Deterministic Byzantine-Resilient Robot Gathering

by

Ullash Saha

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
August 2020

Thesis advisor

Dr. Avery Miller

Author

Ullash Saha

Deterministic Byzantine-Resilient Robot Gathering

Abstract

Gathering is a fundamental problem in the field of mobile robot algorithms. The goal is to gather the robots at a single point, which can be useful for sharing information or coordinating future actions. In this thesis, we consider models in which m robots are placed at nodes in a graph of size n, and the goal is to gather them at a single node of the graph. Here, each robot has an identifier (ID) and runs its own deterministic algorithm, i.e., there is no centralized coordinator. We consider a particularly challenging scenario: there are f maliciously faulty robots in the team that can behave arbitrarily, and even have the ability to change their IDs to any value at any time. There is no way to distinguish these robots from non-faulty robots, other than perhaps observing strange or unexpected behaviour. In this thesis, we seek to design a deterministic algorithm that is run by each robot to eventually have all the non-faulty robots located at the same node in the same round. It is known that no algorithm can solve this task unless there at least $f+1$ non-faulty robots in the team. In this thesis, we design an algorithm that runs in polynomial time with respect to n and m that matches this bound, i.e., it works in a team that has exactly $f+1$ non-faulty robots. In our model, we have equipped the robots with sensors that en-

able each robot to see the subgraph (including robots) within some distance H of its current node. We also prove that the gathering task is solvable if this visibility range H is at least the radius of the graph, and not solvable if H is any fixed constant.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my sincere appreciation and gratitude to my thesis supervisor Dr. Avery Miller, to give me the opportunity to do research under his supervision and provide me with invaluable guidance throughout this research. His helpful insights and immense knowledge on the subject improved the quality of the results and presentation of the work. I would like to extend my deepest gratitude to my thesis committee member, Dr. Stephane Durocher and Dr. Parimala Thulasiraman, to spend valuable time reading and evaluating this thesis. I am also grateful to the GADA lab members for their insightful suggestions, support, and warm humor, special thanks to Yeakub Hassan Rafi for his great company for the last two years. Thanks should also go to my friends and roommates for listening, offering me advice, and giving me a lot of amazing memories. I cannot begin to express my thanks to a special person, Joyasree Saha Jui, who has provided me with tremendous moral support through this entire process and writing this thesis. Finally, I would like to mention my parents: Ranjit Saha and Chandana Saha, my siblings: Rajib Saha and Uday Saha, and other family members - Rodela Saha and Rajya Saha, who are the ultimate source of my inspiration.

This thesis is dedicated to my beloved father, mother, and family for their continuous and unconditional love, prayers, and support throughout my life.

Chapter 1

Introduction

The study of algorithms for mobile robots has received significant attention in the field of distributed systems. Tasks that were previously executed by a single expensive robot can nowadays be completed in a distributed manner by deploying multiple cheap robots. Mobile robots play a vital role in real-life applications such as military surveillance, search-and-rescue, environmental monitoring, transportation, mining, infrastructure protection, and so on. In another potential application, mobile robots may represent autonomous vehicles that move on the streets in a city, and the configuration of the streets can be viewed as a network. In networks, robots move from one location to another and collectively complete a task. In order to complete the whole task as a team, the robots could meet each other multiple times to share their partial information or for any other purposes. In addition, robots are sometimes required to meet at a place after they finish the entire task concurrently. Therefore, gathering becomes a fundamental problem for mobile robots in networks.

Gathering is hard to accomplish even in a fault-free system, as the robots may not have any planned location where to meet, nor any initial information about the topology of the network. Moreover, in a distributed system, each robot runs its own deterministic algorithm to make decisions; there is no centralized coordinator. We want a deterministic algorithm that can be run by each robot, and eventually, they will gather at a single node which is not fixed in advance. However, if we consider identical moves by the robots running a deterministic algorithm, gathering is impossible for some network topologies. As an example, consider a group of robots placed in a ring network. If each robot executes the same moves to accomplish gathering, all the robots will move clockwise or counterclockwise in each time. As a result, the distance between the robots will remain the same all times, and they can not be gathered. Therefore, a correct algorithm needs to break the symmetric configuration of the network.

In this thesis, we also consider a particularly challenging scenario in which some of the robots are Byzantine: such robots do not follow our installed algorithm and can behave arbitrarily. We can think of these robots as malicious robots in our system, i.e., they have been compromised by outsiders/hackers, and, knowing the algorithm we intend to run, they can behave in ways that attempt to mislead the non-faulty robots into making incorrect decisions. Moreover, non-faulty robots do not know which of the robots (or even how many of the robots) are Byzantine, because all robots look identical. We face this type of scenario in real-world applications when attackers try to disrupt the normal behavior of systems. This is the worst possible case that we tend to deal with, so algorithms that are resilient to such attacks are

very useful.

The relative number of non-faulty robots versus Byzantine robots is an essential factor in solving this problem. If there are many Byzantine robots compared to the number of non-faulty robots, then the behaviour of the Byzantine robots can be very influential. As shown in previous work [23], a team that contains f Byzantine robots cannot solve gathering if the number of non-faulty robots is less than $f + 1$. The challenge, and the goal of our work, is to provide an efficient gathering algorithm that works when this bound is met, i.e., when the number of non-faulty robots is exactly $f + 1$. We provide such an algorithm in a model in which each robot is endowed with sensors that allow them to see all nodes and robots within a fixed distance H of its current location, where H is at least the radius of the network. We also prove an impossibility result which shows that no algorithm can solve gathering in this model if H is any fixed constant. It's important to note that this impossibility result does not contradict previous results [8, 9, 23, 40] that provide gathering algorithms with no visibility, as those algorithms make assumptions about additional information known to the robots (such as bounds on the network size, or on the number of Byzantine robots) or make assumptions about additional features such as authenticated whiteboards at the nodes. Depending on the application, our assumption that robots are equipped with sensors that gather information about their local environment could be more realistic than assumptions involving knowledge of the global environment, knowledge of the number of faulty robots, or the presence of a secure communication.

The new results that appear in this thesis have been accepted to ALGOSENSORS

2020 (The 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks).

Chapter 2

Preliminaries

2.1 The Model

A model is a system abstraction that specifies what can be observed, a list of operations we are allowed to perform and their effects. Two types of model are mostly considered in the literature for the gathering task: Mobile Robots in the Plane and Mobile Robots in Graphs.

2.1.1 Mobile Robots in the plane

In this model, robots move in a 2D-plane. Each robot cycles through three phases: Look, Compute and Move. In the "Look" phase, each robot makes an observation of the system. In the most popular versions, each robot gets a snapshot of the entire system, i.e., where all other robots are currently located. The set of positions of all robots at a given time is called the configuration of the system at that time. In the

"Compute" phase, each robot picks a point/location where it will move to, based on what it observed in the "Look" phase. It can pick its current location, meaning it does not move. In the "Move" phase, it moves towards the location it chose in the "Compute" phase. Depending on the model, these phases can be synchronous for all robots or can be asynchronous as well.

Robots have very limited capabilities under the most commonly studied version of this model. First of all, robots run the same algorithm and have no ID. This means that if two robots see the exact same thing during a "Look" phase, then they make the exact same decision during the "Compute" phase. Furthermore, a robot is memoryless in a sense that it cannot remember what it saw in earlier "Look" phases, and cannot remember decisions it made in previous "Compute" phases. It only makes a decision depending on what it sees in the current "Look" phase. More precisely, the decisions made by the robots in the "Compute" phase are deterministic methods of the observed environment only, and possibly of the robots ID, if available. In particular, no source of random information is available for making the decision. In addition, robots do not have any sense of chirality, meaning they can not distinguish between clockwise and counter-clockwise. They do not have any sense of ordinal direction as well (i.e., no compass). Another restriction of the robots is that they cannot detect multiple robots at the same point. The observed snapshot during a "Look" phase only indicates which points are occupied; there is no way of knowing how many robots are located at the same point. A significant amount of algorithms research has branched off from this model by modifying the above assumptions.

In some versions of the model, there can be faulty robots in the system. Mostly

two types of robot faults have been considered in the literature - Crash faults and Byzantine faults. A Crash fault implies that the faulty robot can stop its execution at any time of the executions, and does not perform any further computation or movement. On the other hand, a Byzantine fault is more dangerous : faulty robots may behave arbitrarily and maliciously.

2.1.2 Robots' Synchronism

The cycle of "Look", "Compute" and "Move" phases executed by the robots depends on the synchrony of the system. There are three types of synchrony that are usually considered.

- **Fully-synchronous Model (FSYNC):** In FSYNC, time proceeds in synchronized rounds where each round contains a Look-Compute-Move cycle. All robots participate in every round.
- **Semi-synchronous Model (SSYNC):** In SSYNC, time proceeds in synchronized rounds each containing a Look-Compute-Move cycle. However, not necessarily all robots participate in every round. There is a scheduler/adversary who decides which robots participate in which rounds.
- **Asynchronous Model (ASYNC) model:** In ASYNC, there are no rounds. Each robot's Look and Move phases are scheduled by a scheduler/adversary. However, the Compute phase is a local operation of the robot that is not controlled by the scheduler. The schedule events are $Look(i)$ and $Move(i)$, where i specifies a robot in the system. An adversary decides how far the robot gets

to move, but there is a guaranteed minimum distance δ. The main difficulty could be, for example - robot i might Look, then many Moves are performed by other robots, then i decides to move based on out-of-date information.

2.1.3 Mobile robots in Graphs

In this model, robots are placed at nodes in an undirected graph network. There can be different versions of this model. In this thesis, we consider a version which is similar to the model defined in [9]. We consider a team of m robots that are initially placed at arbitrary nodes of an undirected connected graph $G = (V, E)$. We denote by n the number of nodes in the graph, i.e., $n = |V|$. The nodes have no labels. At each node v, the incident edges are labeled with port numbers $0, \ldots, deg(v) - 1$ in an arbitrary way, where $deg(v)$ represents the degree of node v. These port numbers are used by the robots to specify their movements within the network. The two endpoints of an edge need not be labeled with the same port number.

For any two nodes v, w, the *distance between v and w*, denoted by $d(v, w)$, is defined as the length of a shortest path between v and w. The *eccentricity of a node v*, denoted by $ecc(v)$, is the maximum distance from v to any other node, i.e., $ecc(v) = \max_{w \in V}\{d(v, w)\}$. The *radius* of a graph, denoted by R, is defined as the minimum eccentricity taken over all nodes, i.e., $R = \min_{v \in V}\{ecc(v)\}$.

The team of m robots contains f Byzantine robots and $m - f$ non-faulty robots. Each robot α has a distinct identifier (ID) l_α, and it knows its own ID. The Byzantine and non-faulty robots look identical, i.e., there is no way to distinguish them other

than perhaps noticing strange or unexpected behaviour. All robots have unbounded memory, i.e., they can remember all information that they have previously gained during their algorithm's execution. We describe the differences between the two types of robots below.

- **Properties of non-faulty robots.** The non-faulty robots have no initial information about the size or topology of the graph, and they have no information about the number of Byzantine robots. A non-negative integer parameter H defines the *visibility range* of each robot, which we describe in Partial Snapshot below. Each non-faulty robot executes a synchronous deterministic algorithm: in each round, each robot performs one Look-Compute-Move sequence, i.e., it performs the following three operations in the presented order.

  1. **The Look operation:** A non-faulty robot α located at a node v at the start of round t gains information from two types of view.

     - *Local View:* Robot α can see the degree of node v and the port numbers of its incident edges. It can also see any other robots located at v at the start of round t, along with their ID numbers. Intuitively, this is what the robot sees at its current location.
     - *Partial Snapshot:* Robot α sees the subgraph consisting of all nodes, edges, and port numbers that belong to paths of length at most H that have v as one endpoint. Additionally, for each node w in this subgraph, robot α sees the list of all IDs of the robots occupying w at the start of round t. Intuitively, the robot obtains all information

about what's going on within a ball of radius H around itself.

2. **The Compute operation:** Using the information gained during all previous Look operations, a robot α located at a node v deterministically chooses a value from the set $\{null, 0, \ldots, deg(v) - 1\}$. In particular, it chooses *null* if it decides that it will stay at its current node v, and it chooses a value $p \in \{0, \ldots, deg(v) - 1\}$ if it decides that it will move to the neighbour of node v that is the other endpoint of the incident edge labeled with port number p.

3. **The Move operation:** A robot α located at a node v performs the action that it chose during the Compute operation. In particular, it does nothing if it chose value *null*, and otherwise, it moves to a neighbour w of v along the incident edge labeled with the chosen port number p. It can see the port number that it used to enter node w. There is no restriction of how robots move along an edge, i.e., multiple robots may traverse an edge simultaneously, in either direction.

All non-faulty robots wake up at the same time and perform their Look-Compute-Move sequences synchronously in every round (FSYNC).

- **Properties of the Byzantine robots.** We assume that a centralized adversary controls all of the Byzantine robots. This adversary has complete knowledge of the algorithm being executed by the non-faulty robots, and can see the entire network and the positions of all robots at all times. In each round, the adversary can make each Byzantine robot move to an arbitrary neighbouring

node. Further, we assume that the faulty robots are *strongly Byzantine*, which means that the adversary can change the ID of any Byzantine robot at any time (in contrast, a *weakly Byzantine* robot would have a fixed ID during the entire execution).

2.2 The Gathering Problem

In this thesis, we consider the gathering task for the mobile robots in graphs under the fully-synchronous scheduler. Assume that m robots are initially placed at nodes of a network, where f of the robots are strongly Byzantine. The robots synchronously execute a deterministic distributed algorithm. Eventually, all non-faulty robots must terminate their algorithm in the same round, and at termination, all non-faulty robots must be located at the same node.

2.3 Roadmap

In Chapter 3, we present the literature reviews related to robot-gathering problem in 2D spaces and graphs. In the literature, Bouchard et al. [9] provided a polynomial-time Gathering algorithm that works in the same model that we consider in this thesis, but in the case where the visibility radius H is equal to 0. We describe their proposed algorithm in Chapter 4. We also include the algorithms of Dieudonné et al. [23] that work for the model in which robots know the upper bound on the number

of Byzantine robots. In Chapter 5, we design and analyse our own algorithm that solves Gathering in any graph with n nodes containing m robots, f of which are strongly Byzantine, and where each non-faulty robot has visibility range H equal to the radius of the graph (or larger). In Chapter 6, we prove our three negative results; (i) the Gathering task is not solvable if the number of non-faulty robot is less than $f + 2$ in the presence of f weakly Byzantine robots and with robot's visibility range $H = 0$, even if the robots know the exact value of m and f, (ii) the Gathering task is not solvable if the number of non-faulty robot is less than $f + 1$ in the presence of f strongly Byzantine robots, (iii) the Gathering task is not solvable if the visibility range H is a fixed constant in the presence of $f \geq 0$ strongly Byzantine robots. Finally, we provide conclusion and outline some future directions of our thesis work in Chapter 7.

.

Chapter 3

Literature Review

The study of algorithms for mobile robots is extensive, as evidenced by a recent survey [30]. The Gathering problem has been investigated thoroughly under a wide variety of model assumptions, as summarized in [7, 20, 26] for continuous models (i.e., mobile robots in the plane) and in [14, 37] for discrete models (i.e., mobile robots in graphs).

3.1 Mobile Robots in the Plane

The Gathering problem has been studied primarily in continuous spaces [3, 6, 15, 16]. Cieliebak and Prencipe [16] first proposed a deterministic algorithm for gathering oblivious mobile robots (number of robots, $m = 3, 4$) in a two dimensional space. Their proposed algorithm even works in the presence of 5 or more robots (but for a restricted set of initial configurations). In their model, the authors endowed the

robots with multiplicity detection such that a robot can detect multiple robots on a single point. Later, Flocchini and Santoro [15] designed an algorithm that works for any initial configuration and under any asynchronous scheduler, but there should be at least 3 robots in the system to make their algorithm work. Balabonski et al. [3] introduced a new algorithm for the Gathering task under synchronous scheduler. In their work, they considered a weaker model in a sense that robots cannot detect multiple robots on a single point. However, none of the above mentioned algorithms [3, 15, 16] solves Gathering correctly when there are exactly 2 robots in the system.

The Gathering task in faulty systems has also been studied in the literature [1, 6, 17, 35]. Most of the algorithms [3, 15, 16] previously discussed in this section do not work properly in crash faulty system, and none of the algorithms can handle the Byzantine faults in view of [1]. Agmon and Peleg [1] first provided a deterministic algorithm for gathering $n \geq 3$ oblivious robots in the semi-synchronous settings where at most one robot is crash-faulty. The authors also proposed an algorithm to solve Gathering for Byzantine-tolerant mobile robots under fully-synchronous settings where the total number of faulty robots satisfies $f < \frac{1}{3}n$. Courtieu et al. [17] introduced a new Byzantine-tolerant algorithm that solves the Gathering in 2D space with a constraint that there cannot be exactly two points with same multiplicity in the initial configuration. Their algorithm works for semi-synchronous models, and robots are not bound to share the same chirality. Finally, Pattanayak et al. [35] proposed a wait-free (i.e., up to n-1 robots might suffer a crash fault) gathering algorithm for semi-synchronous models.

The Gathering task is generally named as Rendezvous when there are exactly

two robots in the system. Firstly, Cieliebak and Prencipe [16] proved that this is impossible to solve the Rendezvous problem under the model of Suzuki and Yamashita [2]. Then, Viglietta [41] solved the rendezvous problem by providing the robots an external memory register, called a light, that is visible by the other robot; values of this register are called colors. Their proposed algorithm uses only two colors for solving Rendezvous in semi-synchronous systems, but it uses three colors in non-rigid asynchronous systems (non-rigid means that robots can be stopped on the way to its destination during its Move phases). Later, Okumura et al. [33] solved the Rendezvous problem optimally (i.e., using 2 colors) in the LC atomic asynchronous setting (i.e., Look and following Compute occurs at the same time). In addition, they proposed an algorithm that executes Rendezvous under all asynchronous settings making an assumption that robot knows the minimum distance that it covers in each Move phase. Finally, Heriban et al. [31] designed an algorithm that executes the Rendezvous task using only 2 colors under all asynchronous systems without any additional assumption. We note that the algorithms discussed earlier in this section [3, 15, 16] solve the Gathering task properly for 3 or more robots without using any lights or external memory.

3.2 Mobile Robots in Graphs

Different types of task have been studied in the literature under discrete models, as examples - the Exclusive searching problem in [24], the Exploration problem in [27–29], and the Gathering problem in [18, 22]. D'Angelo et al. [24] first introduced

the Exclusive Searching problem in ring network. In this problem, robots are placed on anonymous distinct nodes of a bidirectional ring topology, and the aim is to make the robots clear all the edges of the ring : if an edge is traversed by any robot or two endpoints are visited by any robot, then the edge is considered to be cleared. Exclusive search means that no two robots will occupy the same node or edge while clearing the edges. Robot Exploration is another fundamental task that was investigated recently. Exploration requires that every node of the network is visited by at least one of the robots. Robot Exploration was studied in various types of networks such as trees [27], paths [28], rings [29] and general graphs [12]. A specific type of exploration problem is where a single robot must visit every node of the graph. This version was studied in [10, 38] and its solutions have been used as a core subroutine in the literature [8, 9, 23] for solving the Gathering problem.

Tasks for robots in networks have been investigated mostly under two versions of the model: (i) robots with strong visibility (robots can see the full graph) (ii) robots with local view only (robot can only see its own current node). With regards to partial visibility, the authors of [13] showed that equipping mobile robots with 1-hop visibility strictly increases their computational power as they can solve new instances of Graph Exploration. In [5], the authors show that Uniform Scattering is possible in grid graphs by a team of anonymous robots that move asynchronously, have partial visibility, have constant-sized memory, and are equipped with a compass. The authors of [4, 32] also study the Uniform Scattering problem with partial visibility and limited memory, but in connected subsets of grid graphs and when there is a restriction on where the agents can enter the environment. In [25], the authors consider the task of

having all robots converge to a 2x2 grid within any grid graph, where the robots are anonymous, synchronous, have partial visibility, and have no memory.

3.2.1 The Gathering Task in Graphs with Faulty Robots

The Gathering problem has been studied in networks under various fault scenarios. Previous work has focused on the case where robots have no visibility, i.e., they cannot see beyond their own current location. In [11], the authors study Gathering when faults can temporarily prevent a robot from moving in some rounds. The authors of [34] provide self-stabilizing gathering algorithms for two synchronous mobile robots, which means that synchronous two-robot gathering can be solved under any type of transient fault. In [36], Gathering was solved when robots moved asynchronously and were subject to crash faults.

Most relevant to our current work are the results about Gathering in networks when some of the robots can be Byzantine [8, 9, 23, 40]. In [40], the authors consider weakly Byzantine agents and add authenticated whiteboards to the model. Each node has a whiteboard where robots can leave messages: each robot has a dedicated space on each whiteboard that it can write to, and no other robots (even Byzantine ones) can write or erase a space that does not dedicated to them. All spaces on a whiteboard can be read by all robots. Additionally, each robot has the ability to write "signed" messages that authenticate the ID of the writer and whether the message was originally written at the current node. In this model, the authors provide an algorithm such that all correct robots gather at a single node in $O(f \cdot |E|)$ rounds, where f is the upper bound on the number of Byzantine robots and $|E|$ is the number

of edges in the network.

For the model we consider in our work (but with visibility range 0), the Gathering problem was first considered in [23]. The authors explored the gathering problem under four variants of the model: (i) known size of the graph, weakly Byzantine robots (ii) known size of the graph, strongly Byzantine robots (iii) unknown size of the graph, weakly Byzantine robots (iv) unknown size of the graph, strongly Byzantine robots. In all cases, the authors assume that the upper bound f on the number of Byzantine robots is known to all non-faulty robots. The authors provided a deterministic polynomial-time algorithms for the two models with weakly Byzantine robots. In the model when the size of the graph is known, their algorithm works for any number of non-faulty robots in the team. When the size of the graph is unknown, their algorithm works when the number of non-faulty robots in the team is $f + 2$. They prove a matching lower bound for the case of unknown graph size: no algorithm can solve Gathering if the number of non-faulty robots in the team is less than $f + 2$. For the model with strongly Byzantine robots and known graph size, the authors provided a randomized algorithm that guarantees that the agents gather in a finite number of rounds, and with high probability terminates in n^{cf} rounds for some constant $c > 0$. They also provide a deterministic algorithm whose running time is exponential in n and the largest ID belonging to a non-faulty agent. In both cases, the number of non-faulty robots in the team is assumed to be at least $2f + 1$. The authors also prove a lower bound for this model: no algorithm can solve Gathering if the number of non-faulty robots in the team is less than $f + 1$. Finally, for the model with strongly Byzantine robots and unknown graph size, they provide a deterministic algorithm

	Robots know the size of the graph		Robots do not know the size of the graph	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Weakly Byzantine	1	1 (Poly-time-Alg.)	$f+2$	$f+2$ (Poly-time-Alg.)
Strongly Byzantine	$f+1$	$2f+1$ (Exp-time-Alg.)	$f+2$	$4f+2$ (Exp-time-Alg.)

Table 3.1: Upper and lower bounds on the number of non-faulty robots needed to solve Gathering in the presence of f Byzantine robots (according to the result in [23])

that works when the number of non-faulty robots in the team is at least $4f+2$. The running time is exponential in n and the largest ID belonging to a non-faulty agent. They also prove a lower bound in this model: no algorithm can solve Gathering if the number of non-faulty robots in the team is less than $f+2$. Table 3.1 summarizes their results about Gathering in the presence of weakly and strongly Byzantine robots.

Subsequent work focused on the case of strongly Byzantine robots and attempted to close the gaps between the known upper and lower bounds on the number of non-faulty robots in the team. This was achieved in [8], as the authors provided algorithms that work when the number of non-faulty robots in the team are $f+1$ and $f+2$ for the cases of known and unknown graph size, respectively. However, the running times of these algorithms were also exponential in n and the largest ID belonging to a non-faulty agent. In this thesis, we have investigated that the lower bound of $f+2$ on the number of non-faulty robots for the model with weakly Byzantine and unknown size of the graph holds even if the robots know the exact number of non-faulty and faulty robots in the team. The proof is given in Section 6.

More recently, the authors of [9] considered a version of the above model that does not assume knowledge of the graph size nor the upper bound f on the number of strongly Byzantine agents. Instead, they consider the amount of initial knowledge as a resource to be quantitatively measured as part of an algorithm's analysis.

			known size of the graph		Unknown size of the graph	
Algorithms	Know the upper bound f?	Byzantine Type	Lower Bound	Upper Bound	Lower Bound	Upper Bound
Dieudonné et al. [23]	Yes	Weakly Byzantine	1	1 (Poly-time-Alg.)	$f+2$	$f+2$ (Poly-time-Alg.)
Dieudonné et al. [23]	Yes	Strongly Byzantine	$f+1$	$2f+1$ (Exp-time-Alg.)	$f+2$	$4f+2$ (Exp-time-Alg.)
Bouchard et al. [8]	Yes	Strongly Byzantine	$f+1$	$f+1$ (Exp-time-Alg.)	$f+2$	$f+2$ (Exp-time-Alg.)
Bouchard et al. [9]	No	Strongly Byzantine	$f+1$	$5f^2+6f+2$ (Poly-time-Alg.)	$f+2$	$5f^2+6f+2$ (Poly-time-Alg. with $O(\log\log\log n)$ bits of initial information)

Table 3.2: Results about Gathering in previous works

In this model, they designed an algorithm whose running time is polynomial in n and the number of bits in the smallest ID belonging to a non-faulty agent, where $O(\log\log\log n)$ bits of initial information is provided to all robots. The initial information they provide is the value of $\log\log n$, which the algorithm uses as a rough estimate of the graph size. Their algorithm works if the number of non-faulty robots in the team is at least $5f^2+6f+2$. They also provide a lower bound on the amount of initial knowledge: for any deterministic polynomial Gathering algorithm that works when the number of non-faulty robots in the team is at least $5f^2+6f+2$ and whose running time is polynomial in n and the number of bits in the smallest ID, the amount of initial information provided to all robots must be at least $\Omega(\log\log\log n)$ bits.

Table 3.2 summarizes the results of the algorithms from [8, 9, 23]. We note that the upper and lower bounds on the number of non-faulty robots match with poly-time algorithms, in weakly Byzantine case. In strongly Byzantine case, upper bound matches in exp-time algorithm, but upper bound is $5f^2+6f+2$ for poly-time algorithm. The poly-time algorithm also requires $O(\log\log\log n)$ bits advice in the case of strongly Byzantine and unknown size of the graph.

Chapter 4

The Known Algorithms

In this chapter, we provide more details about the most important previous work discussed in Chapter 3.2.1, i.e., the algorithms from [8, 9, 23] that solve Gathering in the presence of Byzantine robots (with visibility $H = 0$). In [23], the authors explored the gathering problem under four variants of the model: (i) known size of the graph, weakly Byzantine robots (ii) known size of the graph, strongly Byzantine robots (iii) unknown size of the graph, weakly Byzantine robots (iv) unknown size of the graph, strongly Byzantine robots. In all cases, the authors assume that the upper bound f on the number of Byzantine robots is known to all non-faulty robots. In Chapters 4.1 - 4.4, we summarize the high level idea of their algorithms for each of the variants. In [8], Bouchard provides improved bounds on the number of non-faulty robots in the case of strongly Byzantine robots, however they use a brute-force approach that requires exponential running time. In Chapter 4.5, we summarize the polynomial-time algorithm from [9] in the case of strongly Byzantine robots.

Preliminaries: The algorithms use universal exploration sequences [38] as a core subroutine which allows a non-faulty robot to visit every node of a graph, from an arbitrary unknown node, in polynomial time. We call this procedure $EXPLO(n)$ that works only for the model in which the robots know the size n of the graph. For the model with unknown size of the graph, the authors describe two different procedures based on [10, 19], which allow a non-faulty robot to visit every node in a graph of unknown size using a stationary or movable token (the procedure with stationary tokens is called EST, and the procedure with movable tokens is called EMT). The role of the tokens is played by the robots as well.

For the models with weakly Byzantine robots, the authors implement the idea of Ta-Shma and Zwick [39] that solves Rendezvous in non-faulty systems, i.e., it gathers two robots in a graph of known or unknown size. We denote by TZ(l) their procedure of deterministic Rendezvous [39], where l is the ID of the executing robot. The TZ procedure consists of two subroutines: $EXPLO(n)$ to explore the graph (robots run $EXPLO(n)$ for graphs of increasing sizes in the model with unknown size of the graph), and waiting periods whose duration depends on the ID of the executing robot. The procedure ensures that if two non-faulty robots with distinct IDs l_α and l_β respectively runs TZ(l_α) and TZ(l_β), then at some point, one of the robots stays idle at a node (during its waiting period), and the other robot finds it by running $EXPLO(n)$.

4.1 Algorithm Weak-Byz-Known-Size

The algorithm works for the model in which every robot (including the Byzantine robots) has a distinct ID, and the Byzantine robots cannot change their IDs throughout the entire execution. In this algorithm, every non-faulty robot maintains a blacklist to track down the Byzantine robots' IDs. But, we first discuss the initial ideas of the algorithm assuming that there is no Byzantine robot in the graph. The algorithm works as follows.

At the beginning, every robot starts executing deterministic Rendezvous in terms of its own ID (i.e., a robot starts executing TZ(l), where l is its ID). When a robot meets another robot, they both stop executing their previous procedure of Rendezvous, and start running TZ(l'), where l' is the smaller between the two IDs. The idea behind this procedure is to make the robots move together. The procedure goes on: when multiple robots are on the same node, they select the smallest ID in the group, and run Rendezvous accordingly. But this procedure does not ensure gathering in the presence of Byzantine robots: a Byzantine robot can hold the smallest ID in a group, then it can leave the group and join another group later (the ID of the Byzantine robot may become the smallest in the new group as well), both groups will continue running Rendezvous with respect to the same ID, and that does not guarantee Gathering.

The next idea is to run the Rendezvous in terms of the smallest remaining ID in the group. But a problem still remains there: a Byzantine robot with the smallest ID in the team may join a group, leave it, then join the group again, and so on.

Hence, the non-faulty robots need to keep track the robots that had leave the group once. Therefore, every non-faulty robot maintains a blacklist to track down the IDs of the Byzantine robots. The authors define the blacklist such a way that, the blacklist remains unique at a given time for a group of robots located on the same node. Hence, every non-faulty robot chooses the smallest non-blacklisted ID in the group to run the Rendezvous accordingly, that ensures the gathering in polynomial times (polynomial in n and the largest robot ID). This algorithm matches the lower bound of 1 on the required number of non-faulty robots.

4.2 Algorithm Weak-Byz-Unknown-Size

The algorithm's progress can be divided into three parts. The aim of the first part is to gather a group of robots such that there are at least $f+2$ non-blacklisted robots in the group. In order to do that, each robot continues to run the procedure Rendezvous in a similar way of Algorithm Weak-Byz-Known-Size (but for the graphs of increasing sizes), except the fact that there is no particular deadline for the termination since the size of the graph is unknown to the robots. More precisely, a robot continues executing the Algorithm Weak-Byz-Known-Size for the graphs of increasing sizes until it meets at least $f + 1$ other non-blacklisted robots in a group.

The second part of the algorithm gets to let the robots learn the size of the graph by running the procedure EMT (exploration with movable token). In order to run EMT, at least $f + 1$ non-blacklisted robots in each group play the role of the movable tokens, and the other robot plays the role of the explorer. This EMT repeats for

multiple times: every robot in a group plays the role of the explorer in each repetition one by one (the other robots play the role of tokens). After executing EMT for a significant amount of times, all robots learn the size of the graph. This execution is not possible with less than $f + 2$ robots in each group: if there are at most $f + 1$ robots in the group, then f robots will play the role of the tokens in which all the robots can be Byzantine, and that can cause the other non-faulty robot (explorer) to build a wrong map of the graph.

The third part is exactly same as Algorithm Weak-Byz-Known-Size since every robot knows the size of the graph now, and it completes Gathering in polynomial rounds with respect to n and the largest robot ID. This algorithm also matches the lower bound of $f + 2$ on the required number of non-faulty robots.

4.3 Algorithm Strong-Byz-Known-Size

This algorithm is considered for the model in which Byzantine robots can change their IDs at any round of the execution. Therefore, the authors could not able to reuse the concept of blacklisting in this algorithm. The algorithm Strong-Byz-Known-Size consists of two parts. The first part of the algorithm implements a brute force approach where the non-faulty robots try to build a group of at least $2f + 1$ robots by guessing all possible initial configurations of the graph. Robots test every possible structure of a n-node graph including its port assignments, possible labels of $2f + 1$ non-faulty robots, and their initial positions in the graph one by one. For each guessed initial configuration, robots (whose ID belongs to the current guessed configuration)

try to reach the target node (say the smallest robot ID's node) in the real graph. Eventually, the robots correctly guess the graph and configuration they are actually in, so they reach the target node, and build a group of at least $2f + 1$ robots. It should be noted that such a group sometimes can be created even for a wrong guess (multiple such groups can be created as well). Whenever a group of $2f + 1$ robots is formed, every robot in the group waits for a significant amount of time to let the other robots join the group. The idea behind forming a group of $2f + 1$ is to make sure that there are at least $f + 1$ non-faulty robots in the group (which is important for the second part of the algorithm).

In the second part of the algorithm, robots in each group simply run the procedure Rendezvous with respect to the unique identifier of their group (the unique identifier is calculated by $p_1^{l_1} \cdots p_m^{l_m}$ where l_i is the i^{th} id in the group, and p_i is the i^{th} prime number). As we know that, after the first part of the algorithm, there can be multiple groups of at least $2f + 1$ robots, each group runs Rendezvous in terms of its group identifier. The group size of $2f + 1$ ensures that the identifier will be unique in each group since there cannot be two groups of $2f + 1$ robots with the same set of IDs. However, a Byzantine robot can leave a group any time, but at least $f + 1$ robots always remains in every group. When two such groups meet, they merge and recompute the group identifier and continues to run Rendezvous in terms of the new identifier. The algorithm ensures Gathering in exponential time with respect to n and largest robot ID due to potentially guessing an exponential number of different graphs and configurations before succeeding, and the algorithm also requires at least $2f + 1$ non-faulty robots to accomplish the gathering.

4.4 Algorithm Strong-Byz-Unknown-Size

The algorithm's approach can be divided into three parts. The aim of the first part is to gather a group of at least $4f + 2$ robots. The procedure of creating such a group is similar to the first part of the Algorithm Strong-Byz-Known-Size, but since the size of the graph is unknown here, robots try every possible configuration of $4f+2$ robots for the graphs of increasing sizes.

The second part of the algorithm gets to let the robots learn the size of the graph by running the procedure EST (exploration with stationary token). In a group of at least $4f+2$ robots, $2f+1$ play the role of stationary tokens and the other robots play the role of explorers. The idea behind forming a group of $4f + 2$ robots in the first part of the algorithm was to make sure that there remains at least $f + 1$ non-faulty robots in each group of tokens and explorers, otherwise robots may build a wrong map. The EST procedure makes sure that every robot knows the size of the graph at the end of the second part.

The third part of the algorithm gets to merge the groups that were created in the previous parts of the algorithm. In each group, robots act as follows: (i) $2f + 1$ robots stay at their current position and play the role of 'messengers' to notify the other groups about their unique identifier, (ii) all other robots run $EXPLO(n)$ to explore the graph and find the tokens with the smallest identifier. At each node, explorers trust the identifier that is given by at least $f+1$ 'messengers' robots. Since there are at least $2f+1$ robots in each group of tokens, explorers always get the correct identifier of the groups. Explorers of each group then come back to their initial node

on which they started the second part, and notify the other robots (stationary tokens) about the group with the smallest identifier and its position. Every group then simply goes to the node on which the tokens with the smallest identifier are located, and accomplishes Gathering. The running time of this algorithm is exponential as well, and it requires at least $4f + 2$ non-faulty robots.

4.5 The Polynomial-Time Algorithm with Strong Byzantine Robots

This algorithm works for the model in which the non-faulty robots do not even know the upper bounds on the number of Byzantine robots. But the algorithm requires some initial global knowledge of size $O(\log \log \log n)$ to let the non-faulty robots learn the upper bounds on the size of the graph. Moreover, the algorithm requires at least $5f^2 + 6f + 2$ non-faulty robots to accomplish the Gathering (we can represent $5f^2 + 6f + 2$ as $(x-1)(f+1)+1$, where $x \geq 4f + 2$). First, we discuss two building blocks of this algorithm: GROUP and MERGE.

Procedure GROUP: The aim of this first procedure is to gather a group of at least $x - f$ non-faulty robots at a single node in some round. Robots complete this procedure in two phases, the first phase guarantees that eventually there is a round in which x robots meet at a node (even though the robots do not realize this event since they do not know the value of x or f). At the beginning, the non-faulty robots are divided into two groups - followers and searchers. The role of a follower is to stay idle at its current position, and the role of a searcher is to find the followers,

and marks their positions in its own map (a searcher builds a map and marks the position of each follower as a possible node on which x robots could meet). We can think of an ideal case: there is no Byzantine robots in the team, only one non-faulty robot acts as follower, and the other non-faulty robots play the role of the searchers, then the non-faulty robots (searchers) simply run $EXPLO(n)$, necessarily find the unique follower, and gather on that node. But in the presence of f Byzantine robots, there can be up to $f + 1$ followers in the graph (we assume exactly one non-faulty robot acts as follower). In this case, a searcher needs to run the first procedure in multiple steps. In the first step, every searcher runs $EXPLO(n)$ and goes to each node: it builds a map of its own while running $EXPLO(n)$, and marks the nodes on which it sees a follower. From the next steps, searcher simply goes to the closest node (on its map) on which it saw a follower with the smallest ID, and waits for a significant amount of time (if the searcher sees that the follower leaves the node at any time, it updates its map, and from the next step it goes to the node according to its updated map). Robots run the first procedure for a significant amount of steps such that there exists a step s in which no robot updates its map: in that step, there can be at most $f+1$ different nodes on which all the searchers can be located, and so there will be a node on which x robots meet (since the number of non-faulty robots is at least $(x-1)(f+1)+1)$). But everything we have discussed so far works under the assumption that there is a single non-faulty follower in the team. Now, if there are multiple non-faulty robots that play the role of followers, then each follower acts as follows: in each step, after waiting for a prescribed amount of time, if the follower sees no searcher on its node, then it plays the role of a searcher (explore the graph,

build a map, join the closest follower with smallest ID, if something wrong happens, update the map, and run the procedure again on the updated map). This overcomes the issue of multiple non-faulty followers, and guarantees the gathering of x robots on a node.

Procedure MERGE: This subroutine gets to merge the groups that were created in the previous subroutine. The subroutine uses an election procedure to elect a particular node on which all the groups will be gathered at the end. First, the robots in each group split themselves into two subgroups: half of the robots play the role of supporters and half of the robots play the role of voters. The supporters act as follows: each supporter remains idle at its current node and promotes the list of distinct IDs of the robots that were in its group at the beginning of this subroutine. The voters act as follows: each voter runs $EXPLO(n)$, visits each node, notes down the list of IDs it gets from the supporters at each node. and elects the node v_{max} on which it gets the lexicographically largest list of IDs. The list from the non-faulty supporters at node v_{max} contains at least $4f+2$ robots, since there exists a node on which at least $4f+2$ number non-faulty robots gathered in the first subroutine. A voter assumes that, at each node, half of the supporters gives him the correct information, and the other half gives the wrong information. Thus at least $\frac{4f+2}{2} \geq 2(f+1)$ supporters stay at v_{max} in the MERGE operation, and when the voters from the other groups visit the node, they accept the information of at least $\frac{2(f+1)}{2} = (f+1)$ robots as the correct information. Now, if the Byzantine robots try to fool some voters to elect a different node w, then there should be at least $f+1$ supporters at w to promote the exact list of IDs as the $f+1$ non-faulty supporters did in v_{max}, which is not possible since there

can be at most f Byzantine robots at w to promote the wrong list. Hence, every non-faulty voters can identify the unique v_{max}. The whole procedure will be repeated by changing the roles of the robots: the robots that played the role of supporter will play the role of voters now, and vice versa. Thereby, every non-faulty robot can identify the unique v_{max}, and they gather at v_{max}.

The Algorithm: we will now describe the full algorithm that consists of the GROUP and MERGE subroutines. The algorithm works as follows. A non-faulty robot α first finds the size of the graph $n = 2^{(2^{GK})}$, where GK is the given global knowledge of value $\log\log n$. Now, in order to execute the GROUP subroutine successfully, there should be at least one follower and one searcher in the team (if all the robots are followers or all the robots are searchers, then the GROUP subroutine does not work properly). Hence, a non-faulty robot first transforms its label by applying the label transformation derived in [21], and runs the GROUP and MERGE subroutines for multiple times in terms of the binary representation of its transformed ID. For each bit, α acts as follows: if the current bit is 0, α plays the role of a supporter (runs GROUP and MERGE), and if the current bit is 1, it plays the role of a voter (runs GROUP and MERGE). Since the transformed IDs of the robots are unique by at least 1 bit, eventually there will be a phase in which there will be at least 1 follower, and 1 searcher in the team, the robots will execute the GROUP and MERGE successfully, and will accomplish gathering. The robots also use two subroutines called LEARN and CHECK-GATHERING after executing each GROUP and MERGE operation to make sure that every robot knows that whether the gathering has been completed or not.

Chapter 5

Our Algorithm

In this section, we provide the details and analysis for our polynomial-time algorithm that solves Gathering in the presence of f strongly Byzantine robots when there are least $f + 1$ non-faulty robots in the team. Recall that, in our model, we assume that each robot is equipped with sensors that give it some visibility range H, but they have no information about the number of Byzantine robots.

5.1 Definitions and Terminology

The following graph-theoretic definitions will be used throughout the description and analysis of our algorithm. Recall that the *eccentricity of a node* v, denoted by $ecc(v)$, is the maximum distance from v to any other node, i.e., $ecc(v) = \max_{w \in V}\{d(v, w)\}$. The *radius* of a graph, denoted by R, is defined as the minimum eccentricity taken over all nodes, i.e., $R = \min_{v \in V}\{ecc(v)\}$. For any graph G, the

center of graph G is the set of all nodes that have minimum eccentricity, i.e., all nodes $v \in V(G)$ such that $ecc(v) = R$, and the *center graph of a graph* G, denoted by $C(G)$, is defined as the subgraph induced by the center nodes.

The following terminology will be used to refer to what a non-faulty robot α can observe in the Look operation of any round t during the execution of an algorithm.

- The *local view at a node* v *for round* t is denoted by $Lview(v, t)$, and refers to all of the following information: the degree of v, the port numbers of its incident edges, and a list of the IDs of all other robots located at node v at the start of round t.

- The *snapshot at a node* v *for round* t is denoted by $Sview(v, t)$, and refers to all of the following information: the subgraph consisting of all nodes, edges, and port numbers that belong to paths of length at most H that have v as one endpoint, and, for each node w in this subgraph, the list of IDs of all robots occupying w at the start of round t.

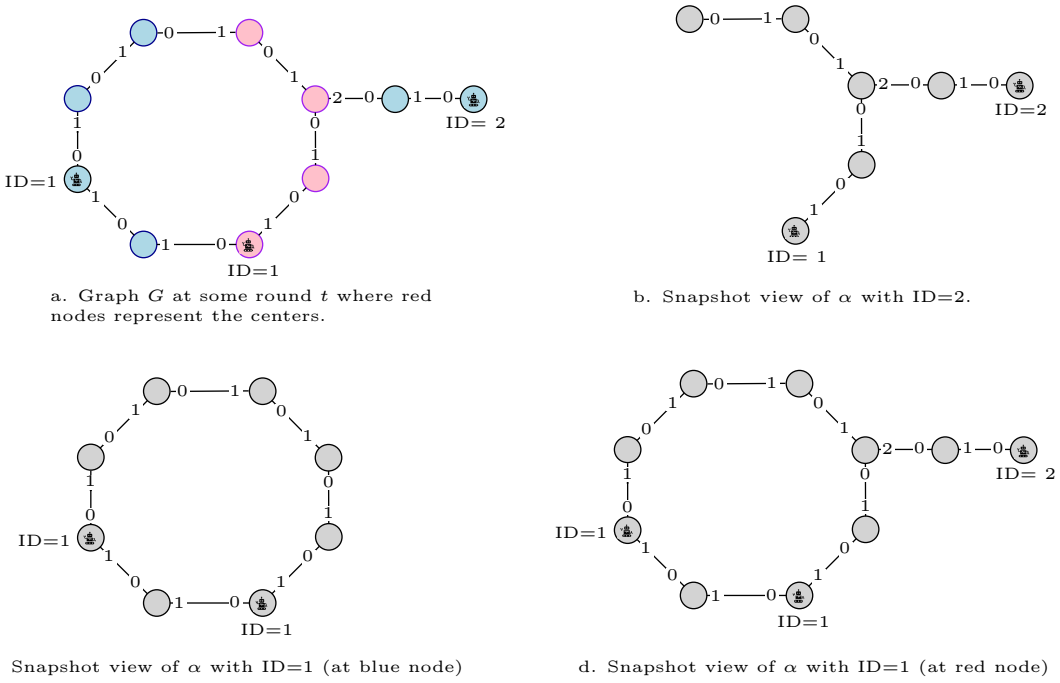
- For any graph G and round t, an ID l is called a *singleton ID* at round t if the total number of times that l appears as a robot ID at the nodes of G is exactly 1 at round t.

- For any two nodes u and v, a shortest path π refers to a path (a sequence of port numbers) that minimizes the distance $d(u, v)$. However, there can be multiple smallest paths between two nodes in a graph; in this situation, a robot chooses the port sequence which is lexicographically smallest.

In Figure 5.1, we show an example of α's snapshot view in an arbitrary graph G at some round t. Figure 5.1a shows the current configuration of the entire graph G at round t. We suppose that the visibility range of each robot is exactly 4 (which is equal to radius of the graph). In Figure 5.1a, there are two robots with $ID = 1$ (so at least one of the robots is Byzantine). If α (with $ID = 1$) is located at the specified blue node, then it sees the snapshot view as shown in Figure 5.1c. Otherwise, if it is located at the red node, it sees the entire graph (i.e., Figure 5.1d). There is another robot ID (ID=2) in Figure 5.1, which will be called a singleton ID at round t, since there is exactly 1 robot with ID=2 in G at round t. The non-faulty robot α (with ID=2) sees the snapshot view as shown in Figure 5.1b. It should be noted that a robot that sees its own ID several times in its snapshot view at different locations may not know which one corresponds to itself (see Figure 5.1c). Also, a robot cannot determine if its snapshot contains the whole graph, so even if it sees exactly one robot with ID k, it might be a Byzantine robot, and the non-faulty robot with ID k might be located somewhere beyond its visibility range (Figure 5.1b).

5.2 H-View-Algorithm

Let α be an arbitrary non-faulty robot with ID l_α in the graph G. In what follows, we assume that the visibility range of a non-faulty robot is at least equal to the radius of the graph, i.e., $H \geq R$. We also assume that the number of non-faulty robots is at least $f + 1$.



a. Graph G at some round t where red nodes represent the centers.

b. Snapshot view of α with ID=2.

c. Snapshot view of α with ID=1 (at blue node)

d. Snapshot view of α with ID=1 (at red node)

Figure 5.1: Snapshot views with visibility range $H = 4$

5.2.1 High Level Description

The algorithm's progress can be divided into three parts. The first part gets each non-faulty robot to move to a node v_{max} such that the robot's snapshot view from v_{max} contains all the nodes of the network G. This is the purpose of our Find-Lookout subroutine. Each robot α produces a list of potential nodes in its initial snapshot view where it thinks it might be located, and it does this by comparing its local view with the degree and robot list of each node in its initial snapshot. It cannot be sure of its initial position within its snapshot view since Byzantine robots can forge α's ID and position themselves at other nodes that have the same degree as α's current node. From each guessed initial position, α computes a port sequence of a depth-first traversal of its snapshot view and tries following it in the real network. Since

one of the guessed initial positions must be correct, at least one of the depth-first traversals will successfully visit all nodes contained in its initial snapshot view. Since the visibility range is at least the radius of the network, the robot's initial snapshot view must contain a node in the center of the network G, so at least one step of at least one of the traversals will visit a node in the center of G. When located at such a node, the robot will see all nodes in the network. So, by observing how many nodes it sees at every traversal step, and keeping track of where it saw the maximum, it can correctly remember and eventually go back to a node v_{max} from which it saw all nodes in the network.

The second part of the algorithm ensures that, eventually, there is a robot with a singleton ID that is located in the center of the network G. This is the purpose of our March-to-Center subroutine, which depends highly on the fact that each robot starts this part of the algorithm at a node v_{max} from which it can see every node in the network. If a robot starts March-to-Center knowing where in its snapshot view it is located, then the robot moves directly to the center of the G: it computes the center of its snapshot, and moves to one of the nodes in the center of this snapshot, which is also the center of the entire network G. If all robots do this, then the center of the network will contain a singleton ID, since there are more non-faulty robots than Byzantine robots, and all non-faulty robots have distinct ID's. The difficult case is, once again, when a robot α is not sure where in its snapshot it is located at the start of March-to-Center. This is because the Byzantine robots can forge α's ID and position themselves at other nodes with the same degree as α's current node. In this case, α will not move during March-to-Center, and simply watch to

see if it can spot any inconsistencies between its local view and its possible starting locations in its snapshot. The key observation, which we will prove, is that at least one of the following must happen in each execution of March-to-Center: either a robot successfully moves to a node in the center of the network, or, at least one robot sees an inconsistency and narrows down its list of possible starting locations. So, after enough repetitions of March-to-Center, we can guarantee that there will be a robot with a singleton ID that is located in the center of the network. The location of the robot with the smallest such singleton ID is chosen as v_{target}, the place where the robots will gather.

The third part of the algorithm gets each robot to successfully move to the target node v_{target}, which completes the gathering process. This is the purpose of our Merge subroutine. As above, if a robot starts Merge knowing where in its snapshot view it is located, then it can simply compute a sequence of port numbers that leads to v_{target} and follow it. The difficult case is when a robot α is not sure where in its snapshot it is located at the start of Merge. In this case, α just tries one node from its list of possibilities, computes a sequence of port numbers that leads to v_{target}, and tries to follow it. If it notices any inconsistencies along the way or after it arrives, it deletes the guessed starting node from its list. After each Merge, each robot reverses the steps it took during the Merge in order to go back to where it started so that it can run Merge again. Each execution of Merge finishes in one of two ways: all robots have gathered, or, at least one robot has eliminated one incorrect guess about its starting position. So, after a carefully chosen number of repetitions, we can guarantee that the last performed Merge gathers all robots at the same node.

5.2.2 Algorithm Details

First, we define the subroutine **Find-Lookout()**, which we assume is executed by robot α located at a node v in round 0.

- α stores the initial snapshot $Sview(v, 0)$ in its memory as S_0.
- α determines which nodes in S_0 might be its starting location, i.e., α computes the set of nodes $w \in Sview(v, 0)$ where the degree of w and the list of robot ID's at w is the same as v's local view in round 0.
- For each such w in turn:
  - α computes a port sequence τ corresponding to a depth-first traversal of S_0 starting at w, and attempts to follow this port sequence in the actual network.
  - In every round of the attempted traversal, α takes note of the number of nodes it sees in its snapshot view, and remembers the maximum such number n_{max}, a node v_{max} where it witnessed this maximum, the number m_{max} of robots it saw in its snapshot when located at v_{max}, and the sequence of ports τ_{max} that it used to reach v_{max} from its starting location.
  - Whether or not this attempt is successful, α returns to its starting node by reversing the steps it took during the attempt.
- For the largest n_{max} seen in any of the traversal attempts, α goes to the corresponding node v_{max} using the sequence τ_{max}.

Next, we define the subroutine **March-to-Center(P_α)**, whose goal is to establish a singleton ID in the center of the network G and select its location as the target node v_{target} at which to gather. At a high level, the parameter P_α represents a subset of nodes in α's snapshot view such that each element in P_α is a node that α believes could be its current location. Suppose that a robot α starts its execution of **March-to-Center** at node v in round t.

- If $|P_\alpha| = 1$, then
  - α uses its current snapshot $Sview(v, t)$ to compute a shortest path π starting at the node $v_0 \in P_\alpha$ and ending at a node $v_{closest}$ in the center graph $C(Sview(v, t))$ that minimizes the distance $d(v_0, v_{closest})$.
  - α moves according to the port sequence in π and then waits $H - |\pi|$ rounds at $v_{closest}$.
- If $|P_\alpha| > 1$, then
  - α waits at its current node v for H rounds, and observes every node $v_j \in P_\alpha$ in every snapshot view during the waiting period.
  - If, in any round of the waiting period, α notices that some v_j does not have any robot with ID l_α, it removes v_j from P_α (as it has determined that it is not currently located at v_j).
- In both cases, at the end of the waiting period, α checks if there is a singleton ID in the center graph $C(Sview(v, t + H))$ of its snapshot view. If there is such a singleton ID, α sets v_{target} as the node that contains a robot with the smallest

singleton ID in $C(Sview(v, t + H))$. Otherwise, v_{target} is set to *null*.

Next, we define the subroutine **Merge**$(P_\alpha, v_{target}, n)$ that will be used to attempt to gather the robots at the chosen target node v_{target}. At a high level, the parameter P_α represents a subset of nodes in α's snapshot view such that each element in P_α is a node that α believes could be its current location. The parameter n represents the number of nodes that α has calculated to be the network size.

- Using its own snapshot view, robot α determines a shortest path π starting at the first node $v_0 \in P_\alpha$ and ending at v_{target}.
- Then, α attempts to move according to the port sequence in π to reach v_{target}.
- If there is a round in which the next port to take along path π does not exist in α's local view, or, α arrived at its current node using a port that is different than the port it should have arrived on according to path π, then:
  - α deletes v_0 from P_α, then waits $H - t_{\alpha,Move}$ rounds at its current node, where $t_{\alpha,Move}$ is the number of rounds that α took to reach its current node.
- Else, robot α was able to follow the port sequence in π without noticing any inconsistency, so:
  - α waits $H - |\pi|$ rounds at its current node v. In each of these rounds t', robot α considers its current snapshot $Sview(v, t')$:
    - If α sees that the number of nodes in this view is less than n, then α removes v_0 from P_α.

- If α sees no robot with l_α at v_{target} in $Sview(v, t')$, then α removes v_0 from P_α.
- If it sees that its local view does not match the local view of v_{target} in $Sview(v, t')$ (i.e., it sees a different degree, or a different list of robots at time t'), then α removes v_0 from P_α.

Finally, using the subroutines defined above, we describe the complete **H-View-Algorithm** that is executed by each robot α to solve Gathering. Suppose that α starts the algorithm located at a node v in round 0.

1. α executes **Find-Lookout()**, after which it is located at a node v_{max}.
2. α waits at v_{max} until round $(m_{max} + 2) \cdot n_{max}^2$, which we'll call round x.
3. In round x, robot α creates a set P_α consisting of all nodes in $Sview(v_{max}, x)$ where α might currently be located, i.e., the nodes $w \in Sview(v_{max}, x)$ where the degree of w and the list of robot ID's at w is the same as v's local view in round x.
4. initialize v_{target} to null, initialize *phase* to 1.
5. repeat the following until $(v_{target} \neq null)$:
   (a) α executes **March-to-Center**(P_α)
   (b) $phase \leftarrow phase + 1$
6. repeat the following until $phase > \lceil \frac{m_{max}}{2} \rceil$
   (a) α executes **Merge**$(P_\alpha, v_{target}, n_{max})$

(b) α performs the steps of previous **Merge** in reverse order

(c) $phase \leftarrow phase + 1$

7. α executes **Merge**(P_α, v_{target}, n_{max})

8. terminate()

5.3 Analysis

We consider three main parts of the algorithm. Our first goal is to show that, immediately after robot α executes **Find-Lookout**, it has moved to a node v_{max} such that the snapshot view from v_{max} contains $n_{max} = n$ the actual number of nodes in the graph and $m_{max} = m$the actual number of robots in the team.

Lemma 5.3.1. *By round $(m + 2) \cdot n^2$, each non-faulty robot α is located at a node v_{max} such that the snapshot view at v_{max} contains n nodes and m robots.*

Proof. Consider an arbitrary robot α's execution of the H-View-Algorithm starting at a node v. First, α computes the set of nodes $w \in Sview(v, 0)$ where the degree of w and the list of robot ID's at w is the same as v's local view in round 0. In particular, this means that each such node w contains α's ID l_α in its list of robots. Since at most $f + 1$ robots can have ID l_α in round 0 (i.e., α itself and at most f Byzantine robots), we get that the number of nodes w in $Sview(v, 0)$ that look the same as $Lview(v, 0)$ is at most $f + 1$. Consequently, this means that the number of different depth-first traversals attempted by α is at most $f + 1$. Each depth-first traversal takes at most $2|E|$ rounds, which is less than n^2. Together with the reversal to return

back to its starting node, we get that each attempt takes at most $2n^2$ rounds, so all traversals are complete by round $2(f+1) \cdot n^2$. Since one of the computed traversal sequences starts at α's real initial location, it follows that at least one of the traversal attempts visits all nodes in $Sview(v, 0)$. By the definition of the network's center and the fact that $H \geq R$, it follows that $Sview(v, 0)$ must contain a node that is in the network's center, and we just showed that α necessarily visited all nodes in $Sview(v, 0)$. Since the snapshot view at any node in the center of the network contains all of the network's nodes (since $H \geq R$), it follows that α visits at least one node at which the snapshot view contains all n nodes (and contains all m robots). Robot α will save such a node as v_{max}, it will set $n_{max} = n$ and $m_{max} = m$, and it will set τ_{max} to be a port sequence from v to v_{max}. The final traversal of the path τ_{max} to get from v to v_{max} takes at most another n^2 rounds, so, in total, α arrives at v_{max} by round $(2f+3) \cdot n^2$. Since the number of non-faulty robots is at least $f+1$, we get that $m \geq 2f+1$, so $f \leq \frac{m-1}{2}$. Thus, $(2f+3) \cdot n^2 \leq (m+2) \cdot n^2$. □

The second part of the algorithm consists of the executions of March-to-Center. Our main goal is to prove that, after at most $f+1$ executions of March-to-Center, every robot sets its v_{target} variable to the same non-null value. To this end, we prove that each execution of March-to-Center by the non-faulty robots is started at the same time (Lemma 5.3.5), and, at the end of each execution, every robot is located at a node such that its snapshot contains all of the network's nodes (Lemma 5.3.2). These facts will be used later to conclude that any particular feature seen by one robot can be seen by all other robots at the same time.

Lemma 5.3.2. *At the end of each execution of March-to-Center by any non-faulty*

robot α, the robot resides at some node v such that its snapshot view contains all the nodes of G.

Proof. We consider the two cases in the description of March-to-Center. We note that, at the end of each execution of March-to-Center by a non-faulty robot α, either α is at the node v_{max} where it started the execution, or, it is at a node $v_{closest}$ which is defined to be in $C(Sview(v_{max}, t))$, i.e., the center graph of α's snapshot view from node v_{max}. In the first case, Lemma 5.3.1 tells us that the snapshot view from node v_{max} contains all the nodes of G. In the second case, we observe that $v_{closest}$ is in the center of G since it is in the center graph of α's snapshot view from node v_{max} (which contains all nodes of G). But by the definition of center, the distance from $v_{closest}$ to any node in G is at most $R \leq H$, so all nodes of G are in the snapshot view from $v_{closest}$ as well. □

Lemma 5.3.3. *Suppose that every non-faulty robot starts an execution of March-to-Center in the same round $t' > 0$. In round $t' + H$, every non-faulty robot has the same snapshot view.*

Proof. We see from the description of March-to-Center that there can be two cases in each execution: moving along the path π for $|\pi|$ rounds followed by a waiting period of length $H - |\pi|$, or, a waiting period of length H. In both cases, the execution takes exactly H rounds. Now, by Lemma 5.3.2, we see that at the end of the execution, i.e., in round $t' + H$, each robot's snapshot view is the entire graph. □

Lemma 5.3.4. *For any positive integers i and t', suppose that every non-faulty robot starts its i^{th} execution of March-to-Center in round t'. Then, at the start of round*

$t' + H$, exactly one of the following is true: (i) every non-faulty robot sets v_{target} equal to a non-null value, or, (ii) every non-faulty robot has v_{target} equal to null, and they all start their $(i+1)^{th}$ execution of March-to-Center.

Proof. By Lemma 5.3.3, in round $t' + H$, every robot gets the same snapshot view S. There are two cases to consider. In the first case, suppose that there is a singleton ID in the center graph of S. Then, according to the description of March-to-Center, every non-faulty robot sets its variable v_{target} to the node that contains a robot with the smallest singleton ID, which implies that every non-faulty robot has v_{target} equal to a non-null value. In the second case, suppose that there is no singleton ID in the center graph of S. Then, according to the description of March-to-Center, v_{target} at each non-faulty robot remains null. According to the description of the H-View-Algorithm, this means that all non-faulty robots will execute March-to-Center again. □

We now proceed to show that each execution of March-to-Center by the non-faulty robots is started at the same time. This is useful because it means that the robots make decisions using the same snapshot view, which minimizes the influence of the Byzantine robots: if a Byzantine robot imitates a non-faulty robot's ID l in a fixed round t, then it cannot imitate any other ID's in the same round.

Lemma 5.3.5. *For any positive integer k, suppose that all non-faulty robots start their k^{th} execution of March-to-Center and have $v_{target} = null$. For every positive integer $i \leq k$, every non-faulty robot starts executing its i^{th} execution of March-to-Center in round $(m+2)n^2 + (i-1)H$.*

Proof. We prove the statement by induction on i.

Base case: From the description of the H-View-Algorithm, each non-faulty robot executes March-to-Center for the first time starting in round $(m_{max} + 2) \cdot n_{max}^2 = (m + 2) \cdot n^2$. Thus, the statement is true for $i = 1$.

Inductive step: Assume that, for some $j \in \{1, \ldots, k-1\}$, the statement is true for $i = j$. In particular, assume that every robot started its j^{th} execution of March-to-Center in round $(m + 2)n^2 + (j - 1)H$. By the description of March-to-Center, there can be two cases in their j^{th} execution: moving along the path π for $|\pi|$ rounds followed by a waiting period of length $H - |\pi|$, or, a waiting period of length H. In both cases, the execution takes exactly H rounds. By Lemma 5.3.4 and the fact that no robot has set its v_{target} variable to a non-null value before the k^{th} execution, we get that in round $(m + 2)n^2 + (j - 1)H + H = (m + 2)n^2 + j \cdot H$, every robot starts its $(j + 1)^{th}$ execution of March-to-Center. □

Lemma 5.3.6. *Let $k > 0$ be the smallest integer such that at least one non-faulty robot sets its v_{target} to a non-null value during its k^{th} execution of March-to-Center, and suppose that this execution of March-to-Center starts in round t'. Then, every non-faulty robot sets v_{target} to the same value at the start of round $t' + H$.*

Proof. By Lemma 5.3.5, for every positive integer $i \leq k$, every robot starts its i^{th} execution of March-to-Center in the same round, so all robots start the k^{th} execution of March-to-Center in round t'. Lemma 5.3.4 implies that, at the start of round $t'+H$, either every robot sets a non-null value of v_{target}, or, variable v_{target} is null for every robot. The second case does not occur since we know that at least one non-faulty robot sets its v_{target} to a non-null value during its k^{th} execution of March-to-Center.

Therefore, the first case occurs: all robots set their v_{target} to a non-null value at the start of round $t' + H$. Moreover, by Lemma 5.3.3, every robot has the same snapshot view S in round $t'+H$. Hence, by the description of March-to-Center, every robot sets its v_{target} to the same node: the node that contains a robot with smallest singleton ID in the center graph of S. □

Corollary 5.3.6.1. *If there exists a positive integer k such that at least one non-faulty robot sets its v_{target} to a non-null value during its k^{th} execution of March-to-Center, then all non-faulty robots set v_{target} to the same non-null value at the start of round $(m + 2)n^2 + kH$.*

We now set out to show that all robots set their v_{target} variable to a non-null value within $f + 1$ executions of March-to-Center. The idea behind the proof is to show that, in each execution of March-to-Center that ends with $v_{target} = null$, at least one non-faulty robot makes progress towards determining its correct location within its snapshot view. Once there are enough robots that have determined their correct location (more than the number of Byzantine robots), we are guaranteed to have at least one singleton ID appear in the center of the graph, and all robots will set their v_{target} as the location of the smallest such ID.

To formalize the argument, we introduce a function Φ that measures how much progress has been made by all robots towards determining their correct location within their snapshot view. In what follows, for each $t \geq (m + 2) \cdot n^2$, we denote by $P_{\alpha,t}$ the value of variable P_α at robot α in round t. From the description of the H-View-Algorithm, recall that P_α is set by each robot α for the first time in round $(m+2)\cdot n^2$, and the value assigned in this round is the set of nodes in α's snapshot view that

match its local view, i.e., the nodes that have the same degree and the same list of robot ID's as α's current location. In subsequent rounds, the only changes to P_α involve the removal of nodes, so $P_{\alpha,t+1} \subseteq P_{\alpha,t}$ for all $t > (m+2) \cdot n^2$. For any fixed round $t \geq (m+2) \cdot n^2$, we denote by Φ_t the sum $\sum_\alpha |P_{\alpha,t}|$, which is taken over all non-faulty robots α. We now prove some useful bounds on Φ_t and how its value changes in each execution of March-To-Center.

Proposition 5.3.7. *In any round $t \geq (m+2) \cdot n^2$, we have $m - f \leq \Phi_t \leq m$.*

Proof. First, we show that $\Phi_t \leq m$. Since each $P_{\alpha,t}$ only contains nodes where the ID l_α appears in round t, it follows that $|P_{\alpha,t}|$ is bounded above by the number of robots whose ID in round t is equal l_α. As each robot has exactly one ID in round t (including the Byzantine robots), it follows that $\Phi_t = \sum_\alpha |P_{\alpha,t}| \leq m$. Next, to show that $\Phi_t \geq m - f$, we observe that there are $m - f$ non-faulty robots, and each non-faulty robot α has $|P_{\alpha,t}| \geq 1$ in every round $t \geq (m+2) \cdot n^2$. This is because a non-faulty robot α only removes a node v from P_α if it performs March-to-Center or Merge under the assumption that it starts the execution from node v in its snapshot view, but notices an inconsistency between this assumption and its observed experience. Since α's actual v_{max} node from which it starts March-to-Center or Merge would not result in any inconsistency, this node would never be removed from P_α, which implies that $|P_\alpha| \geq 1$ after the first round in which P_α is given a value. □

Lemma 5.3.8. *Consider any execution of March-to-Center by the non-faulty nodes, and suppose that the execution starts in round t'. Then, exactly one of the following occurs: (i) all non-faulty robots set their v_{target} variable to a non-null value at the start of round $t' + H$, or, (ii) $\Phi_{t'+H} \leq \Phi_{t'} - 1$.*

Proof. By Lemma 5.3.4, exactly one of the following occurs at the start of round $t' + H$:

- All non-faulty robots set their v_{target} variable to some non-null value, or,
- Variable v_{target} is null for every robot. By Lemma 5.3.3, we know that in round $t + H'$, all non-faulty robots have the same snapshot view S, and, by Lemma 5.3.2, S contains all the nodes of G. As there are at least $f+1$ non-faulty robots and exactly f Byzantine robots, there must be at least one non-faulty robot β whose ID will be a singleton ID in S. But since v_{target} is null for every non-faulty robot, this implies that there is no singleton ID in $C(S)$ in round $t' + H$, and so β is located outside of $C(S)$. According to the description of March-to-Center, it must be the case that $|P_\beta| > 1$ in round t', because otherwise β would have moved to a node in the center of its snapshot view in this execution of March-to-Center. Consequently, according to March-to-Center, the robot β removes all other nodes from P_β except the one node that contains its ID l_β (as l_β is a singleton ID). Thus, the value of $|P_\beta|$ decreases during some round in the range $t', \ldots, t' + H$, so it follows that $\Phi_{t'+H} \leq \Phi_{t'} - 1$.

□

Theorem 5.3.9. *There exists a positive integer $k \leq f + 1$ such that every non-faulty robot sets its variable v_{target} to the same non-null value at the start of round $(m + 2)n^2 + kH$.*

Proof. First, suppose that there is at least one non-faulty robot that sets its v_{target} to a non-null value during one of its first f executions of March-to-Center. In this

case, the desired result follows directly from Corollary 5.3.6.1. So, in what follows, we assume that all non-faulty robots have $v_{target} = null$ during the first f executions of March-to-Center. Therefore, all non-faulty robots start their $(f+1)^{th}$ execution of March-to-Center with $v_{target} = null$, and by Lemma 5.3.5, they start this execution in round $(m+2)n^2 + fH$. By Lemmas 5.3.2 and 5.3.3, each non-faulty robot starts this execution with the same snapshot view, which we'll denote by S, that contains all the nodes of G.

By Lemma 5.3.8, after each of the first f executions of March-to-Center, the value of Φ decreases by at least 1. It follows that $\Phi_{(m+2)n^2+fH} \leq \Phi_{(m+2)n^2} - f$. However, by Proposition 5.3.7, we know that $\Phi_{(m+2)n^2} \leq m$ and $\Phi_{(m+2)n^2+fH} \geq m - f$, so altogether we conclude that $\Phi_{(m+2)n^2+fH} = m - f$. But $m - f$ is the number of non-faulty robots, so the sum $\Phi_{(m+2)n^2+fH} = \sum_\alpha |P_{\alpha,(m+2)n^2+fH}|$ has $m - f$ non-zero terms. This implies that each $|P_{\alpha,(m+2)n^2+fH}|$ is equal to exactly 1. Therefore, by the description of March-to-Center, all non-faulty robots move to a node in the center graph of their snapshot view S. This means that there are at least $f+1$ non-faulty robots in the center of S in round $(m+2)n^2 + (f+1)H$, and at least one of their ID's is a singleton ID since there are at most f Byzantine nodes. Thus, by the description of March-to-Center, every non-faulty robot sets its v_{target} to the same node: the node that contains a robot with smallest singleton ID in the center graph of S, which proves the desired statement with $k = f + 1$. □

Now we come to the third part of the algorithm which consists of the executions of Merge. By the description of the H-View-Algorithm, non-faulty robots start executing their Merge operation immediately after setting a non-null value of v_{target}. Moreover,

by Theorem 5.3.9, we see that every robot sets its v_{target} variable to the same non-null value in the same round, and so every non-faulty robot starts executing its first execution of Merge at the same time as well. More specifically, we denote by k the number of executions of March-to-Center performed by the non-faulty robots, and conclude that all non-faulty robots start their first execution of Merge in round $(m+2)n^2 + kH$. By the description of Merge, each execution of Merge consists of exactly H rounds, and according to the H-View-Algorithm, an additional H rounds are then used to perform the steps of Merge in reverse. These observations imply the following fact.

Lemma 5.3.10. *For any positive integer i, if an i^{th} execution of Merge is performed, then all non-faulty robots start this execution in round $(m+2)n^2 + (k+2(i-1))H$.*

Our final goal is to show that all non-faulty robots gather at v_{target} after at most $(f+2)-k$ executions of Merge, where k is the number of March-to-Center operations executed by the non-faulty robots. Before proving this in Theorem 5.3.14, we establish the following technical results.

Lemma 5.3.11. *For any $t \geq 0$, suppose that v is a node such that at least $m-f$ robots are located at v at the start of round t. Then, the local view at v in round t is unique. More precisely, for any node $v' \neq v$, we have $Lview(v', t) \neq Lview(v, t)$.*

Proof. For any v, v' such that $v \neq v'$, if there are at least $m-f$ robots at v in round t, there can be at most f robots at v' in round t. Since there are at least $f+1$ non-faulty robots, it follows that $m \geq 2f+1$, so $m-f > f$. In particular, this means that the number of ID's in $Lview(v, t)$ is strictly greater than the number of ID's in

$Lview(v', t)$, so $Lview(v, t) \neq Lview(v', t)$. □

Lemma 5.3.12. *Consider any execution of Merge by the non-faulty nodes, and suppose that the execution starts in round t'. Then at least one of the following holds: (i) all non-faulty robots are gathered at v_{target} in round $t'+H$, or, (ii) $\Phi_{t'+H} \leq \Phi_{t'} - 1$.*

Proof. Assume that (i) does not hold in round $t' + H$, i.e., at least one non-faulty robot is not located at v_{target} in round $t' + H$. There are two possibilities:

- **There are at least $m - f$ robots at v_{target} in round $t' + H$.** By Lemma 5.3.11, each robot β that is at a node $v' \neq v_{target}$ in round $t'+H$ has a local view $Lview(v', t' + H)$ that is different than $Lview(v_{target}, t' + H)$. Hence, according to the description of Merge, each such robot β removes a node from its P_β, i.e., the value of $|P_\beta|$ decreases in some round in the range $t', \ldots, t' + H$. It follows that $\Phi_{t'+H} \leq \Phi_{t'} - 1$.

- **There are fewer than $m - f$ robots at v_{target} in round $t' + H$.** As the number of non-faulty robots is $m - f$, it follows that there is at least one non-faulty robot α whose ID l_α is not seen at v_{target} in α's snapshot view in round $t' + H$. Hence, according to the description of Merge, α removes a node from its P_α, i.e., the value of $|P_\alpha|$ decreases in some round in the range $t', \ldots, t' + H$. It follows that $\Phi_{t'+H} \leq \Phi_{t'} - 1$.

□

Lemma 5.3.13. *During the execution of the H-View-Algorithm, if $k \geq 1$ executions of March-to-Center are performed followed by $f + 2 - k$ executions of Merge, then all*

non-faulty robots are gathered at v_{target}.

Proof. By the description of the H-View-Algorithm and Corollary 1, if k executions of March-to-Center are performed, then v_{target} was set for the first time by all non-faulty robots at the end of the k^{th} execution of March-to-Center. By Lemma 5.3.8, after each of the first $k-1$ executions of March-to-Center, the value of Φ decreases by at least 1. It follows that $\Phi_{(m+2)n^2+(k-1)H} \leq \Phi_{(m+2)n^2} - (k-1)$. By Proposition 5.3.7, we know that $\Phi_{(m+2)n^2} \leq m$, so it follows that $\Phi_{(m+2)n^2+(k-1)H} \leq m-(k-1)$. Since the value of Φ never increases (the algorithm only ever removes nodes from the P_α sets) it follows that $\Phi_{(m+2)n^2+kH} \leq m-(k-1)$ as well, where round $(m+2)n^2+kH$ is when the first Merge execution begins. Now, we consider the first $f+1-k$ executions of Merge by the non-faulty robots, and we consider two cases:

- Suppose that, for some $i \in \{1,\ldots,f+1-k\}$, all non-faulty robots are gathered at v_{target} at the end of the i^{th} execution of Merge. Since the number of non-faulty robots is $m-f$, it follows that there would be at least $m-f$ robots at v_{target}. By Lemma 5.3.11, the local view at v_{target} would be unique in G, and the local view of each non-faulty robot would exactly match it. Hence, according to the description of Merge, no non-faulty robot would modify its P_α set, and so the next execution of Merge (if any) would start from the same node v_0. It follows that in all subsequent executions of Merge (in particular, the $(f+2-k)^{th}$ execution) all non-faulty robots will be gathered at v_{target}.

- Suppose that, for every $i \in \{1,\ldots,f+1-k\}$, at least one non-faulty robot is not located at v_{target} at the end of the i^{th} execution of Merge. Then, according

to Lemma 5.3.12, the value of Φ decreases by at least 1 in each such execution. As the value of Φ was bounded above by $m-(k-1)$ at the start of the first Merge execution, and it decreases by at least $f+1-k$ during the first $f+1-k$ executions of Merge, it follows that, after the $(f+1-k)^{th}$ execution of Merge, the value of Φ is at most $m-f$. However, by Proposition 5.3.7, we know that Φ is at least $m-f$, so altogether we conclude that the value of Φ after the $(f+1-k)^{th}$ execution of Merge is exactly $m-f$. But $m-f$ is the number of non-faulty robots, so the summation represented by Φ has $m-f$ non-zero terms. This implies that each $|P_\alpha|$ is equal to exactly 1 for each non-faulty robot α. Then, in the final execution of Merge, i.e., in execution $f+2-k$, each non-faulty robot will compute a path to v_{target} using its snapshot view, but using its actual location as starting node v_0. This means that all non-faulty nodes will be located at v_{target} at the end of execution $f+2-k$ of Merge.

□

Finally, we verify that the H-View-Algorithm ensures that Merge is executed at least $f+2-k$ times after k executions of March-to-Center. The Merge operation is executed until the value of *phase* is greater than $\lceil m/2 \rceil$, and from the assumption that the number of non-faulty robots is at least $f+1$, we know that $m \geq 2f+1$. In particular, this means that the combined number of March-to-Center and Merge executions is at least $f+1$, and then one more Merge is executed after exiting the 'repeat' loop. This concludes the proof of correctness of the H-View-Algorithm.

Theorem 5.3.14. *In any n-node graph with radius R, if the H-View-Algorithm is*

performed by any team of m robots consisting of f Byzantine robots and at least $f+1$ non-faulty robots with visibility $H \geq R$, then Gathering is solved within $(m+2) \cdot n^2 + H \cdot m \in O(mn^2)$ rounds.

Proof. By Lemma 5.3.1, every non-faulty robot spends exactly $(m+2)n^2$ rounds for the **Find-Lookout** operation. Then, by Theorem 5.3.9, there exists a positive integer $k \leq f+1$ such that every non-faulty robot sets its variable v_{target} at the start of the round $(m+2)n^2 + kH$. More precisely, robots spend exactly kH rounds performing the **March-to-Center** executions. After that, every robot spends exactly $(\lceil m/2 \rceil - k)2H + H$ rounds for its **Merge** executions, after which all non-faulty are located at v_{target} (by Lemma 5.3.13). In total, the number of rounds is $(m+2) \cdot n^2 + kH + (\lceil m/2 \rceil - k)2H + H$. For the minimum value of $k = 1$, we get that the robots use at most $(m+2) \cdot n^2 + H \cdot m$ rounds to accomplish the gathering. As $H \leq n$ (at most full visibility), the number of rounds is in $O(mn^2)$, i.e., polynomial in the network size and team size. □

Chapter 6

Impossibility Results

In this Chapter, we provide three impossibility results. In Chapter 6.1, we strengthen a known result for Gathering in the presence of weakly Byzantine robots. In Chapter 6.2, we re-prove a known result so that it applies to the main model considered in this thesis, which allows us to conclude that our H-View-Algorithm uses the optimal number of non-faulty robots. In Chapter 6.3, we prove a lower bound on the visibility range required in order to solve Gathering in the presence of strongly Byzantine robots.

6.1 A lower bound on the number of non-faulty robots: weakly Byzantine faults, unknown graph size, and $H = 0$

In [23], the authors prove that no algorithm can solve gathering if the number of non-faulty robots is less than $f + 2$ with visibility $H = 0$. They assume that the graph size is unknown and the faulty robots are weakly Byzantine. The authors also assume that the upper bound f on the number of Byzantine robots is known to all non-faulty robots. Now, we prove that this lower bound of $f + 2$ holds even if the non-faulty robots know the exact number of faulty and non-faulty robots in the team.

Theorem 6.1.1. *No algorithm can solve Gathering for the model with weakly Byzantine faults and unknown size of the graph if the number of non-faulty robots (with visibility 0) is less than $f + 2$, even if the robots know the exact values of m and f.*

Proof. To prove the theorem, it is enough to show that, in some class of network $G_{n,k}$, $f+1$ number of non-faulty robots can not accomplish Gathering, with visibility range $H = 0$, in the presence of f Byzantine robots.

First, we recall a class of network $G_{n,k}$ from Theorem 3.11 [23]. Let $n \geq 2$ and $k \geq 4$ be two even integers. We construct a network $G_{n,k}$ in such a way that the set of nodes in $G_{n,k}$ can be divided into k pairwise disjoint clusters, and each cluster contains n number of nodes (there will be kn number of nodes in total in the network). We denote by $C_0, C_1, C_2..., C_{k-1}$ the sequence of clusters in $G_{n,k}$. The nodes in the network are connected as follows: the nodes inside the same cluster are connected

to each other, every node of each cluster is connected to every node of its adjacent clusters, i.e., every nodes of cluster C_i is connected to every node of cluster C_{i-1} and C_{i+1} (arithmetic on indices is modulo k). Hence, every node in the graph has degree $3n-1$. Now, at each node u, the incident edges are labeled in such a way that an edge $\{u, v\}$ has the same port number at node u and at node v, and is called the color of the edge. Edges inside a cluster are colored with $0, 1, ..., n-2$ (this is always possible since n is even and every complete graph with an even number of nodes is 1-factorable). Every node u of a cluster C_i is adjacent to every node of cluster C_{i+1} and C_{i-1}, colored as follows. If i is even, the colors of the edges between u and nodes in C_{i+1} would be $n-1, n, ..., 2n-2$. And if i is odd, then the colors of the edges between u and nodes in C_{i+1} would be $2n-1, 2n...., 3n-2$. Hence, every network $G_{n,k}$ is fully symmetric (the local view of a robot is the same regardless of its location within the network). We denote by $C_{\mathcal{G}}$ the class of networks $G_{n,k}$.

Suppose, there exists an algorithm A that accomplishes Gathering in every network of class $C_{\mathcal{G}}$ with $f+1$ non-faulty robots in the presence of f weakly Byzantine robots in the team. First, we consider an execution EX_1 of algorithm A in a network $G_{f+2,4}$ of class $C_{\mathcal{G}}$. The graph $G_{f+2,4}$ implies that there are 4 clusters in the network, and each cluster contains $f+2$ number of nodes. We consider that there are even number of Byzantine robots in the network (f is even), and so each cluster's size $f+2$ is even as well. The initial positions of the robots are as follows: f Byzantine robots with ID's $f+2, \cdots 2f+1$ are placed all together at a single node v in cluster C_0, $f+1$ non-faulty robots with ID's $1, 2, .., f+1$ are placed at distinct nodes of cluster C_0 (except at node v). Now, we describe the behaviors of the Byzantine robots: the

Byzantine robots hide themselves from the non-faulty robots throughout the execution by not meeting any non-faulty robot in the execution. If, at some round $t > 0$, the Byzantine robots are required to move from one node to another node (in order to hide themselves), they all move to the same node in cluster C_0 (they always stay inside the cluster C_0). More precisely, we can define the behaviour of the Byzantine robots during rounds $t = 1, 2, \cdots$ of execution EX_1 as follows.

- Suppose, at some round $t - 1 \geq 0$, the Byzantine robots are located at some node $u \in C_0$. Now, if the algorithm prescribes some non-faulty robots to move to node u at round t, then all Byzantine robots move to some other node $w \in C_0$ in round t, such that there would not be any other robot (other than the Byzantine robots) at w in round t. This is always possible since there are $f + 2$ nodes in each cluster, and there can be at most $f + 1$ nodes occupied by the non-faulty robots at a time, and so there is always a node in C_0 at which there is no non-faulty robot.

As A is assumed to be a correct algorithm, there exists some round t in which the non-faulty robots terminate and gather at some node v'. More precisely, algorithm A accomplishes Gathering in execution EX_1 within t rounds, in the presence of f Byzantine and $f + 1$ non-faulty robots in the network of $G_{f+2,4}$, where no non-faulty robot meets any Byzantine robot throughout the entire execution.

Next, we consider the second execution EX_2 of algorithm A in a network $G_{f+2,4t+2}$ of class C_k. The graph $G_{f+2,4t+2}$ implies that there are $4t + 2$ clusters in the network, and each cluster contains $f + 2$ number of nodes. There are $f + 1$ non-faulty and

f Byzantine robots in this execution as well, but we switch some IDs between the robots. The robots are labeled as follows: the Byzantine robots are labeled with IDs $2, \cdots, f+1$, one non-faulty robot is labeled with ID 1, and f other non-faulty robots are labeled with IDs $f+2, \cdots, 2f+1$. The initial positions of the robots are as follows: the Byzantine robots with IDs $2, \cdots, f+1$ and one non-faulty robot with ID 1 are placed at distinct nodes of cluster C_0, f other non-faulty robots with IDs $f+2, \cdots 2f+1$ are placed in cluster C_{2t+1}. We now demonstrate that the Byzantine robots in EX_2 can behave in such a way that, for each round $i = 0, 1, \cdots, t$, the non-faulty robot with ID 1 cannot distinguish between executions EX_1 and EX_2. We denote by α the non-faulty robot with ID 1. We can define the behaviour of the Byzantine robots during rounds $i = 1, 2, \cdots, t$ of execution EX_2 as follows.

- For each round $i > 0$, if a non-faulty robot β with ID l_β meets α in round i of execution EX_1, then the Byzantine robot with ID l_β meets α in round i of execution EX_2.

- For each round $i > 0$, if a non-faulty robot β with ID l_β does not meet α in round i of execution EX_1, then the Byzantine robot with ID l_β does not meet α in round i of execution EX_2. More precisely, it stays at some other node which is not occupied by the robot α but located at the same cluster. This is always possible, in view of the structure of the network, since the robots with IDs $1, \cdots, f+1$ are located at the same cluster initially.

The executions EX_1 and EX_2 will be identical according to α's point of view for the following reasons (i) all nodes in networks $G_{f+2,4}$ and $G_{f+2,4t+2}$ look identical,

(ii) there are same number of robots ($f+1$ non-faulty and f Byzantine) in both executions, (iii) In execution EX_2, the Byzantine robots mimic the actions of non-faulty robots $2, 3, ...f+1$ of EX_1, (iv) In EX_2, time t is not enough for the non-faulty robots $f+2, ...2f+1$ starting in cluster C_{2t+1} to meet α which is initially placed in cluster C_0, since the distance between C_0 and C_{2t+1} is $2t+1$, Therefore, robot α will terminate the algorithm in round t in execution EX_2 as well. This leads to a contradiction: since α terminates the algorithm in EX_2 before meeting any other non-faulty robots. □

6.2 A lower bound on the number of non-faulty robots: strongly Byzantine faults and known graph size

First, we recall Theorem 4.7 from [23], which states that there is no deterministic algorithm that solves Gathering in the presence of f strongly Byzantine robots if the number of non-faulty agents is at most f (and these non-faulty agents know the size of the graph). This impossibility result was proven in a model where robots have no visibility beyond their local view (i.e., visibility $H = 0$). We adapt the proof to work under the assumption that each non-faulty robot has full visibility of the entire graph in every round, which proves that our algorithm is optimal with respect to the number of non-faulty robots in the team.

Theorem 6.2.1. *There is no deterministic algorithm that solves Gathering if the*

number of Byzantine robots in the team is f and the number of non-faulty robots is at most f, even if the non-faulty robots have visibility H equal to the diameter of the graph.

Proof. Suppose that there is a deterministic algorithm A such that A accomplishes Gathering in the presence of f non-faulty and f Byzantine robots in the team. We assume that the non-faulty robots have visibility c equal to the diameter of the graph or less.

First, we construct an instance consisting of a cycle graph $G = (V, E)$ with $2f$ number of nodes (i.e., $|V| = 2f$). At each node $v \in V$, the two incident edges are labeled with port numbers 0 and 1 such that 0 leads clockwise and 1 leads anticlockwise. We denote by v_0 an arbitrary node in G, and $v_0, v_1, \cdots, v_{2f-1}$ the sequence of consecutive nodes in clockwise direction. Now, we execute algorithm A under two different initial positions of the robots on G. In the first execution, the initial positions of the robots are as follows: f non-faulty robots with IDs $0, 1, \cdots, f-1$ are placed respectively at nodes $v_0, v_1, \cdots, v_{f-1}$, and f Byzantine robots with IDs $0, 1, \cdots, f-1$ are placed respectively at node $v_f, v_{f+1}, \cdots, v_{2f-1}$.

We consider that in each round $i = 1, 2, \cdots$ of EX_1, each Byzantine robot with ID $l \in 0, 1, \cdots, f-1$ acts same as the non-faulty robot with ID l does in round i (i.e., a Byzantine robot with ID l performs the same instructions of algorithm A for ID l). Consequently, for every ID $l \in 0, 1, \cdots, f-1$, the non-faulty robot with ID l and the Byzantine robot with ID l stay diametrically opposite through out the execution. As A is assumed to be a correct algorithm, there exists some round t in which the non-

faulty robots terminate and gather at some node v. And for the symmetric executions of the Byzantine robots, all Byzantine robots gather at diametrically opposing node w (distance $d(v, w) = f$) in round t.

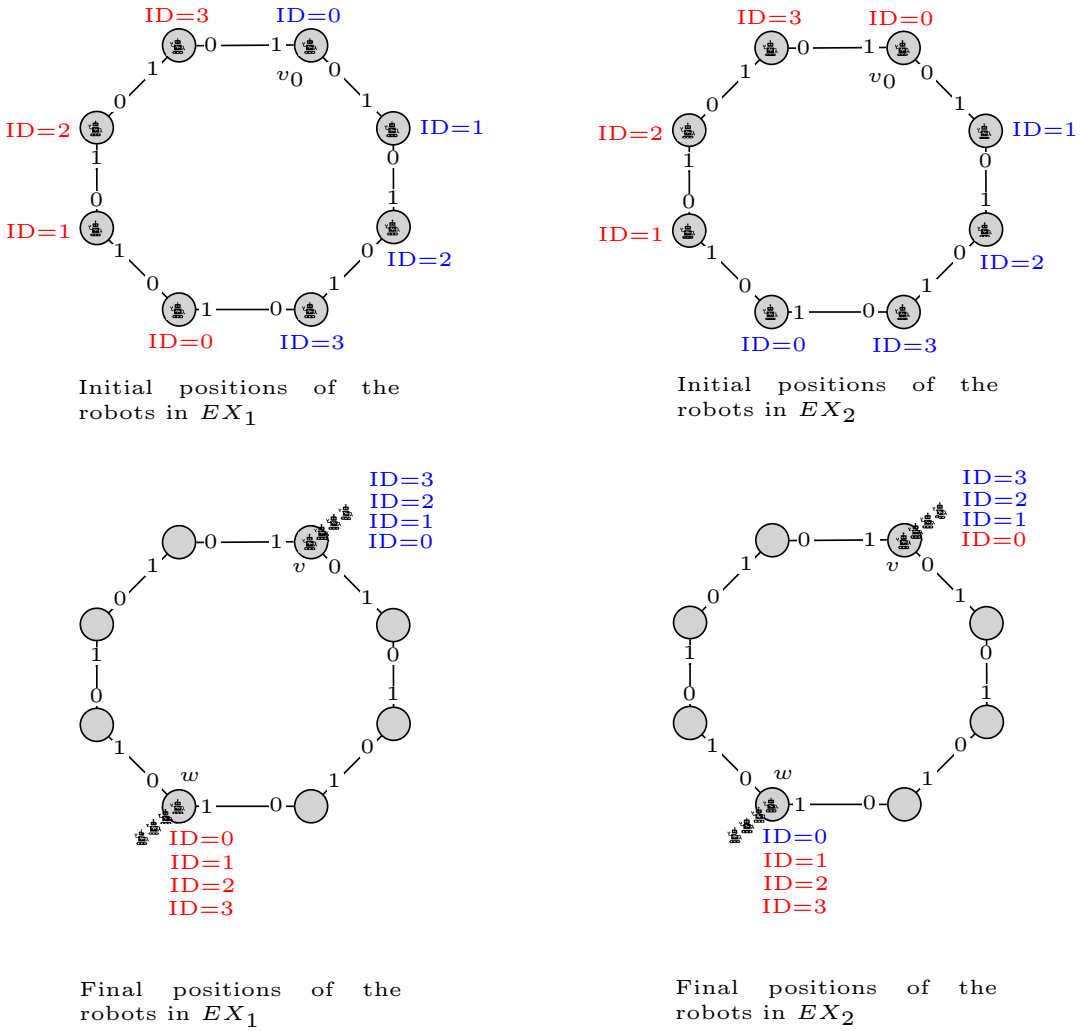


Figure 6.1: Two executions of algorithm A where each red ID represents a Byzantine robot, and each blue ID represents a non-faulty robot in the graph: EX_2 fails to gather the non-faulty robots

Now, for the second execution, we switch the initial positions between the robots with IDs 0. The initial positions of the robots in EX_2 are as follows: the non-faulty robot with ID 0 is placed at node v_f, and $f - 1$ non-faulty robots with ID

$1, \cdots, f-1$ are placed respectively at nodes $v_1, \cdots, v_{f-1}$, the Byzantine robot with ID 0 is placed at node v_0, and $f-1$ Byzantine robots with ID $1, \cdots, f-1$ are placed respectively at node $v_{f+1}, \cdots, v_{2f-1}$. We consider that each Byzantine robot with ID $l \in 0, 1, \cdots, f-1$ acts same as the non-faulty robot with ID l throughout EX_2 as well. We note that visibility range of a non-faulty robot α is c in both executions, which means that its snapshot view consists of $2c+1$ nodes in every round of both executions. Due to symmetric configuration and symmetric executions of the Byzantine robots, in each round $t \geq 0$, α sees exactly the same local view and same snapshot view in both executions. More precisely, no robot can distinguish between executions EX_1 and EX_2. Necessarily, robots will terminate the algorithm in round t in execution EX_2 as well. But this time the non-faulty robot with ID 0 will end up at node w along with $f-1$ Byzantine robots of ID $1, 2, \cdots f-1$, and there will be $f-1$ non-faulty robots at node v along with one Byzantine robot of ID 0 (see Figure 6.1). Hence, algorithm A fails to gather the non-faulty robots in EX_2. This leads to a contradiction that A is not a correct algorithm. □

6.3 A lower bound on the visibility range

For the main model considered in this thesis, we prove that to solve Gathering in arbitrary graphs, the visibility H of each non-faulty robot must somehow depend on the radius of the graph. In particular, it is not sufficient to fix some constant visibility range. We remark that this does not contradict the existence of previously-known algorithms that work when $H = 0$, as those algorithms make additional assumptions

that are not present in our model (e.g., knowledge of the graph size, knowledge of the number of Byzantine robots, or whiteboards at the nodes).

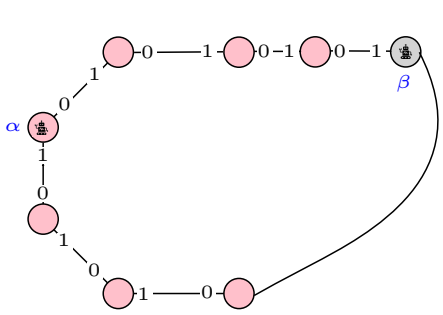
Theorem 6.3.1. *There is no deterministic algorithm that can solve Gathering when executed in any graph by any team of m robots consisting of $f \geq 0$ Byzantine robots and at least $f + 1$ non-faulty robots if the visibility range H of each non-faulty robot is a fixed constant c.*

Proof. Let c be any fixed positive integer. To obtain a contradiction, assume the existence of a deterministic algorithm A that can solve Gathering when executed in any graph by any team of m robots consisting of $f \geq 0$ Byzantine robots and at least $f + 1$ non-faulty robots if the visibility range H of each non-faulty robot is equal to c.

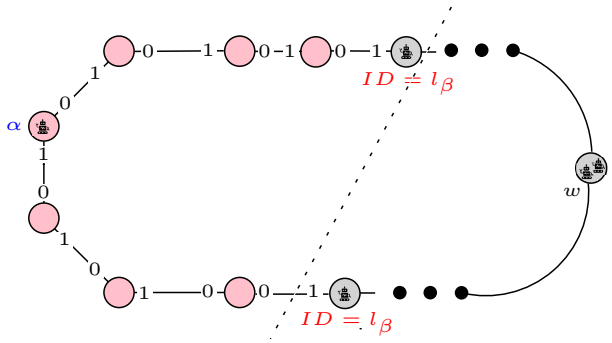
First, we construct an instance consisting of a cycle graph $C_1 = (V_1, E_1)$ with an even number of nodes $|V_1| = 2c + 2$. The radius R_1 of C_1 is $c + 1$. At each node $v \in V_1$, the two incident edges are labeled with port numbers 0 and 1 such that 0 leads clockwise and 1 leads anticlockwise. The initial positions of the robots in C_1 are as follows: a non-faulty robot α with ID l_α is placed at some node v_0, and a non-faulty robot β with ID l_β at a node w such that the distance $d(v_0, w) = R_1 = c + 1$. There are no Byzantine robots in C_1. Consider the execution EX_1 of algorithm A on instance C_1. As A is assumed to be a correct algorithm, there exists some round r_1 in which robots α and β have terminated and gathered at some node $v_{target} \in V_1$.

Next, we construct a second instance consisting of a cycle graph $C_2 = (V_2, E_2)$ with an even number of nodes $|V_2| = 4r_1+2(c+1)$. The radius of R_2 of C_2 is $2r_1+c+1$.

At each node $v \in V_2$, the two incident edges are labeled with port numbers 0 and 1 such that 0 leads clockwise and 1 leads anticlockwise. The initial positions of the robots in C_2 are as follows: the non-faulty robot α with ID l_α is placed at node v_0 (as in the first instance C_1 above), a Byzantine robot with ID l_β is placed at a node v_{CW} that is distance exactly $c+1$ away from v_0 in the clockwise direction, and another Byzantine robot with ID l_β is placed at a node v_{ACW} that is distance exactly $c+1$ away from v_0 in the anticlockwise direction. Further, we place 2 non-faulty robots at a node w such that $d(v_0, w) = R_2 = 2r_1 + c + 1$. These 2 non-faulty robots have distinct ID's that are not equal to l_α or l_β. The number of Byzantine robots is $f = 2$, and there are $3 = f + 1$ non-faulty robots (one at v_0 and two at w). We denote by EX_2 the execution of algorithm A on instance C_2.



Graph C_1 with visbilty range c=3.



Graph C_2 where each red ID represents a Byzantine robot, and there are $4r_1$ nodes on the right side of the dotted line.

Figure 6.2: Initial positions of the robots in two executions of the algorithm where red nodes are inside the visibility range of α

We now demonstrate that the Byzantine robots in C_2 can behave in such a way that, for each round t, the robot α with ID l_α cannot distinguish between executions EX_1 and EX_2, i.e., robot α's local view and snapshot view in every round are the same across both executions. This leads to a contradiction: since α terminates its

algorithm in round r_1 in execution EX_1, it will also terminate its algorithm in round r_1 in execution EX_2, and since the initial distance between α and the other non-faulty robots is strictly greater than $2r_1$, it follows that α terminates before the non-faulty robots can gather.

First, note that α's visibility range is c in both executions, which means that its snapshot view consists of $2c + 1$ nodes in every round of both executions. By the initial placement of the robots in both executions, we note that in round $t = 0$ of both executions, there are no robots within distance c of α's initial position v_0. So, α's local view in round 0 of both executions consists of a node of degree 2 containing the ID l_α, and, α's snapshot view in round 0 of both executions consists of a path of length $2c+1$ nodes with only ID l_α located at the middle node. Further, we note that the two other non-faulty robots in C_2 are never visible to α in in the first r_1 rounds of execution EX_2 execution: their initial distance to α is $2r_1 + c + 1$, so in round r_1, each of their distances to α is at least $c + 1$.

To define the behaviour of the Byzantine robots in C_2 during rounds $t = 1, \ldots, r_1$ of execution EX_2, we observe the execution EX_1. In particular:

- For each round $t > 0$ of EX_1 in which α does not see β in its snapshot view: the Byzantine robots follow the same port in round $t - 1$ of EX_2 as α did in round $t - 1$ in EX_1. Doing so ensures that both Byzantine robots remain at distance $c + 1$ from α at the start of round t in EX_2, i.e., are not in α's snapshot view.

- For each round $t > 0$ of EX_1 in which α sees β in its snapshot view but did not see β in its snapshot view in round $t-1$: the Byzantine robot on the appropriate

side of α (clockwise or counterclockwise) moves so that it appears at the same node in α's snapshot view in round t of EX_2 as β does in round t of EX_1. The other Byzantine robot follows the same port as α does in round $t-1$ (so that its distance from α at the start of round t is still $c+1$, i.e., it does not appear in α's snapshot view).

- For each round $t > 0$ of EX_1 in which α sees β in its snapshot view and also saw β in its snapshot view in round $t-1$: the Byzantine robot that was in α's snapshot view in round $t-1$ of EX_2 follows the same port in round $t-1$ of EX_2 as β did in round $t-1$ of EX_1. The other Byzantine robot follows the same port as α does in round $t-1$ (so that its distance from α at the start of round t is still $c+1$, i.e., it does not appear in α's snapshot view).

It is clear from this behaviour that α sees the same thing up to round r_1 in both executions EX_1 and EX_2: when α sees no other robots in round t of EX_1, then both Byzantine robots move so that they are both at distance $c+1$ from α in round t of EX_2; moreover, when α sees β in round t of EX_1, then one Byzantine robot (which has ID l_β) moves so that its position relative to α in round t of EX_2 is the same as β's relative position to α in round t of EX_1, while the other Byzantine robot moves so that it is at distance $c+1$ from α in round t of EX_2.

□

Chapter 7

Conclusion

7.1 Contributions

Our main contribution of this thesis is to solve the Gathering problem efficiently in the graph model, in the presence of Byzantine robots. We consider a graph-based model in which each robot has no initial information other than its own ID and has some visibility range H. We prove that no algorithm can solve Gathering in the presence of Byzantine robots if H is any fixed constant (Theorem 6.3.1). We also design an algorithm that solves Gathering in any graph with n nodes containing m robots, f of which are strongly Byzantine, and where each non-faulty robot has visibility range H equal to the radius of the graph (or larger). Our algorithm has the following desirable properties:

- It is efficient: the number of rounds is polynomial with respect to n and m, in contrast to several previous algorithms [8, 23] whose running times are expo-

nential in n and the largest robot ID.

- It works when the number of non-faulty robots in the team is $f + 1$ (or larger), which is optimal due to Theorem 6.2. This significantly improves on the best previous polynomial-time algorithm [9], which requires at least $5f^2 + 6f + 2$ non-faulty robots.

- It does not assume any initial global knowledge, in contrast to previous algorithms [8, 9, 23] that assume a known bound on the graph size or on the number of Byzantine robots, or some number of advice bits. Such assumptions might be unrealistic in many applications.

7.2 Future Work

In this thesis, we prove that some constant visibility range c of each non-faulty robot is not sufficient to solve the Gathering in arbitrary graphs. In particular, the visibility H of each non-faulty robot must somehow depend on the radius R of the graph. But, we could not establish any particular lower bound on H with respect to R. We have tried to find a case in which no algorithm can solve the Gathering with visibility $H = R - 1$. But, in order to establish such lower bound, we cannot apply the same proof technique that we used to prove Theorem 6.3.1: when we try to change the underlying graph and use indistinguishably to show that Gathering does not occur in the new graph, the visibility radius of the robot is different in the bigger graph, so we cannot conclude that the robot will behave in the same way as it did in the old. Establishing a lower bound on H with respect to R is left as an open

problem. Furthermore, we may change our defined model such as - we can provide an external memory register (light) that can be visible by other robots, or we can work on the semi-synchronous model, and try to answer the same questions addressed in this thesis.

Bibliography

[1] N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.

[2] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999.

[3] T. Balabonski, A. Delga, L. Rieg, S. Tixeuil, and X. Urbain. Synchronous gathering without multiplicity detection: a certified algorithm. *Theory of Computing Systems*, 2018.

[4] E. M. Barrameda, N. Santoro, W. Shi, and N. Taleb. Sensor deployment by a robot in an unknown orthogonal region: Achieving full coverage. In *20th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2014*, pages 951–960, 2014.

[5] L. Barrière, P. Flocchini, E. M. Barrameda, and N. Santoro. Uniform scattering of autonomous mobile robots in a grid. *Int. J. Found. Comput. Sci.*, 22(3):679–697, 2011.

[6] S. Bhagat, S. G. Chaudhuri, and K. Mukhopadyaya. Gathering of opaque robots in 3d space. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, page 2. ACM, 2018.

[7] S. Bhagat, K. Mukhopadhyaya, and S. Mukhopadhyaya. Computation under restricted visibility. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, pages 134–183. Springer, 2019.

[8] S. Bouchard, Y. Dieudonné, and B. Ducourthial. Byzantine gathering in networks. *Distributed Computing*, 29(6):435–457, 2016.

[9] S. Bouchard, Y. Dieudonné, and A. Lamani. Byzantine gathering in polynomial time. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 147:1–147:15, 2018.

[10] J. Chalopin, S. Das, and A. Kosowski. Constructing a map of an anonymous graph: Applications of universal sequences. In *International Conference On Principles Of Distributed Systems*, pages 119–134. Springer, 2010.

[11] J. Chalopin, Y. Dieudonné, A. Labourel, and A. Pelc. Rendezvous in networks in spite of delay faults. *Distributed Computing*, 29(3):187–205, 2016.

[12] J. Chalopin, P. Flocchini, B. Mans, and N. Santoro. Network exploration by silent and oblivious robots. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 208–219. Springer, 2010.

[13] J. Chalopin, E. Godard, and A. Naudin. Anonymous graph exploration with

binoculars. In *Distributed Computing - 29th International Symposium, DISC 2015*, pages 107–122, 2015.

[14] S. Cicerone, G. D. Stefano, and A. Navarra. Asynchronous robots on graphs: Gathering. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, pages 184–217. Springer, 2019.

[15] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the robots gathering problem. In *International Colloquium on Automata, Languages, and Programming*, pages 1181–1196. Springer, 2003.

[16] M. Cieliebak and G. Prencipe. Gathering autonomous mobile robots. In *SIROCCO*, volume 9, pages 57–72. Citeseer, 2002.

[17] P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. A certified universal gathering algorithm for oblivious mobile robots. *arXiv preprint arXiv:1506.01603*, 2015.

[18] G. D'Angelo, G. Di Stefano, R. Klasing, and A. Navarra. Gathering of robots on anonymous grids and trees without multiplicity detection. *Theoretical Computer Science*, 610:158–168, 2016.

[19] G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theor. Comput. Sci.*, 355(3):315–326, 2006.

[20] X. Défago, M. Potop-Butucaru, and S. Tixeuil. Fault-tolerant mobile robots. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, pages 234–251. Springer, 2019.

[21] A. Dessmark, P. Fraigniaud, D. R. Kowalski, and A. Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.

[22] G. Di Stefano and A. Navarra. Optimal gathering of oblivious robots in anonymous graphs and its application on trees and rings. *Distributed Computing*, 30(2):75–86, 2017.

[23] Y. Dieudonné, A. Pelc, and D. Peleg. Gathering despite mischief. *ACM Transactions on Algorithms (TALG)*, 11(1):1, 2014.

[24] G. D'Angelo, A. Navarra, and N. Nisse. A unified approach for gathering and exclusive searching on rings under weak assumptions. *Distributed Computing*, 30(1):17–48, 2017.

[25] M. Fischer, D. Jung, and F. Meyer auf der Heide. Gathering anonymous, oblivious robots on a grid. In *13th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2017*, pages 168–181, 2017.

[26] P. Flocchini. Gathering. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, pages 63–82. Springer, 2019.

[27] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theoretical Computer Science*, 411(14-15):1583–1598, 2010.

[28] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. How many oblivious robots can explore a line. *Information Processing Letters*, 111(20):1027–1031, 2011.

[29] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65(3):562–583, 2013.

[30] P. Flocchini, G. Prencipe, and N. Santoro, editors. *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*. Springer, 2019.

[31] A. Heriban, X. Défago, and S. Tixeuil. Optimally gathering two robots. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, page 3. ACM, 2018.

[32] T. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, and J. S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Fifth International Workshop on the Algorithmic Foundations of Robotics, WAFR 2002*, pages 77–94, 2002.

[33] T. Okumura, K. Wada, and Y. Katayama. Optimal asynchronous rendezvous for mobile robots with lights. *arXiv preprint arXiv:1707.04449*, 2017.

[34] F. Ooshita, A. K. Datta, and T. Masuzawa. Self-stabilizing rendezvous of synchronous mobile agents in graphs. In *Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS 2017*, pages 18–32, 2017.

[35] D. Pattanayak, K. Mondal, H. Ramesh, and P. S. Mandal. Fault-tolerant gathering of mobile robots with weak multiplicity detection. In *Proceedings of the 18th International Conference on Distributed Computing and Networking*, page 7. ACM, 2017.

[36] A. Pelc. Deterministic gathering with crash faults. *Networks*, 72(2):182–199, 2018.

[37] A. Pelc. Deterministic rendezvous algorithms. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, pages 423–454. Springer, 2019.

[38] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.

[39] A. Ta-Shma and U. Zwick. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In *Symposium on Discrete Algorithms: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, volume 7, pages 599–608, 2007.

[40] M. Tsuchida, F. Ooshita, and M. Inoue. Byzantine-tolerant gathering of mobile agents in arbitrary networks with authenticated whiteboards. *IEICE Transactions*, 101-D(3):602–610, 2018.

[41] G. Viglietta. Rendezvous of two robots with visible bits. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 291–306. Springer, 2013.