from the first configuration in which at least one good agent is not dormant.

2.3 Existing Algorithm

In the proposed algorithm, we incorporate a rendezvous algorithm as a subroutine to ensure that an agent meets other agents. This algorithm, due to Ta-Shma and Zwick [23], enables two agents to meet in any undirected and connected graph of n nodes, starting from different nodes. We refer to this algorithm as $\mathsf{REN}(label)$, where *label* is a positive integer given as input. When an agent a_i starts $\mathsf{REN}(label_i)$ for some positive integer $label_i$, it alternates between exploring and waiting periods based on $label_i$. The agent executes an exploring algorithm using a universal exploration sequence, due to Reingold[21], during the exploring period and waits at the node for the duration required by the exploring algorithm during the waiting period. The scheduling of these periods ensures that two agents with different positive integers meet, typically when one agent waits while the other explores. When two agents with distinct positive integers $label_i$ and $label_j$ execute this procedure, this time complexity of $\mathsf{REN}(label)$, denoted by $t_{\mathsf{REN}}(label)$, is $\tilde{O}(n^5 \cdot \lambda)$, where λ is the length of $\min(label_i, label_j)$, and $\tilde{O}$ hides a poly-logarithimic factor of n. We have the following theorem for this algorithm.

▶ **Theorem 4.** *(Theorems 2.3 and 3.2 of Ta-Shma and Zwick [23]) Let* a_i *and* a_j *be two agents, and* $label_i$ *and* $label_j$ *be positive integers such that* $label_i \neq label_j$ *holds. If* a_i *and* a_j *start* $\mathsf{REN}(label_i)$ *and* $\mathsf{REN}(label_j)$ *in rounds* r_i *and* r_j*, respectively, they meet at a single node by round* $\max(r_i, r_j) + t_{\mathsf{REN}}(\min(label_i, label_j))$*.*

We denote the t-th round of $\mathsf{REN}(id)$ by $\mathsf{REN}(id, t)$ for integer $t \geq 0$.

3 Impossibility results

In this section, we present the following impossibility results for the solvabilities of the `GAT` and the `SS-GAT`: there is no gathering algorithm without global knowledge, and there is no self-stabilizing gathering algorithm with n, k, f, and Λ_g.

3.1 No gathering algorithm without global knowledge

In this section, we show that no algorithm solves the `GAT` without global knowledge. To establish this result, we prove that without global knowledge, an agent cannot visit all nodes of a ring and declare the termination. While this proof is well-known, we provide a rigorous proof for the result since it is critical for demonstrating the impossibility of solving the `GAT`.

▶ **Lemma 5.** *There is no algorithm without global knowledge for a single agent to visit all nodes of a ring and declare the termination after visiting all nodes.*

Proof. We prove this lemma by contradiction. Suppose such an algorithm $\mathcal{A}_e$ exists. Let R_3 be a ring composed of 3 nodes, with each edge labeled with port numbers in the clockwise direction. When an agent a_i starts $\mathcal{A}_e$ from a node of R_3, it visits all nodes of R_3 and declares the termination within T rounds for some integer T. Consider a ring R_ℓ composed of $\ell > T$ nodes, where each edge is labeled with port numbers in the clockwise direction, similar to R_3. If a_i starts $\mathcal{A}_e$ from a node of R_ℓ, a_i declares the termination after T rounds, despite not having visited all nodes in R_ℓ, because a_i cannot distinguish R_3 and R_ℓ. This contradiction proves the lemma. ◀

Next, we use the above lemma to prove that no algorithm can solve the GAT without global knowledge.

▶ **Theorem 6.** *For any number $g \geq 1$ of good agents, no algorithm solves the GAT without any global knowledge.*

Proof. To derive a contradiction, we assume that such an algorithm exists $\mathcal{A}_g(id, n', k', f', \Lambda'_g)$. Since the algorithm works without any global knowledge, we denote it simply as $\mathcal{A}_g(id)$ in the following discussion. We analyze two cases on the number g of good agents.

Case 1 ($g = 1$). Suppose that there is only one good agent in the network. Then, by the definition of the GAT, this agent eventually transitions into a terminal state at some node, regardless of whether it meets Byzantine agents. Assume that there exists an ID id^* such that, for any graph $G = (V, E)$ and any initial position $v \in V$, when the good agent with ID id^* executes $\mathcal{A}_g(id^*)$, it always visits all nodes in V before terminating. Note that this must hold even if the Byzantine agent does not interact with the good agent. This implies that there could exist an exploring algorithm with a single agent by simulating $\mathcal{A}_g$ with id^* for G and v. This contradicts Lemma 5.

Case 2 ($g \geq 2$). Suppose that there are at least two good agents in the network. Assume that no ID is guaranteed to visit all nodes in the network before terminating. That is, for every ID id, there exists a graph $G = (V, E)$ such that when a good agent starts $\mathcal{A}_g$, the good agent enters a terminal state *before* visiting all nodes in V. We first consider the case $g = 2$. Let a_i and a_j be good agents with different IDs. By the assumption above, there exists a graph $G_i = (V_i, E_i)$ and a starting node $v_i \in V_i$ such that a_i, starting from v_i, terminates without visiting some node $u_i \in V_i$. Similarly, there exists a graph $G_j = (V_j, E_j)$ and node $v_j \in V_j$ such that a_j, starting from v_j, terminates without visiting some node $u_j \in V_j$. Consider a graph $G_{ij} = (V_{ij}, E_{ij})$ where $V_{ij} = V_i \cup V_j$ and $E_{ij} = E_i \cup E_j \cup \{(u_i, u_j)\}$. In G_{ij}, when a_i and a_j start $\mathcal{A}_g(a_i.id)$ and $\mathcal{A}_g(a_j.id)$ from v_i and v_j respectively, a_i never visits u_i and a_j never visits u_j. Since u_i and u_j are the only connection between V_i and V_j, two agents terminate at different nodes without ever meeting, which is a contradiction. For the case $g \geq 3$, the same contradiction can be derived by applying the same construction used in the case $g = 2$ to any pair of agents.

Hence, no such algorithm $\mathcal{A}_g$ can exist for g. ◀

Note that some algorithms, e.g., considered in [8, 19], have the basic property that they terminate even if there is only a good agent.

3.2 No self-stabilizing gathering algorithm with n, k, f and Λ_g

In this section, we prove that no algorithm can solve SS-GAT with n, k, f, and Λ_g.

▶ **Theorem 7.** *Let n be the number of nodes, k be the number of agents, f be the number of Byzantine agents, and Λ_g be the length of the largest ID among IDs of good agents. When $k \leq 2f$, no algorithm solves the SS-GAT even with the global knowledge n, k, f, and Λ_g.*

Proof. To derive the contradiction, we assume that there exists an algorithm $\mathcal{A}_{ssg}(id, n', k', f', \Lambda'_g)$ with $(n', k', f', \Lambda'_g) = (n, k, f, \Lambda_g)$ that solves the problem. We suppose that n and f are even and $k \leq 2f$ holds. Let I_α and I_β be disjoint sets of agent IDs such that $|I_\alpha| = |I_\beta| \leq f$ holds and each length of IDs in $I_\alpha \cup I_\beta$ is Λ_g. Such I_α and I_β should exist when $\Lambda_g \geq \lceil \log(2f) \rceil + 1$.

Consider a graph $G_1 = (V_1, E_1)$ with n nodes where the degree of each node is at most $n/2 - 1$. Suppose that all agents in I_α are good and all agents in I_β are Byzantine in G_1. When good agents execute $\mathcal{A}_{ssg}(id, n', k', f', \Lambda'_g)$ starting from $c \in C_{all}$, $|I_\alpha|$ good agents in I_α eventually meet and enter terminal states in $S_{ter}(n', k', f', \Lambda'_g)$ at a node v_1. Note that these executions include the situation that each good agent does not meet any Byzantine agent; therefore, we can assume that there are only $|I_\alpha| \leq f$ good agents in v_1 after these agents enter terminal states. Let such good agents in I_α be $a^1_0, a^1_1, \ldots, a^1_{|I_\alpha|-1}$.

Similarly, consider another graph $G_2 = (V_2, E_2)$ with n nodes where the degree of each node in G_2 is at most $n/2 - 1$. Suppose that all agents in I_α are Byzantine, and all agents in I_β are good in G_2. When good agents execute $\mathcal{A}_{ssg}(id, n', k', f', \Lambda'_g)$ starting from $c' \in C_{all}$, $|I_\beta|$ good agents in I_β eventually meet and enter terminal states in $S_{ter}(n', k', f', \Lambda'_g)$ at a node v_2. As with G_1, we assume that there are only $|I_\beta|$ good agents in v_2 after these agents enter terminal states. Let such good agents in I_β be $a^2_0, a^2_1, \ldots, a^2_{|I_\beta|-1}$.

Let $S_1 = (V^S_1, E^S_1)$ (resp. $S_2 = (V^S_2, E^S_2)$) be a star whose internal node, denoted by v'_1 (resp. v'_2), has the same degree as $d(v_1)$ (resp. $d(v_2)$), and $L_1 = (V^L_1, E^L_1)$ (resp. $L_2 = (V^L_2, E^L_2)$) be a line composed of $n/2 - d(v_1) - 1$ nodes (resp. $n/2 - d(v_2) - 1$ nodes). We consider a tree $T_3 = (V^S_1 \cup V^L_1 \cup V^S_2 \cup V^L_2, E^S_1 \cup E^L_1 \cup E^S_2 \cup E^L_2 \cup \{(u^S_1, u^S_2), (u^S_1, u^L_1), (u^S_2, u^L_2)\})$, where u^S_1 is a leaf in S_1, u^S_2 is a leaf in S_2, u^L_1 is a node in L_1, and u^L_2 is a node in L_2. We also consider a configuration c^* where v'_1 hosts $|I_\alpha|/2$ good agents and $|I_\alpha|/2$ Byzantine agents and v'_2 hosts $|I_\beta|/2$ good agents and $|I_\beta|/2$ Byzantine agents. Specifically, at v'_1, every agent $a^{1'}_i$ matches a^1_i in both ID ($a^{1'}_i.id = a^1_i.id$) and state ($s^{1'}_i = s^1_i$). Similarly, at v'_2, each agent $a^{2'}_i$ has an ID and state identical to its counterpart a^2_i ($a^{2'}_i.id = a^2_i.id$ and $s^{2'}_i = s^2_i$). Since T_3, G_1, and G_2 share the same values n, k, f, and Λ_g, we can observe that $S_{ter}(n', k', f', \Lambda'_g)$ of each agent is identical. Therefore, when good agents execute $\mathcal{A}_{ssg}(id, n', k', f', \Lambda'_g)$ starting from c^*, they thereafter remain stationary because their p_{out} and s' are $\perp$ and a state in $S_{ter}(n', k', f', \Lambda'_g)$ after the start, respectively. This implies that the good agents cannot meet at a single node, which is a contradiction. ◀

4 Self-stabilizing perpetual gathering algorithm with N, K, F and Λ_g

In this section, we provide an overview of our proposed algorithm for solving the SS-PGAT with N, K, F, and Λ_g. We then detail the behaviors of the proposed algorithm. Throughout the explanation and proof for the proposed algorithm, we assume that all good agents start an algorithm from $c_0 \in C_{all}$ simultaneously. We discuss how to remove this assumption in Section 5.

4.1 Overview

We present the underlying idea of the proposed algorithm. We first show two solutions for the PGAT: one in non-Byzantine environments, where no Byzantine agents exist, and another in Byzantine environments. The latter solution is based on an idea by Dieudonné et al. [8]. We then extend the latter solution to solve the SS-PGAT. We assume that agents start from any of C_{ini} when explaining the solutions for the PGAT.

To solve the PGAT in non-Byzantine environments, agents behave as follows: if an agent a_i is alone at the current node, it starts REN(*seed*), using $a_i.id$ as the input *seed* for the rendezvous algorithm. Whenever a_i meets other agents, a_i stops the execution of the current rendezvous algorithm and starts it using the smallest ID among IDs in MG_i as *seed*. According to Theorem 4, the number of agents traveling together grows, eventually ensuring that all good agents stay at the same node. However, this approach fails in a Byzantine environment. Consider a Byzantine agent b with an ID smaller than the smallest ID among the good agent IDs. Agent b repeatedly meets some good agents and leaves them. As a result, those good agents restart the rendezvous algorithm each time this action of b occurs. Those good agents cannot meet the remaining good agents because this repeated restarting prevents uninterrupted execution for a sufficiently long period. Moreover, it is also ineffective for agents to record the IDs of previously left agents and then exclude those recorded IDs when selecting *seed*. Agent b could leave some good agents and then travel with the remaining good agents. This causes the good agents to divide into two groups.

This problem can be resolved using the idea of Dieudonné et al. [8]. This idea extends the approach in non-Byzantine environments by modifying the determination of *seed*, as follows, to mitigate the influence of Byzantine agents. In this idea, a *group* is defined as a set of agents at a single node executing the rendezvous algorithm using the same *seed*. To ignore Byzantine agents, when the group members change, the agents in the group compute the largest subset S_b of agents at the current node such that every agent outside of S_b has observed each member of S_b leaving the group at some point in the past. Then, if the group meets agents neither in the group nor in S_b, or agents not in S_b leave the group, the group stops the execution of the current rendezvous algorithm and starts it using the smallest ID among IDs in MG_i and not in S_b as *seed*.

In this paper, for future extensions, we represent this idea using a labeled graph $G_{CG} = (V_{CG}, E_{CG})$, called *confidence graph*, where for an agent a_i, $|V_{CG}| = |MG_i|$, each node label corresponds one-to-one with the ID of an agent in MG_i, and each edge $(u, w) \in E_{CG}$ indicates that the two agents with the labels of nodes u and w trust each other. The detailed behaviors using this graph in each round are as follows: Agent a_i first checks whether each agent a_j at the current node is trustworthy. If a_i detects fraud by a_j, a_i does not trust a_j; otherwise, a_i trusts a_j. Agent a_i then simulates how a_j would evaluate trust relationships. Once a_i completes the above simulation for all agents in MG_i, it derives the trust relationships among those agents in MG_i. Next, a_i maps the trust relationships onto a confidence graph and calculates the IDs reachable from the node with $a_i.id$ in the graph to determine the group members. If the group members change from the previous round, a_i updates *seed*; otherwise, it continues using the same *seed*. This behavior ensures that good agents at the same node make the same confidence graph, as they base it on the states of agents at the same node at the beginning of the current round. Thus, good agents at the same node always select the same ID as *seed*. Consequently, after meeting, good agents travel together, and eventually, all good agents stay at the same node from a certain round.

While this approach solves the PGAT without global knowledge, it cannot be directly applied to the SS-PGAT, as agents start from a configuration of C_{all}, that is, each agent may start with a different state. In cases where some good agents already have a memory of traveling together with other good agents in the initial configuration, they may never trust those agents. As a result, the good agents remain divided into several groups.

To address this issue, the proposed algorithm restricts the length τ of the period that an agent suspects the other agents. In the proposed algorithm, each agent derives τ from global knowledge. All good agents stop suspecting the other good agents after the first τ

Algorithm 1 $\mathcal{A}_{sspg}(id, n', k', f', \Lambda'_g)$.

1: $a_i.numRound \leftarrow (a_i.numRound + 1) \bmod t_{\mathtt{REN}}(2^{\Lambda_g+1})$
2: Execute UpdateTrustRelationship()
3: Execute SelectSeed()
4: Execute REN$(a_i.seed, a_i.numRound)$

Table 2 Variables of agent a_i.

Variable	Explanation
numRound	The number of rounds since the beginning of the current REN.
seed	The ID used for the current REN.
$T_{wit}[id_1][id_2]$	Each element keeps an estimated round since an agent with id_1 witnessed the cheating of an agent with id_2, and it takes a value from 1 to τ.
R	A set of IDs of group members at the end of the previous round.

rounds. By appropriately setting τ, we will prove that a good agent meets another good agent within τ rounds, i.e., before a Byzantine agent suspected by the good agent can regain the trust of the good agent. Thus, the proposed algorithm ensures that all good agents stay at the same node within 2τ rounds.

4.2 Details

Algorithm 1 outlines the behavior for each round of Algorithm $\mathcal{A}_{sspg}(id, n', k', f', \Lambda'_g)$ and uses Algorithms 2 and 4 as sub-routines. Tables 2 and 3 summarize the variables used by an agent a_i in $\mathcal{A}_{sspg}(id, n', k', f', \Lambda'_g)$ and functions for arrays in $\mathcal{A}_{sspg}(id, n', k', f', \Lambda'_g)$, respectively. In $\mathcal{A}_{sspg}(id, n', k', f', \Lambda'_g)$, $i \in S_1$ for an index i and an array S_1, and $j \in S_2$ for an index j and a two-dimensional array S_2, indicates that i is included in the indices of S_1 and j is included in the indices of first dimension of S_2, respectively. Additionally, when an agent assigns a value c to $S[i]$ for an array S and a non-existent index i, we simply describe $S[i] \leftarrow c$. Each agent calculates $\tau = (6KF + 1) \cdot t_{\mathtt{REN}}(2^{\Lambda_g+1})$ based on global knowledge N, K, F, and Λ_g.

We focus on the process of each round by an agent a_i. First, a_i increments the value of $a_i.numRound$ by one. If the value assigned to $a_i.numRound$ exceeds $t_{\mathtt{REN}}(2^{\Lambda_g+1})$, a_i stores the value modulo $t_{\mathtt{REN}}(2^{\Lambda_g+1})$ in $a_i.numRound$. Next, using Algorithm 2, a_i calculates the trust relationship of all agents in MG_i. Then, using Algorithm 4, a_i selects an ID id_{seed} as the input of the rendezvous algorithm and reaches a consensus on *numRound* among agents in MG_i with the same id_{seed}. This prevents them from staying at different nodes in the next round. Finally, using $a_i.seed$ and $a_i.numRound$, a_i executes a rendezvous algorithm to meet other good agents.

4.2.1 Calculate a trust relationship

Algorithm 2 is the pseudo-code of Algorithm UpdateTrustRelationship. This algorithm aims to allow an agent a_i to calculate the trust relationship of all agents in MG_i. To do this, a_i uses a two-dimensional array $a_i.T_{wit}$. The indices in each dimension of $a_i.T_{wit}$ are comprised of agent IDs, and for two agents a_j and a_ℓ, the value of $a_i.T_{wit}[a_j.id][a_\ell.id]$ indicates the trust level as perceived by a_i. Specifically, if this value is between 1 and $\tau - 1$, it implies that a_i thinks a_j does not trust a_ℓ; otherwise (i.e., if this value is τ), it represents that a_i

Table 3 Functions used in $\mathcal{A}_{sspg}(id, n', k', f', \Lambda'_g)$.

Function	Explanation
OVERWRITE(S_1, S_2)	Overwrites an array S_1 (resp. a two-dimensional array S_1) with an array S_2 (resp. a two-dimensional array S_2).
REMOVE(S, i)	Removes an index i from indices of S if S is an array, and removes an index i from indices of the first-dimension of S if S is a two-dimensional array.

Algorithm 2 UpdateTrustRelationship.

```
1: OVERWRITE(TmpT_wit, a_i.T_wit)
2: for all a_j ∈ MG_i do
3:     TmpT_wit[a_j.id] ← UpdateValue(a_j.id)
4: end for
5: for all a_j ∈ {a_p | a_p.id ∈ a_i.T_wit ∧ a_p ∉ MG_i} do
6:     REMOVE(TmpT_wit, a_j.id)
7: end for
8: OVERWRITE(a_i.T_wit, TmpT_wit)
```

thinks a_j trusts a_ℓ. (Recall that the value is in the range $1, \ldots, \tau$.) To update $a_i.T_{wit}$, the algorithm employs a temporary variable $TmpT_{wit}$ to store the updated values. This ensures that $a_i.T_{wit}$ is not referenced before all elements are updated. Agent a_i finally replaces $a_i.T_{wit}$ with $TmpT_{wit}$ using OVERWRITE().

To calculate the trust relationship of all agents in MG_i, this algorithm first updates the trust relationship for a_i itself and then for the other agents. More specifically, a_i first fixes the first-dimension of $a_i.T_{wit}$ to a_i and executes UpdateValue$(a_i.id)$ to update elements of $a_i.T_{wit}[a_i.id]$. Algorithm UpdateValue$(a_j.id)$ returns an array with the updated values for the elements of $a_i.T_{wit}[a_j.id]$. Following this, a_i updates the elements of $a_i.T_{wit}[a_j.id]$ for an agent a_j in $MG_i \setminus \{a_i\}$ using UpdateValue$(a_j.id)$. Finally, a_i removes IDs of all agents not present at the current node from the first dimension of $a_i.T_{wit}$ using REMOVE().

Next, we explain the details of UpdateValue$(a_j.id)$ to update all elements in $a_i.T_{wit}[a_j.id]$. In this algorithm, a_i decides the updated value for an element $a_i.T_{wit}[a_j.id][a_\ell.id]$ for an agent a_ℓ as follows.

UV-Case (1) Assume that a_ℓ belongs to the same group as a_j at the end of the previous round. If a_j detects an anomaly for a_ℓ, the updated value is 1. Here, a_j detects an nomaly for a_ℓ if any of the following conditions hold: a_ℓ does not belong to MG_i; or a_j and a_ℓ have different values for any of the variables R, $numRound$, $seed$, or T_{wit}. These discrepancies indicate that a_ℓ may not have followed the expected protocol behavior in the previous round. If a_j does not detect an anomaly for a_ℓ and a_j does not trust a_ℓ, the updated value is $a_j.T_{wit}[a_j.id][a_\ell.id] + 1$.

UV-Case (2) If a_j meets a_ℓ for the first time, the updated value is τ.

UV-Case (3) If a_j has previously met a_ℓ, a_ℓ does not belong to the same group as a_j in the previous round, and a_j does not trust a_ℓ, then the updated value is $a_i.T_{wit}[a_j.id][a_\ell.id]+1$.

To verify the condition in UV-Case (1), a_i uses the results of detectAnomaly(a_j, a_ℓ). In UV-Case (2), where the indices of $a_i.T_{wit}[a_i.id]$ do not include $a_j.id$, a_i implicitly adds $a_j.id$ to the indices of $TmpT_{wit}[a_i.id]$.

In summary, indices in the first-dimension of $a_i.T_{wit}$ only include IDs of agents in MG_i. The updated value of each element $a_i.T_{wit}[a_j.id][a_\ell.id]$ depends on the locations and states of a_j and a_ℓ at the beginning of the round. This gives us the following observation.

Algorithm 3 UpdateValue($a_j.id$).

Function detectAnomaly$(a_j, a_\ell) = (a_\ell \notin MG_i \vee a_j.R \neq a_\ell.R \vee a_j.numRound \neq a_\ell.numRound \vee a_j.seed \neq a_\ell.seed \vee a_j.T_{wit} \neq a_\ell.T_{wit})$: This function returns whether agent a_j detects an anomaly for agent a_ℓ.

```
1: OVERWRITE(TmpArray, ∅)
   //UV-Case (1)
2: for all a_ℓ.id ∈ a_j.R do
3:     if detectAnomaly(a_j, a_ℓ) = True then
4:         TmpArray[a_ℓ.id] ← 1
5:     else if a_i.T_wit[a_j.id][a_ℓ.id] < τ then
6:         TmpArray[a_ℓ.id] ← a_j.T_wit[a_j.id][a_ℓ.id] + 1
7:     end if
8: end for
   //UV-Case (2)
9: for all a_ℓ ∈ {a_p | a_p ∈ MG_i \ a_j.R ∧ a_p.id ∉ a_j.T_wit[a_j.id]} do
10:    TmpArray[a_ℓ.id] ← τ
11: end for
   //UV-Case (3)
12: for all a_ℓ ∈ {a_p | a_p.id ∈ a_j.T_wit[a_j.id] ∧ a_p.id ∉ a_j.R ∧ a_j.T_wit[a_j.id][a_p.id] < τ} do
13:    TmpArray[a_ℓ.id] ← a_j.T_wit[a_j.id][a_ℓ.id] + 1
14: end for
15: Return TmpArray
```

▶ **Observation 8.** *Every pair of good agents at the same node has the same T_{wit}.*

4.2.2 Select a seed

Algorithm 4 is the pseudo-code of Algorithm SelectSeed. This algorithm serves two following objectives: (1) an agent a_i selects an ID id_{seed} for use in the current round of the rendezvous algorithm, and (2) agent a_i reaches a consensus on *numRound* with other agents in MG_i who share the same id_{seed}.

To achieve Objective (1), a_i first creates a confidence graph. Formally, a confidence graph is defined as follows.

▶ **Definition 9.** *Let* $\mathbf{CG}(MG_i, T_{wit})$ *be the undirected graph whose node set is the set of agents in* MG_i*, and there is an edge between two nodes* id_1 *and* id_2 *if and only if* $T_{wit}[id_1][id_2] = T_{wit}[id_2][id_1] = \tau$*. We call this graph the* confidence graph*.*

After creating the confidence graph, a_i identifies the IDs that are reachable from the node with $a_i.id$ in $\mathbf{CG}(MG_i, a_i.T_{wit})$ and selects these agents as group members. Agent a_i stores the IDs of these group members in a temporary variable *TmpR* to detect any changes in group membership since the end of the previous round. Finally, a_i selects the smallest ID among IDs of the group members and stores it in $a_i.seed$.

To achieve Objective (2), a_i compares the current group members with the agents in $a_j.R$ for each group member a_j. If there are discrepancies, a_i initializes $a_i.numRound$. Here, agents in $a_j.R$ means group members of a_j in the previous round. This comparison checks if each group member belongs to a different group than in the previous round. Thus, when agents in $a_j.R$ are different from them by incoherence, a_i can detect changes in members of the group with a_j. Therefore, this comparison ensures that all group members have a consistent *numRound* by the end of a round.

Algorithm 4 SelectSeed.

```
1: TmpR ← {a_j.id | a node with a_j.id is reachable from a node with a_i.id in CG(MG_i, a_i.T_wit)}
2: a_i.seed ← min(TmpR)
3: for all a_j ∈ TmpR do
4:     if TmpR ≠ a_j.R then
5:         a_i.numRound ← 0
6:     end if
7: end for
8: a_i.R ← TmpR
```

4.3 Correctness and Complexity

In this subsection, we prove the correctness and complexity of the algorithm proposed in Section 4. To do this, we first prove that two good agents at the same node make the same confidence graph. For simplicity of discussion, we define the time at $c_0 \in C_{all}$ as round 1.

Owing to space limitations, several proofs have been omitted. The complete proofs are available in the full version of this paper [10].

▶ **Lemma 10.** *For $r \geq 1$, in any round r, if two good agents exist at the same node, they create the same confidence graph in r.*

Proof. Let a_i and a_j be such agents. Given that both agents are the same node, it follows that $MG_i = MG_j$; therefore, V_{CG} of a_i and a_j are identical.

Without loss of generality, each edge in E_{CG} of a_i is determined by $a_i.T_{wit}$ and MG_i. According to Observation 8, $a_i.T_{wit} = a_j.T_{wit}$. Consequently, E_{CG} of a_i and a_j are also identical. ◀

Next, we prove that if two good agents belong to the same group at the end of a round, they do not detect an anomaly in each other in the next round.

▶ **Lemma 11.** *Consider any round r such that $r \geq 2$. If two different good agents a_i and a_j belong to the same group at the end of r,* detectAnomaly(a_i, a_j) = detectAnomaly(a_j, a_i) = *False holds in $r + 1$.*

Next, we prove that a good agent does not detect an anomaly in another good agent on or after round 2.

▶ **Lemma 12.** *Let a_i and a_j be two good agents. Agent a_i does not execute $a_i.T_{wit}[a_i.id][a_j.id] \leftarrow 1$ in any round on or after round 2.*

Proof. We prove this lemma by contradiction. Assume that a_i executes $a_i.T_{wit}[a_i.id][a_j.id] \leftarrow 1$ in a certain round after round 2. Let r be such a round. It holds that at the beginning of r, (1) $a_j \in a_i.R$ and (2) detectAnomaly(a_i, a_j) = *True*.

From (1), a_i and a_j exist at the same node in $r-1$. From Observation 8, $a_i.T_{wit} = a_j.T_{wit}$ holds. In addition, from (1), the node with label $a_j.id$ is reachable from the node with label $a_i.id$ in $\mathbf{CG}(MG_i, a_i.T_{wit})$ in $r-1$. This also holds true for $a_j.id$ since $\mathbf{CG}(MG_i, a_i.T_{wit}) = \mathbf{CG}(MG_j, a_j.T_{wit})$ holds in $r-1$ from Lemma 10. Thus, a_i and a_j belong to the same group at the end of $r-1$.

From Lemma 11, detectAnomaly(a_i, a_j) = *False* holds in r, which is inconsistent with (2). This contradiction proves the lemma. ◀

From Lemma 12, we have the following corollary.

▶ **Corollary 13.** *Let a_i be a good agent, and a_j be another good agent such that $a_j \in a_i.T_{wit}[a_i.id]$ holds. Let $r^j_{ini} \geq 1$ be the first round where $a_j \in a_i.T_{wit}[a_i.id]$ holds. By $r^j_{ini} + \tau$, $a_i.T_{wit}[a_i.id][a_j.id] = \tau$ holds.*

Next, we prove the number of times a good agent is interrupted while executing a rendezvous algorithm during τ rounds.

▶ **Lemma 14.** *Let a_i be a good agent. For any round r on or after round 2, a_i changes $a_i.seed$ and initializes $a_i.numRound$ at most $3kf$ times between r and $r + \tau$.*

Next, we prove that two good agents meet within τ rounds.

▶ **Lemma 15.** *For any round r on or after round 1, any pair of good agents meet between r and $r + \tau$.*

Finally, we prove the correctness and complexity of $\mathcal{A}_{sspg}(id, n', k', f', \Lambda'_g)$.

▶ **Theorem 16.** *Let N be the upper bound on the number of nodes, K be the upper bound on the total number of agents, F be the upper bound on the number of Byzantine agents, and Λ_g be the length of the largest ID among IDs of good agents. If N, K, F, and Λ_g are given to agents, Algorithm 1 solves the SS-PGAT with global knowledge in $O(K \cdot F \cdot t_{REN}(2^{\Lambda_g}))$ rounds.*

Proof. First, we prove that $a_i.T_{wit}[a_i.id][a_j.id] = \tau$ and $a_j.T_{wit}[a_j.id][a_i.id] = \tau$ hold from $\tau + 2$ onwards for any pair a_i, a_j of good agents. From Lemma 15, a_i and a_j meet between rounds 2 and $\tau+1$. Therefore, $a_i \in a_j.T_{wit}[a_j.id]$ and $a_j \in a_i.T_{wit}[a_i.id]$ hold at the beginning of $\tau + 2$. If a_i (resp. a_j) meets a_j (resp. a_i) for the first time, $a_i.T_{wit}[a_i.id][a_j.id] = \tau$ holds (resp. $a_j.T_{wit}[a_j.id][a_i.id] = \tau$ holds). Otherwise, $a_i \in a_j.T_{wit}[a_j.id]$ (resp. $a_j \in a_i.T_{wit}[a_i.id]$) has already held at the beginning of $\tau + 2$, and $a_i.T_{wit}[a_i.id][a_j.id] = \tau$ holds (resp. $a_j.T_{wit}[a_j.id][a_i.id] = \tau$ holds) by $\tau + 2$ from Corollary 13. From Lemma 12, after $a_i.T_{wit}[a_i.id][a_j.id] = \tau$ holds (resp. $a_j.T_{wit}[a_j.id][a_i.id] = \tau$ holds), a_i (resp. a_j) does not execute $a_i.T_{wit}[a_i.id][a_j.id] \leftarrow 1$ (resp. $a_j.T_{wit}[a_j.id][a_i.id] \leftarrow 1$). Hence, $a_i.T_{wit}[a_i.id][a_j.id] = \tau$ and $a_j.T_{wit}[a_j.id][a_i.id] = \tau$ hold from $\tau + 2$ onwards.

From Lemma 15 and the above discussion, a_i and a_j meet by $2\tau + 1$ and belong to the same group in this round. From Lemma 11, a_i and a_j stay at a single node from $2\tau + 1$ onwards. Given that τ is $(6KF + 1) \cdot t_{\mathtt{REN}}(2^{\Lambda_g+1})$, this theorem holds. ◀

5 Discussion

In Section 4, we present an algorithm to solve the SS-PGAT without global knowledge. Here, we discuss (1) removing the assumption of the simultaneous startup, and (2) the memory space required for each agent in this section.

Removing the assumption of simultaneous startup

For the execution without simultaneous startup, let the first round, denoted as round 1, be the round in which the first agent a is selected by the adversary and starts the algorithm. A similar discussion as Lemma 15 ensures that within at most $\tau + 1$ rounds, the other good agents meet a and start the algorithm. This fact holds since dormant agents stay at the node where they are located, and in the first τ rounds, there are at least $t_{\mathtt{REN}}(a.seed)$ rounds during which a does not change $a.seed$ and does not initialize $a.numRound$. This observation

ensures that the good agents start the algorithm within at most $\tau + 1$ rounds. After all good agents start the algorithm, the self-stabilizing property of the algorithm guarantees that the good agents successfully meet at a node and subsequently stay at a node within $2\tau + 1$ rounds. Therefore, without the assumption of the simultaneous startup, the algorithm solves the SS-PGAT within $3\tau + 1$ rounds, which is asymptotically the same as the algorithm with simultaneous startup.

Memory Space

To analyze the memory space, we consider the maximum size of each variable.

- For Variable *numRound*, since it stores at most $t_{\mathtt{REN}}(2^{\Lambda_g+1})$, its maximum size is given by $\log(\tilde{O}(N^5 \cdot \Lambda_g))$.
- For Variable *seed*, since it stores at most ID 2^{Λ_g+1}, its maximum size is given by $\Lambda_g + 1$.
- For Variable T_{wit}, the indices of its first dimension accommodate at most k IDs, while those of the second dimension store at most 2^{Λ_g+1} IDs. Every element stores at most $(6KF + 1) \cdot t_{\mathtt{REN}}(2^{\Lambda_g+1})$. Thus, its maximum size is $O(k \cdot 2^{\Lambda_g} \cdot \log(K \cdot F \cdot t_{\mathtt{REN}}(2^{\Lambda_g}))$.
- For Variable R, it stores at most k IDs, thus its maximum size is $O(k \cdot \Lambda_g)$.

In summation, the required memory space is $O(k \cdot 2^{\Lambda_g} \cdot \log(K \cdot F \cdot t_{\mathtt{REN}}(2^{\Lambda_g}))$, and this result is primarily determined the maximum size of Variable T_{wit}.

The memory space can be reduced by additional operations. For each index i of the first dimension of T_{wit}, an agent ensures that indices of $T_{wit}[i]$ comprise at most K IDs. More concretely, when an agent stores a new ID in the indices of $T_{wit}[i]$, it deletes the ID of the least seen agent from indices of $T_{wit}[i]$ if the number of indices of $T_{wit}[i]$ exceeds K. This operation allows agents to eliminate the IDs of non-existent agents introduced by incoherence, and thus the maximum size of T_{wit} becomes $O(k \cdot K \cdot \log(K \cdot F \cdot t_{\mathtt{REN}}(2^{\Lambda_g}))$, that is, becomes smaller. Thus, memory space can be lowered to $O(k \cdot K \cdot \log(K \cdot F \cdot t_{\mathtt{REN}}(2^{\Lambda_g}))$.

6 Conclusion

In this paper, we presented two impossibility results regarding the solvability of the GAT and SS-GAT, and proposed an algorithm to solve the SS-PGAT. We thoroughly analyzed the global knowledge required to solve the GAT and SS-GAT, examining each case with and without such knowledge. Based on the impossibility results, we introduced the PGAT as a relaxed variant of the GAT and investigated the global knowledge required for this problem as well.

Our findings show that no algorithm can solve the GAT without global knowledge, and no algorithm can solve the SS-GAT for any k and f with n, k, f, and Λ_g. On the other hand, for the SS-PGAT, we provided an algorithm that solves the problem with N, K, F, and Λ_g, where these values represent the upper bounds of n, k, and f, respectively. The time complexity of our proposed algorithm is $O(K \cdot F \cdot t_{\mathtt{REN}}(2^{\Lambda_g}))$.

Despite these results, the solvability of the SS-PGAT without global knowledge remains an open problem.

References

1 Steve Alpern and Shmuel Gal. *The Theory of Search Games and Rendezvous*, volume 55 of *International series in operations research and management science*. Kluwer, first edition, 2003.

2 Sébastien Bouchard, Yoann Dieudonné, and Bertrand Ducourthial. Byzantine gathering in networks. *Distributed Computing*, 29(6):435–457, 2016. doi:10.1007/s00446-016-0276-9.

3 Sébastien Bouchard, Yoann Dieudonné, and Anissa Lamani. Byzantine gathering in polynomial time. *Distributed Computing*, 35(3):235–263, 2022. doi:10.1007/s00446-022-00419-9.

4 Sébastien Bouchard, Yoann Dieudonné, and Andrzej Pelc. Want to gather? no need to chatter! *SIAM Journal on Computing*, 52(2):358–411, 2023. doi:10.1137/20m1362899.

5 Jérémie Chalopin, Yoann Dieudonné, Arnaud Labourel, and Andrzej Pelc. Rendezvous in networks in spite of delay faults. *Distributed Computing*, 29:187–205, 2016. doi:10.1007/s00446-015-0259-2.

6 Shantanu Das. Graph explorations with mobile agents. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 403–422. Springer International Publishing, 2019. doi:10.1007/978-3-030-11072-7_16.

7 Anders Dessmark, Pierre Fraigniaud, Dariusz R Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46:69–96, 2006. doi:10.1007/s00453-006-0074-2.

8 Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Transactions on Algorithms*, 11(1):1:1–1:28, 2014. doi:10.1145/2629656.

9 Samir Elouasbi and Andrzej Pelc. Deterministic rendezvous with detection using beeps. *International Journal of Foundations of Computer Science*, 28(1):77–97, 2017. doi:10.1142/S012905411750006X.

10 Jion Hirose, Ryota Eguchi, and Yuichi Sudo. Self-stabilizing weakly byzantine perpetual gathering of mobile agents. *CoRR*, abs/2504.08271, 2025. doi:10.48550/arXiv.2504.08271.

11 Jion Hirose, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue. Weakly Byzantine gathering with a strong team. *IEICE TRANSACTIONS on Information and Systems*, 105(3):541–555, 2022. doi:10.1587/transinf.2021fcp0011.

12 Jion Hirose, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue. Fast gathering despite a linear number of weakly byzantine agents. *Concurrency and Computation: Practice and Experience (CCPE)*, 36(14), 2024. doi:10.1002/cpe.8055.

13 Michal Koucký. Universal traversal sequences with backtracking. *Journal of Computer and System Sciences*, 65(4):717–726, 2002. doi:10.1016/S0022-0000(02)00023-5.

14 Dariusz R Kowalski and Adam Malinowski. How to meet in anonymous network. *Theoretical Computer Science*, 399(1-2):141–156, 2008. doi:10.1016/j.tcs.2008.02.010.

15 Avery Miller and Andrzej Pelc. Time versus cost tradeoffs for deterministic rendezvous in networks. *Distributed Computing*, 29:51–64, 2016. doi:10.1007/s00446-015-0253-8.

16 Avery Miller and Ullash Saha. Fast Byzantine gathering with visibility in graphs. In *16th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS 2020)*, pages 140–153. Springer, 2020. doi:10.1007/978-3-030-62401-9_10.

17 Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. Fast deterministic gathering with detection on arbitrary graphs: The power of many robots. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2023, St. Petersburg, FL, USA, May 15-19, 2023*, pages 47–57. IEEE, IEEE, 2023. doi:10.1109/IPDPS54959.2023.00015.

18 Fukuhito Ooshita, Ajoy K. Datta, and Toshimitsu Masuzawa. Self-stabilizing rendezvous of synchronous mobile agents in graphs. In *19th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2017)*, pages 18–32. Springer International Publishing, 2017. doi:10.1007/978-3-319-69084-1_2.

19 Andrzej Pelc. Deterministic gathering with crash faults. *Networks*, 72(2):182–199, 2018. doi:10.1002/net.21810.

20 Andrzej Pelc. Deterministic rendezvous algorithms. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 423–454. Springer International Publishing, 2019. doi:10.1007/978-3-030-11072-7_17.

21 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008. doi:10.1145/1391289.1391291.

22 Yuichi Sudo, Fukuhito Ooshita, and Sayaka Kamei. Brief announcement: Self-stabilizing graph exploration by a single agent. In *38th International Symposium on Distributed Computing, DISC 2024*, pages 55:1–55:7, 2024. doi:10.4230/LIPIcs.DISC.2024.55.

23 Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms*, 10(3):12:1–12:15, 2014. doi:10.1145/2601068.

24 Masashi Tsuchida, Fukuhito Ooshita, and Michiko Inoue. Byzantine-tolerant gathering of mobile agents in arbitrary networks with authenticated whiteboards. *IEICE TRANSACTIONS on Information and Systems*, 101(3):602–610, 2018. doi:10.1587/transinf.2017FCP0008.

25 Masashi Tsuchida, Fukuhito Ooshita, and Michiko Inoue. Byzantine-tolerant gathering of mobile agents in asynchronous arbitrary networks with authenticated whiteboards. *IEICE TRANSACTIONS on Information and Systems*, 103(7):1672–1682, 2020. doi:10.1587/transinf.2019EDP7311.