Deterministic Gathering with Crash Faults

Andrzej Pelc[*]

Abstract

A team consisting of an unknown number of mobile agents, starting from different nodes of an unknown network, have to meet at the same node and terminate. This problem is known as *gathering*. We study deterministic gathering algorithms under the assumption that agents are subject to *crash faults* which can occur at any time. Two fault scenarios are considered. A *motion fault* immobilizes the agent at a node or inside an edge but leaves intact its memory at the time when the fault occurred. A more severe *total fault* immobilizes the agent as well, but also erases its entire memory. Of course, we cannot require faulty agents to gather. Thus the gathering problem for fault prone agents calls for all fault-free agents to gather at a single node, and terminate.

When agents move completely asynchronously, gathering with crash faults of any type is impossible. Hence we consider a restricted version of asynchrony, where each agent is assigned by the adversary a fixed speed, possibly different for each agent. Agents have clocks ticking at the same rate. Each agent can wait for a time of its choice at any node, or decide to traverse an edge but then it moves at constant speed assigned to it. Moreover, agents have different labels. Each agent knows its label and speed but not those of other agents.

We construct a gathering algorithm working for any team of at least two agents in the scenario of motion faults, and a gathering algorithm working in the presence of total faults, provided that at least two agents are fault free all the time. If only one agent is fault free, the task of gathering with total faults is sometimes impossible. Both our algorithms work in time polynomial in the size of the graph, in the logarithm of the largest label, in the inverse of the smallest speed, and in the ratio between the largest and the smallest speed.

Keywords: gathering, deterministic algorithm, mobile agent, crash fault.

---

[*]Département d'informatique, Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada. E-mail: pelc@uqo.ca. Supported in part by NSERC discovery grant 8136 – 2013 and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

1 Introduction

1.1 The background

A team of at least two mobile entities, called agents, starting from different nodes of a network, have to meet at the same node. This task is known as *gathering*, and it is called *rendezvous* if there are two agents. Both gathering and rendezvous have been extensively studied in the literature under different models. Mobile entities may represent software agents in computer networks, or mobile robots, if the network is a labyrinth or a cave. The reason to meet may be to exchange information or ground samples previously collected by the agents, or to split work in a future task of network exploration or maintenance. The difficulty of the gathering problem, even in the fault-free context, comes from the fact that agents usually do not have any canonical place to meet that is chosen a priori. If agents execute an identical deterministic algorithm, then some symmetry breaking mechanism must be used to enable meeting, as otherwise meeting would be impossible in highly symmetric networks such as, e.g., rings, because agents would make the same moves, always keeping the same distance between them, if they move at the same speed. Since we are interested in deterministic gathering algorithms, we assume that symmetry is broken by assigning a different label to each agent.

In this paper we study the gathering problem under the assumption that agents are subject to *crash faults* which can occur at any time. Two types of such faults are considered. A *motion fault* immobilizes the agent but leaves intact its memory at the time when the fault occurred. A *total fault* immobilizes the agent and erases its entire memory. Of course, we cannot require faulty agents to gather. Thus the gathering problem for fault-prone agents calls for all fault-free agents to gather at a single node, and terminate.

When agents move completely asynchronously, gathering with crash faults of any type is impossible. This can be seen already in a 3-node clique, each of whose nodes is initially occupied by an agent. Suppose that the adversary lets move agent a at speed 1 but keeps agents b and c at their initial nodes until agent a terminates. Since each of the agents b and c may have crashed after the last visit of it by agent a, at some time agent a must stop at some node and terminate, i.e., declare that all fault-free agents have met. This node is different from the starting node of b or of c. At this time the adversary declares that agents b and c are fault-free and lets them go, which makes gathering incorrect, as one of the agents b or c is not with a at the termination time.

Hence we consider a restricted version of asynchrony, similar to the one in [17], where each agent is assigned by the adversary a fixed speed, possibly different for each agent. Agents have clocks ticking at the same rate. Each agent can wait for a time of its choice at any node, or decide to traverse an edge, but then it moves at constant speed assigned to it. Each agent knows its label and speed but not those of other agents.

1.2   The model and the problem

The network is modeled as a simple undirected connected graph. As it is often done in the literature on rendezvous and gathering, we seek gathering algorithms that do not rely on the knowledge of node labels, and can work in anonymous graphs as well. The importance of designing such algorithms is motivated by the fact that agents may be unable to perceive labels of nodes, even when such labels exist, because of limitations of perception faculties of the agents, or nodes may refuse to reveal their labels, e.g., due to security or privacy reasons. It should be also noted that, if nodes had distinct labels, agents might explore the graph and meet in the smallest node, hence gathering would reduce to graph exploration. On the other hand, we make the assumption, standard in the literature of the domain, that edges incident to a node v have distinct labels in $\{0, \ldots, d-1\}$, where d is the degree of v. Thus every undirected edge $\{u, v\}$ has two labels, which are called its *port numbers* at u and at v. Port numbering is *local*, i.e., there is no relation between port numbers at u and at v. Note that in the absence of port numbers, edges incident to a node would be undistinguishable for agents and thus gathering would be often impossible, as the adversary could prevent an agent from taking some edge incident to the current node. Also, the above mentioned security and privacy reasons that may prevent nodes from revealing their labels, do not apply to port numbers.

Agents start from different nodes of the graph. They cannot mark visited nodes or traversed edges in any way. The adversary wakes up some of the agents at possibly different times. A dormant agent is woken up by the first agent that visits its starting node. Agents do not know the topology of the graph nor any bound on its size. They do not know the total number of agents. They have clocks ticking at the same rate. The clock of each agent starts at its wakeup, and at this time the agent starts executing the algorithm.

The adversary assigns two numbers to each agent: a positive integer called its *label* and a positive real called its *speed*. All labels are different, and if the agent has speed α, it traverses each edge in time $1/\alpha$. (We may represent edges as arcs of length 1 between points representing nodes.) Each agent knows its label and its speed, but not those of other agents. During the execution of an algorithm, an agent can wait at the currently visited node for a time of its choice, or it may choose a port to traverse the corresponding edge. In the latter case, the agent traverses this edge at the constant speed α assigned to it, thus getting to the other end of the edge after time $1/\alpha$.

When an agent enters a node, it learns its degree and the port of entry. When two or more agents are at the same point in the graph (either at a node or inside an edge) at the same time, they notice this fact and can exchange all information that is currently in their memory. We assume that the memory of the agents is unlimited: from the computational point of view they are modeled as Turing machines.

Agents are subject to *crash faults* which can occur at any time. We consider two fault scenarios. A *motion fault* immobilizes the agent at a node or inside an edge but leaves intact its memory at the time when the fault occurred. A more severe *total fault* immobilizes the agent as well, but also erases its entire memory. An agent A meeting an agent B which incurred any crash fault knows that B crashed. In the case of a motion fault of the crashed agent, A can read the entire memory

of B frozen at the time of the crash. By contrast, an agent meeting an agent which incurred a total fault cannot learn anything about it, not even its label. An agent that never crashes during the execution of the gathering algorithm is called *good*.

The task of gathering is defined as follows. All good agents have to be at the same node and terminate, i.e., declare that all good agents are at this node. (Since good agents located at the same node can mutually read their memory, once one of them declares termination, all others can do it as well, so we may guarantee that the declaration is simultaneous.) The aim of this paper is the construction of deterministic gathering algorithms under both considered fault scenarios.

We assume that there is at least one good agent, otherwise gathering of good agents is accomplished vacuously. The time of gathering is counted since the wakeup of the first good agent. Note that counting time since the wakeup of the earliest agent does not make sense, because the adversary can wake up an agent, crash it immediately, and then wait an arbitrarily long time before waking up the second agent.

1.3   Our results

We construct a gathering algorithm working for any team of at least two agents in the scenario of motion faults, and a gathering algorithm working in the presence of total faults, provided that at least two agents are good. If only one agent is good, the task of gathering with total faults is sometimes impossible.

The main challenge of our scenario is that agents may have different speeds, and at the same time they are subject to crash faults. To the best of our knowledge, these two difficulties together were never considered before in the context of gathering in networks: in [20], asynchronous rendezvous was considered but only for functional agents, while in [10, 19], agents with Byzantine faults were considered but only in the fully synchronous scenario, and [11] considered rendezvous of two synchronous agents subject to transient delay faults.

In order to overcome this double difficulty, we adopt a combination of event-driven and time-driven approach. At early stages of the gathering process, agents do not know any upper bound on the size of the graph, and hence their decisions cannot rely on time-outs and must be event driven, i.e., they depend on meeting another agent with a specific memory content. By contrast, due to our algorithm design, towards the end of gathering, when an agent is already at its final destination, it knows an upper bound on the size of the graph, and it knows everything about other good agents, hence it may compute a time bound after which all good agents must get together.

While our main focus is feasibility, and we do not attempt time optimization, both our algorithms work in time polynomial in the size of the graph, in the logarithm of the largest label, in the inverse of the smallest speed, and in the ratio between the largest and the smallest speed.

1.4 Related work

The rendezvous problem for two agents was first introduced in [30]. An excellent survey of randomized rendezvous in various scenarios can be found in [4], cf. also [2, 3, 5, 8, 24]. Deterministic rendezvous in networks was surveyed in [28]. In many papers rendezvous was considered in a geometric setting: an interval of the real line, see, e.g., [8, 9, 23], or the plane, see, e.g., [6, 7]).

For the problem of deterministic rendezvous in networks, modeled as undirected graphs, attention concentrated on the study of the feasibility of rendezvous, and on the time required to achieve this task, when feasible. For example, deterministic rendezvous with agents equipped with tokens used to mark nodes was considered, e.g., in [27]. Deterministic rendezvous of two agents that cannot mark nodes but have unique labels was discussed in [16, 25, 31]. All these papers were concerned with the time of rendezvous in arbitrary graphs, under the synchronous scenario. In [16] the authors showed a rendezvous algorithm polynomial in the size of the graph, in the length of the shorter label and in the delay between the starting time of the agents. In [25, 31] rendezvous time was polynomial in the first two of these parameters and independent of the delay.

Memory required by the agents to achieve deterministic rendezvous was studied in [22] for trees and in [13] for general graphs. Memory needed for randomized rendezvous in the ring was discussed, e.g., in [26].

Apart from the synchronous model used in the above papers, several authors investigated asynchronous rendezvous in the plane [12, 21] and in network environments [14, 15, 17]. In the latter scenario the agent chooses the edge which it decides to traverse but the adversary controls the speed of the agent. Under this assumption rendezvous in a node cannot be guaranteed even in very simple graphs, and hence the rendezvous requirement is relaxed to permit the agents to meet inside an edge. In particular, the main result of [20] is an asynchronous rendezvous algorithm working in an arbitrary graph at cost (measured by the number of edge traversals) polynomial in the size of the graph and in the logarithm of the smaller label. The scenario of possibly different fixed speeds of the agents, that we use in the present paper, was introduced in [17], but agents were assumed fault free.

Gathering more than two agents was studied in the plane [1, 12, 17, 21], and in networks [18, 19, 32]. In [18] agents were assumed anonymous (and hence gathering was not always possible), and in [19, 32] they were equipped with distinct labels.

Gathering many agents subject to faults was considered in [1] in the scenario of anonymous agents in the plane, and in [10, 11, 19] in graphs. In [1] agents were anonymous but the authors assumed their powerful sensorial capabilities: an agent could see all the other agents in the plane. In [10, 11, 19] the synchronous model was used. In [11] the faults were assumed relatively benign and transient: the adversary could delay the move of an agent for a few rounds. By contrast, the faults in [10, 19] were of the most severe type, they were Byzantine, and hence, in order to achieve gathering, some restrictive upper bound on the number of Byzantine agents had to be imposed. To the best of our knowledge, the problem of gathering in networks, under the challenging scenario when agents are not synchronous, and are at the same time subject to permanent faults, was never considered

before.

2   Preliminaries

Throughout the paper, the number of nodes of a graph is called its size and is denoted by n. In this section we present three procedures from the literature, that will be used (after suitable modifications) as building blocks in our algorithms. The aim of the first two procedures is graph exploration, i.e., visiting all nodes and traversing all edges of the graph by a single agent. The first procedure, based on universal exploration sequences (UXS), is a corollary of the result of Reingold [29]. Given any positive integer n, it allows the agent to traverse all edges of any graph of size at most n, starting from any node of this graph, using at most $P(n)$ edge traversals, where P is some non-decreasing polynomial. (The original procedure of Reingold only visits all nodes, but it can be transformed to traverse all edges by visiting all neighbors of each visited node before going to the next node.) After entering a node of degree d by some port p, the agent can compute the port q by which it has to exit; more precisely $q = (p + x_i) \mod d$, where x_i is the corresponding term of the UXS.

A *trajectory* is a sequence of nodes of a graph, in which each node is adjacent to the preceding one. Given any starting node v, we denote by $R(n, v)$ the trajectory obtained by Reingold's procedure. The procedure can be applied in any graph starting at any node, giving some trajectory. We say that the agent *follows* a trajectory if it executes the above procedure used to construct it. This trajectory will be called *integral*, if the corresponding route covers all edges of the graph. By definition, the trajectory $R(n, v)$ is integral, if it is obtained by Reingold's procedure applied in any graph of size at most n, starting at any node v.

The second procedure, described in [20], allows an agent to traverse all edges and visit all nodes of any graph of size at most k provided that there is a unique token located on an *extended edge* $u - v$ (for some adjacent nodes u and v) and authorized to move arbitrarily on it. An extended edge is defined as the edge $u - v$ augmented by nodes u and v. (It is well known that a terminating exploration, even of all anonymous rings of unknown size by a single agent without a token, is impossible.) In our applications the roles of the token and of the exploring agent will be played by agents. This procedure was called $ESST$ in [20], for *exploration with a semi-stationary token*, as the token always remains on the same extended edge (even if it can move arbitrarily inside it). For completeness, we quote verbatim the description of the procedure from [20]. The following notion is used in this description. Let m be a positive integer. An application of $R(2m, u)$ to a graph G at some node u is called *clean*, if all nodes in this application are of degree at most $m - 1$.

Procedure $ESST$

The procedure proceeds in phases $i = 3, 6, 9, 12 \ldots$. In any phase i, the agent first applies $R(2i, v)$ at the node v in which it started this phase. Let $(u_1, \ldots, u_{r+1})$ be the trajectory $R(2i, v)$ ($v = u_1$ and $r = P(2i)$). Call this trajectory the trunc of this phase. If it is not clean, or if no token is seen, the agent aborts phase i and starts phase $i + 3$. Otherwise, the agent backtracks to u_1, and

applies $R(i, u_j)$ at each node u_j of the trunc, interrupting a given execution of $R(i, u_j)$ when it sees a token, every time recording the *code* of the path from u_j to this token. This code is defined as the sequence of ports encountered while walking along the path. (If, for some j, the token is at u_j, then this code is an empty sequence.) After seeing a token, the agent backtracks to u_j, goes to u_{j+1} and starts executing $R(i, u_{j+1})$. For each node u_j of the trunc, if at the end of $R(i, u_j)$ either no token is seen during the execution of $R(i, u_j)$, or the agent has recorded at least $\frac{i}{3}$ different codes in phase i, then the agent aborts phase i and starts phase $i + 3$ (in the special case where the agent decides to abort phase i while traversing an edge, phase $i + 3$ starts at the end of this edge traversal). Otherwise, upon completion of phase i, it stops as soon as it is at a node.

The following theorem was proved in [20]:

Theorem 2.1 *There exists a non-decreasing polynomial ρ such that procedure ESST terminates in every graph G of size n after $\rho(n)$ steps. Upon its termination, all edges of G are traversed by the agent.*

We will use the following corollary of the above theorem.

Corollary 2.1 *Consider the procedure ESST performed in some n-node graph G. There exists a polynomial $\pi(n)$ and integers π_i, for all $i \leq \pi(n)$, such that the number of phases of ESST is at most $\pi(n)$, the number of steps in phase i is at most π_i, and all numbers π_i are polynomial in n.*

The third procedure, that will be called *Meeting*, is the main rendezvous algorithm from [20]. It works for an arbitrary graph of unknown size, under the general asynchronous model, where, in consecutive steps, the agent chooses the port number that it decides to use, and the adversary produces any continuous movement of the agent on the respective edge, subject only to the constraint that it ends at the other endpoint of the chosen edge.

Procedure *Meeting* guarantees the meeting of two agents with distinct labels at cost polynomial in the size of the graph and in the logarithm of the smaller label. The following theorem was proved in [20]:

Theorem 2.2 *There exists a polynomial $\Pi(x, y)$, non decreasing in each variable, such that if two agents with different labels L_1 and L_2 execute Procedure Meeting in a graph of size n, then their meeting is guaranteed by the time one of them performs $\Pi(n, \min(\log L_1, \log L_2))$ edge traversals.*

The above theorem guarantees that two agents with different labels L_1 and L_2 executing Procedure Meeting under our present scenario will meet after a total of $\Pi(n, \min(\log L_1, \log L_2))$ edge traversals, even if one of them crashes during the execution of this procedure. This is due to the fact that a walk of an agent under our scenario (at any constant speed and with a possible crash) is an admissible behavior of the agent under the scenario from [20], where the adversary can hold an agent idle for an arbitrary finite time.

We will use the following notation. For a positive integer x, by $|x|$ we denote the length of its binary representation, called the length of x. Hence $|x| = \lfloor \log x \rfloor + 1$. All logarithms are with base 2. For two agents, we say that the agent with larger (smaller) label is larger (resp. smaller). The length (i.e., the logarithm) of the largest label is denoted by Λ, the smallest speed of any agent is denoted by ϵ, and the largest speed of any agent is denoted by κ. For any trajectory $T = (v_0, \ldots, v_r)$, we denote by $\overline{T}$ the reverse trajectory $(v_r, \ldots, v_0)$.

3   Gathering with motion faults

In this section we construct our gathering algorithm working in the presence of motion faults. The algorithm works for any team of at least two agents, at least one of which is good. If the team may consist of a single agent that is good, gathering with termination is impossible, e.g., in rings: a single agent would have to stop and declare termination at some point, while there could be another good agent far away in the ring, not being able to affect the history of the first agent; hence in the latter scenario, the first agent would incorrectly declare gathering with termination as well.

We first describe the high-level idea of the algorithm, skipping the details, and then proceed with the detailed description.. To facilitate the exposition, we introduce six states in which an agent can be: `cruiser`, `explorer`, `token`, `finder`, `recycled explorer` and `gatherer`, and describe the goals that an agent achieves in each of these states.

Algorithm `Gathering with motion faults`

An agent starts executing the algorithm in state `cruiser`. The goal in this state is to meet another agent, using procedure *Meeting*. After meeting another agent in state `cruiser`, agents break symmetry using their labels: the larger agent becomes an `explorer`, the smaller becomes a `token`. The `token` stays put, if they met at a node, and it continues the current edge traversal and waits at its endpoint, if they met inside an edge. The `explorer` tries to find an upper bound on the size of the graph. This is done using a modification of procedure $ESST$, where the agent in state `token` is used as the semi-stationary token. However, a crucial change to this procedure has to be made. It must be modified in a way to permit the `token` to realize that the `explorer` crashed, if this is the case. To this end, after each phase of $ESST$, the `explorer` returns to the `token`, which permits the latter to use a time-out mechanism (based on the `explorer`'s speed that the `token` learned) that would signal a crash of the `explorer`. If the `token` realizes that the `explorer` crashed, it transits to state `finder` and tries to meet the crashed `explorer`. It then transits to state `recycled explorer`, performs $ESST$ with this crashed `explorer` as token, in order to learn the size of the graph.

After learning the size of the graph, an agent transits to state `gatherer`. In this state, an agent uses procedure *Meeting* to meet all other agents, both good and crashed. It uses the upper bound on the size of the graph to estimate the time after which it can be certain to have met all other agents. This is not straightforward, as the agent does not yet know the speeds of all agents, and it has to establish the sufficient time of performing procedure *Meeting* on the basis of currently available information. When meeting another agent, a `gatherer` records its label, speed and a path

towards the starting node of this agent. After meeting all agents, a `gatherer` goes to the starting node of the smallest agent. Then it uses a time-out mechanism, based on the knowledge of speeds of the other agents. It knows that until some time bound, all other good agents should also be at this node. Then the agent terminates, declaring that gathering of all good agents is accomplished.

The main difficulty of the algorithm design is a combination of event driven and time driven approach. Since at various stages of the algorithm execution the agent has still incomplete knowledge about the graph and about other agents, it cannot rely on time-outs at the early stages. For example, before transiting to state `gatherer` the agent does not know any upper bound on the size of the graph, and hence all transitions must be event driven, e.g., meeting another agent in a given state. By contrast, towards the end of the execution, when the agent is already at its final destination, it knows an upper bound on the size of the graph, and it knows everything about other good agents, hence it may compute a time bound after which all good agents must get together.

We now give a detailed description of the algorithm, explaining the actions of the agents in each of the six states. In the following description, the executing agent is called A, has label L and speed θ. We assume that at every meeting, agents mutually learn their memory states. This is for convenience only: it will be seen that, in fact, much less information needs to be transferred.

State `cruiser`

Upon wakeup, agent A is in this state. It performs procedure *Meeting* until one of the following four events happens:

1. it meets another agent in state `cruiser`,

2. it meets a crashed agent B in state `token`,

3. it meets an agent C in state `finder` or `recycled explorer`

4. it meets an agent B in state `gatherer`.

In the first case, let $L_1, \ldots, L_k$ be the labels of agents in state `cruiser` that A met in the first meeting (simultaneous meeting of many agents is possible). Let $L' = \min(L, L_1, \ldots, L_k)$. If $L = L'$, agent A transits to state `token`, otherwise agent A transits to state `explorer`, and considers the agent with label L' as its token.

In the second case, agent A transits to state `explorer`, and considers agent B as its token.

In the third case, agent A learns from C the label of the culprit B of C (see the description of state `token`). Agent A adopts B as its culprit and transits to state `finder`.

In the fourth case, agent A completes the currently executed edge traversal, adopts the upper bound m on the size of the graph from agent B, and transits to state `gatherer`.

State `explorer`

Agent A executes the modified procedure $ESST$, starting at the node of meeting with its token, or, if the meeting occurred inside an edge, completing the current traversal and starting at the cor-

responding endpoint. The procedure is modified in the following way: after each phase (completed or aborted) agent A backtracks to its token following the trajectory $\overline{T}$, where T is the trajectory followed in this phase of $ESST$. Thus the number of phases in the modified Procedure $ESST$ is the same as in $ESST$, and phase i of the modified procedure takes $2\pi_i$ steps (cf. Corollary 2.1). Upon completion of the procedure, the agent traversed all edges of the graph, knows it, and adopts the total number of traversals m as an upper bound on the size of the graph. Agent A gives the integer m to its token and transits to state `gatherer`.

State `token`

Agent A considers all agents that it met when they were in state `cruiser` at the moment of the first meeting (and that transited to state `explorer` upon this meeting) as its `explorers`. Agent A continues its current edge traversal and waits until one of the following two events happens for each of its `explorers`. Either an `explorer` completes the modified procedure $ESST$ giving agent A an upper bound m on the size of the graph, or an `explorer` B is not back after the scheduled time at most $2\pi_i/\beta$ in some phase i, where β is the speed of B (learned by A). Notice that there may be potentially two different upper bounds given to A by `explorers` that successfully completed the modified procedure $ESST$ because these `explorers` may start at different endpoints of the edge where they met their `token`, if the meeting occurred inside an edge. When one of the above events occurs for every `explorer` of the agent A, there are two cases. The first case is when at least one `explorer` managed to complete the modified procedure $ESST$ and gave an upper bound to A. In this case, agent A adopts the minimum of the (possibly two) obtained upper bounds as its upper bound and transits to state `gatherer`. The second case is when all `explorers` of A failed to successfully complete the procedure (because they all crashed). Agent A learns this and adopts the (crashed) explorer B with the smallest label among all its explorers as its *culprit*. Agent A transits to state `finder`.

State `finder`

Agent A continues its current edge traversal to the endpoint v_1 (if it is at a node v_1 when transiting to state `finder`, it skips this action). Then A executes procedures $R(i, v_i)$, for $i = 1, 2, ...$, one after another, where v_{i+1} is the node where $R(i, v_i)$ ended, until it finds its culprit. Agent A transits to state `recycled explorer`.

State `recycled explorer`

Agent A executes procedure $ESST$ with its culprit as token. (Here there is no need to modify procedure $ESST$ because the culprit is crashed anyway, so it could not realize that agent A crashed if this happened). Upon completion of $ESST$, agent A adopts the number of steps in this procedure as an upper bound m on the size of the graph. Agent A transits to state `gatherer`.

State `gatherer`

In this state, agent A knows an upper bound m on the size of the graph. Before describing the actions of the agent in this state, we define two procedures.

Procedure $Ball(w)$

The procedure consists of traversing all paths of length 2, starting at node w, each time returning to node w.

The next procedure uses procedure $Ball$.

Procedure $R^*(m, v)$

Follow the trajectory $R(m, v)$, at each visited node w executing procedure $Ball(w)$ before going to the next node of the trajectory.

In state `gatherer`, agent A starts by executing the procedure $R^*(m, v)$, where v is the node in which A transited to this state. Recall that θ is the speed of the agent, and let c be the number of edge traversals performed by agent A in the execution of this procedure. Let $\gamma = \frac{\theta}{c}$. We will show that, upon completion of $R^*(m, v)$, agent A has achieved the following two goals: it met all agents whose speed is less than γ, and it learned that all agents are awake. It also follows that at this time, agent A learned the lower bound $\delta = \min(\gamma, \epsilon)$ on the speeds of all agents, where ϵ is the minimum speed of all agents.

Next, agent A executes procedure $Meeting$ for time $\tau = 2\Pi(\nu, \log L)/\delta + 3\rho(\nu)/\delta + \sum_{i=1}^{\nu} P(i)/\delta + \nu^2 P(\nu)/\delta$, where $\nu = 2\rho(m)$. Call this period of time the *red* period of agent A.

We will show that after this time, agent A has met all the agents, and hence it knows their labels, their speeds, and the path to the starting node of each of them. Moreover, it knows the largest upper bound μ on the size of the graph, among those adopted by all the agents. Agent A goes to the starting node of the agent whose label is the smallest, and starts its final waiting period.

Let $\nu^* = 2\rho(\mu)$. ν^* is an upper bound on the values of ν calculated by all agents. Let δ^* be the minimum of the values of δ calculated by all agents. Let $\phi = 21\rho(\nu^*)/\delta^* + 7\sum_{i=1}^{\nu^*} P(i)/\delta^* + 8\Pi(\nu^*, \Lambda)/\delta^* + 8(\nu^*)^2 P(\nu^*)/\delta^*$. We will show that the value ϕ is an upper bound on the termination of the red periods of all agents.

The final waiting period of agent A has length $\tau' = \phi + 2\phi\kappa/\epsilon$.

Notice that this final waiting period has the same length for all agents, as it does not depend on the label and the speed of an individual agent. Agent A can calculate this waiting time because, as we will show, when it starts it, it knows all parameters concerning other agents.

We will prove that after this time all good agents must also be at the starting node of the smallest agent, and that agent A learns this fact. After waiting time τ' (or at the time when some other agent declares gathering, if this happens earlier), agent A declares that all good agents are gathered, and terminates.

The proof of correctness and the performance analysis of Algorithm `Gathering with motion faults` are split into a series of lemmas.

Lemma 3.1 *By the time $\Pi(n, \Lambda)/\epsilon$ some agents must meet.*

Proof. Let D be the earliest woken good agent (or one of them, if there are several such agents),

10

and let L_D be its label. By Theorem 2.2, D must meet some agent by the time when D performs $\Pi(n, \log L_D)$ edge traversals executing procedure *Meeting* in state `cruiser`, if no other agents have met before. This must happen by the time $\Pi(n, \log L_D)/\epsilon \leq \Pi(n, \Lambda)/\epsilon$. $\qquad \square$

Let S_0 be the earliest woken good agent. (If there are more such agents, we denote by S_0 any of them.) By definition, time is counted since the wakeup of agent S_0, i.e., its wakeup time is set to 0.

Lemma 3.2 *By the time* $6\rho(n)/\epsilon + 2\sum_{i=1}^{n} P(i)/\epsilon + 2\Pi(n, \Lambda)/\epsilon$ *some agent transits to state* `gatherer`.

Proof. Let t_1 be the time when the first meeting occurred. This was necessarily a meeting of agents in state `cruiser`. First suppose that one of these agents, call it F, was good. If F transits to state `explorer` upon this meeting, then by the time $t_1 + 2\rho(n)/\epsilon$ agent F completes the modified procedure $ESST$, and transits to state `gatherer`. If F transits to state `token` upon this meeting, then by the time $t_1 + 2\rho(n)/\epsilon$ agent F either learns an upper bound on the size of the graph (because some of its `explorers` completed the modified procedure $ESST$) and transits to state `gatherer`, or it learns that all its `explorers` crashed. In the latter case, agent F transits to state `finder`, finds its culprit B within an additional time $\sum_{i=1}^{n} P(i)/\epsilon$ (because within this time it traverses all edges of the graph), transits to state `recycled explorer`, then within an additional time $\rho(n)/\epsilon$, it completes procedure $ESST$ with B as token, learns an upper bound on the size of the graph, and transits to state `gatherer`. Hence, if one of the agents participating in the first meeting was good, then this agent must transit to state `gatherer` by the time $t_1 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon$.

It remains to consider the case, when none of the agents participating in the first meeting was good. If the agent T that transited to state `token` at the first meeting did not crash by the time $t_1 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon$, then the above argument shows that it transits to state `gatherer` by this time. Hence we may assume that this agent crashed by the time $t_1 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon$.

Let t_2 be the time when agent S_0 transits from state `cruiser` to some other state. This happened because of one of the four events in the definition of state `cruiser`.

If in time t_2 agent S_0 met another agent in state `cruiser`, then it transited to state `gatherer` by the time $t_2 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon$, in view of the argument above.

If in time t_2 agent S_0 met a crashed agent in state `token`, then it transited to state `explorer`, and by the time $t_2 + 2\rho(n)/\epsilon$ it completes the modified procedure $ESST$, and transits to state `gatherer`.

If in time t_2 agent S_0 met an agent C in state `finder` or `recycled explorer` then it learns from C the label of the culprit B of C, adopts B as its culprit and transits to state `finder`. It finds its culprit B within an additional time $\sum_{i=1}^{n} P(i)/\epsilon$, transits to state `recycled explorer`, then within an additional time $\rho(n)/\epsilon$ it completes procedure $ESST$ with B as token, learns an upper bound on the size of the graph, and transits to state `gatherer` by the time $t_2 + \rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon$.

If in time t_2 agent S_0 met an agent in state `gatherer`, then it learns from it an upper bound on the size of the graph, and immediately transits to state `gatherer`.

11

Hence, in all cases, agent S_0 transits to state gatherer by the time $t_2 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon$. We now estimate the time t_2 when agent S_0 transits from state `cruiser` to some other state. By our assumption, agent T crashed by the time $t_1 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon$. In the moment of crashing, it could be either still in the state `token`, or in state `finder` or in state `recycled explorer` (otherwise, it would transit to state `gatherer` before crashing, and the lemma is proved). Consider the time $t_1 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon + \Pi(n, \Lambda)/\epsilon$. If by this time agent S_0 is still in state `cruiser`, then by this time it must meet agent T in one of these states because S_0 performed at least $\Pi(n, \Lambda)$ edge traversals executing procedure $Meeting$, while T was still. Hence S_0 must transit from state `cruiser` to some other state by this time. It follows that $t_2 \leq t_1 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon + \Pi(n, \Lambda)/\epsilon$.

Finally, we have $t_1 \leq \Pi(n, \Lambda)/\epsilon$, by Lemma 3.1. We conclude that some agent transits to state `gatherer` by the time $t_2 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon \leq t_1 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon + \Pi(n, \Lambda)/\epsilon + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon \leq 6\rho(n)/\epsilon + 2\sum_{i=1}^{n} P(i)/\epsilon + 2\Pi(n, \Lambda)/\epsilon$. $\qquad\square$

Let m be the upper bound on the size of the graph, adopted by the first agent that transited to state `gatherer`.

Lemma 3.3 *Let S be a good agent, and let t be its waking time. By the time $t + 9\rho(n)/\epsilon + 3\sum_{i=1}^{n} P(i)/\epsilon + 3\Pi(n, \Lambda)/\epsilon + 2m^2 P(m)/\epsilon$ agent S transits to state* `gatherer`.

Proof. Let t' be the time when the first agent, call it H, transits to state `gatherer`. In view of Lemma 3.2, we have $t' \leq 6\rho(n)/\epsilon + 2\sum_{i=1}^{n} P(i)/\epsilon + 2\Pi(n, \Lambda)/\epsilon$. Let m be the upper bound on the size of the graph, adopted by agent H. In the execution of procedure $R^*(m, v)$ agent H performed $c \leq 2m^2 P(m)$ edge traversals. Hence by the time $t' + 2m^2 P(m)/\epsilon$, it completed procedure $R^*(m, v)$ and started procedure $Meeting$, if it did not crash before. Hence agent S must transit from state `cruiser` to some other state by the time $t + t' + 2m^2 P(m)/\epsilon + \Pi(n, \Lambda)/\epsilon$. Indeed, if it did not transit by the time $t + t' + 2m^2 P(m)/\epsilon + \Pi(n, \Lambda)/\epsilon$, then it performed at least $\Pi(n, \Lambda)$ edge traversals executing procedure $Meeting$ while agent H was also executing this procedure, or else H crashed in the meantime. Hence S must have met H and transited to state `gatherer`.

Let t_2 be the time when agent S transits from state `cruiser` to some other state. It follows from the above that $t_2 \leq t + t' + 2m^2 P(m)/\epsilon + \Pi(n, \Lambda)/\epsilon$. By the argument from the proof of Lemma 3.2, agent S must transit to state `gatherer` by the time $t_2 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon$. Hence agent S must transit to state `gatherer` by the time

$$t_2 + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon \leq$$

$$t + t' + 2m^2 P(m)/\epsilon + \Pi(n, \Lambda)/\epsilon + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon \leq$$

$$t + 6\rho(n)/\epsilon + 2\sum_{i=1}^{n} P(i)/\epsilon + 2\Pi(n, \Lambda)/\epsilon + 2m^2 P(m)/\epsilon + \Pi(n, \Lambda)/\epsilon + 3\rho(n)/\epsilon + \sum_{i=1}^{n} P(i)/\epsilon =$$

$$t + 9\rho(n)/\epsilon + 3\sum_{i=1}^{n} P(i)/\epsilon + 3\Pi(n,\Lambda)/\epsilon + 2m^2 P(m)/\epsilon.$$

$\square$

Let μ be the largest of the upper bounds on the size of the graph adopted by any agent.

Lemma 3.4 *By the time $9\rho(n)/\epsilon + 3\sum_{i=1}^{n} P(i)/\epsilon + 3\Pi(n,\Lambda)/\epsilon + 4\mu^2 P(\mu)/\epsilon$ all agents are woken up.*

Proof. In view of Lemma 3.3, agent S_0 transits to state `gatherer` by the time $9\rho(n)/\epsilon + 3\sum_{i=1}^{n} P(i)/\epsilon + 3\Pi(n,\Lambda)/\epsilon + 2\mu^2 P(\mu)/\epsilon$. Hence, by the time $9\rho(n)/\epsilon + 3\sum_{i=1}^{n} P(i)/\epsilon + 3\Pi(n,\Lambda)/\epsilon + 4\mu^2 P(\mu)/\epsilon$, agent S_0 completed procedure $R^*(m_1, v)$, where v is the node at which S_0 transited to state `gatherer`. Hence it traversed all edges of the graph, an thus all agents are woken up by this time. $\square$

Lemma 3.5 *All good agents transit to state `gatherer` by the time $18\rho(n)/\epsilon + 6\sum_{i=1}^{n} P(i)/\epsilon + 6\Pi(n,\Lambda)/\epsilon + 6\mu^2 P(\mu)/\epsilon$.*

Proof. Let S be a good agent and let t be its waking time. By Lemma 3.3, S transits to state `gatherer` by the time $t + 9\rho(n)/\epsilon + 3\sum_{i=1}^{n} P(i)/\epsilon + 3\Pi(n,\Lambda)/\epsilon + 2\mu^2 P(\mu)/\epsilon$. By Lemma 3.4, $t \leq 9\rho(n)/\epsilon + 3\sum_{i=1}^{n} P(i)/\epsilon + 3\Pi(n,\Lambda)/\epsilon + 4\mu^2 P(\mu)/\epsilon$. Hence S transits to state `gatherer` by the time $18\rho(n)/\epsilon + 6\sum_{i=1}^{n} P(i)/\epsilon + 6\Pi(n,\Lambda)/\epsilon + 6\mu^2 P(\mu)/\epsilon$. $\square$

Lemma 3.6 *Any good agent S meets all other agents by the time it executes its red period. This happens by the time $21\rho(\nu)/\delta + 7\sum_{i=1}^{\nu} P(i)/\delta + 8\Pi(\nu,\Lambda)/\delta + 8\nu^2 P(\nu)/\delta$.*

Proof. Let $m(S)$ be the upper bound on the size of the graph, adopted by agent S, let θ be the speed of this agent, and let L be its label. After executing procedure $R^*(m(S), v)$, where v is the node where S transited to state `gatherer`, agent S traversed all edges, and hence it knows that all other agents are woken up. During the execution of $R^*(m, v)$ agent S makes $c \leq 2m(S)^2 P(m(S))$ edge traversals. Also, during this execution, agent S traverses all directed paths of length 2 in the graph. Let $\gamma = \frac{\theta}{c}$, and consider an agent S' whose speed is less than γ. The time c/θ during which agent S makes the c edge traversals executing procedure $R^*(m(S), v)$ is too short for agent S' to make even one full edge traversal. Hence, regardless of the trajectory followed by agent S', it can be in at most two incident edges during this time. Since agent S traverses all such couples of edges, it must meet agent S' during the execution of $R^*(m(S), v)$. It follows that after the completion of procedure $R^*(m(S), v)$, agent S either learns the minimum speed ϵ of all agents (if there exists an agent with speed less than γ) or it learns that γ is a lower bound on the speed of all the agents (if

there is no agent with speed less than γ). Set δ to ϵ in the first case and to γ in the second case. Notice that, upon completion of procedure $R^*(m(S), v)$, agent S learns the value of δ.

Consider any good agent S'', with label L'', whose speed is at least γ. Upon completion of procedure $R^*(m(S), v)$ at time t_1, agent S executes procedure $Meeting$ and agent S'' is already woken up. We will prove that agent S meets agent S'' by the time $t_1 + \tau$. We may assume that by the time $t_2 = t_1 + \Pi(m(S), \log L)/\delta$, agent S'' transited from state $\texttt{cruiser}$ to some other state, for otherwise it would execute procedure $Meeting$ during the time when S performed at least $\Pi(m(S), \log L)$ edge traversals executing this procedure, and hence the agents would meet, in view of Theorem 2.2. This transition of agent S'' happened because of one of the four events in the definition of state $\texttt{cruiser}$.

If agent S'' met another agent in state $\texttt{cruiser}$, then it either transited to state $\texttt{explorer}$ or to state $\texttt{token}$. If S'' transits to state $\texttt{explorer}$ upon this meeting, then by the time $t_2 + 2\rho(n)/\gamma$ agent S'' completes the modified procedure $ESST$, and transits to state $\texttt{gatherer}$. If S'' transits to state $\texttt{token}$ upon this meeting, then by the time $t_2 + 2\rho(n)/\gamma$ agent S'' either learns an upper bound on the size of the graph (because some of its $\texttt{explorers}$ completed the modified procedure $ESST$) and transits to state $\texttt{gatherer}$, or it learns that all its $\texttt{explorers}$ crashed. In the latter case, agent S'' transits to state $\texttt{finder}$, finds its culprit B within an additional time $\sum_{i=1}^{n} P(i)/\gamma$ (because within this time it traverses all edges of the graph), transits to state $\texttt{recycled explorer}$, then within an additional time $\rho(n)/\gamma$, it completes procedure $ESST$ with B as token, learns an upper bound on the size of the graph, and transits to state $\texttt{gatherer}$. Hence, if agent S'' met an agent in state $\texttt{cruiser}$, then S'' must transit to state $\texttt{gatherer}$ by the time $t_2 + 3\rho(n)/\gamma + \sum_{i=1}^{n} P(i)/\gamma$.

If agent S'' met a crashed agent in state $\texttt{token}$, then it transited to state $\texttt{explorer}$, and by the time $t_2 + 2\rho(n)/\gamma$ it completes the modified procedure $ESST$, and transits to state $\texttt{gatherer}$.

If agent S'' met an agent C in state $\texttt{finder}$ or $\texttt{recycled explorer}$ then it learns from C the label of the culprit B of C, adopts B as its culprit and transits to state $\texttt{finder}$. It finds its culprit B within an additional time $\sum_{i=1}^{n} P(i)/\gamma$, transits to state $\texttt{recycled explorer}$, then within an additional time $\rho(n)/\gamma$ it completes procedure $ESST$ with B as token, learns an upper bound on the size of the graph, and transits to state $\texttt{gatherer}$ by the time $t_2 + \rho(n)/\gamma + \sum_{i=1}^{n} P(i)/\gamma$.

If agent S'' met an agent in state $\texttt{gatherer}$, then it learns from it an upper bound on the size of the graph, and immediately transits to state $\texttt{gatherer}$.

Hence, in all cases, agent S'' transits to state $\texttt{gatherer}$ by the time $t_3 = t_2 + 3\rho(n)/\gamma + \sum_{i=1}^{n} P(i)/\gamma \leq t_1 + \Pi(m(S), \log L)/\delta + 3\rho(m(S))/\delta + \sum_{i=1}^{m(S)} P(i)/\delta$.

Let $m(S'')$ be the upper bound on the size of the graph adopted by agent S'' and let v'' be the node at which S'' transited to state $\texttt{gatherer}$. Upon the transition, agent S'' executes procedure $R^*(m(S''), v'')$ within time at most $m(S'')^2 P(m(S''))/\gamma$ and then starts its own red period executing procedure $Meeting$. By the time $t_4 = t_3 + m(S'')^2 P(m(S''))/\gamma + \Pi(m(S), \log L)/\delta$ agent S must meet agent S'' because either S performed at least $\Pi(m(S), \log L)$ edge traversals executing procedure $Meeting$, while S'' also executed procedure $Meeting$, or S'' performed at least $\Pi(m(S), \log L'')$ edge traversals executing procedure $Meeting$, while S also executed procedure $Meeting$.

Hence by the time t_4, agent S must meet every good agent whose speed is at least γ. Note that if agent S'' is not good and (possibly) crashed at some point, its behavior (standing still since the crash) is also a possible behavior of an asynchronous agent during procedure $Meeting$, and hence S must meet every agent whose speed is at least γ by time t_4, if S was still executing procedure $Meeting$.

Since S already met all agents whose speed is less than γ by the time t_1, we conclude that by the time t_4 agent S must meet all agents, if it executed procedure $Meeting$ during its red period for a sufficiently long time.

Consider agent S upon completion of procedure $R^*(m(S), v)$. Agent S knows δ and knows some upper bound on the largest upper bound μ on the size of the graph adopted by any agent. Indeed, any agent T adopts as its upper bound the number of edge traversals in its execution of procedure $ESST$. This number is at most $2\rho(n)$. Hence it is at most $\nu = 2\rho(m(S))$, which S knows. Thus $\mu \le \nu$.

It follows that $t_4 = t_3 + m(S'')^2 P(m(S''))/\gamma + \Pi(m(S), \log L)/\delta \le t_1 + \Pi(m(S), \log L)/\delta + 3\rho(m(S))/\delta + \sum_{i=1}^{m(S)} P(i)/\delta + m(S'')^2 P(m(S''))/\gamma + \Pi(m(S), \log L)/\delta \le t_1 + 2\Pi(\nu, \log L)/\delta + 3\rho(\nu)/\delta + \sum_{i=1}^{\nu} P(i)/\delta + \nu^2 P(\nu)/\delta = t_1 + \tau$.

Hence by the end of its red period in time $t_1 + \tau$, agent S meets all the agents. Since agent S transits to state $\mathtt{gatherer}$ by the time $18\rho(n)/\epsilon + 6\sum_{i=1}^{n} P(i)/\epsilon + 6\Pi(n, \Lambda)/\epsilon + 6\mu^2 P(\mu)/\epsilon$, in view of Lemma 3.5, we have $t_1 \le 18\rho(n)/\epsilon + 6\sum_{i=1}^{n} P(i)/\epsilon + 6\Pi(n, \Lambda)/\epsilon + 6\mu^2 P(\mu)/\epsilon + \nu^2 P(\nu)/\delta \le 18\rho(\nu)/\delta + 6\sum_{i=1}^{\nu} P(i)/\delta + 6\Pi(\nu, \Lambda)/\delta + 7\nu^2 P(\nu)/\delta$.

Since $\tau = 2\Pi(\nu, \log L)/\delta + 3\rho(\nu)/\delta + \sum_{i=1}^{\nu} P(i)/\delta + \nu^2 P(\nu)/\delta$, the end of the red period of agent S occurs by the time $21\rho(\nu)/\delta + 7\sum_{i=1}^{\nu} P(i)/\delta + 8\Pi(\nu, \Lambda)/\delta + 8\nu^2 P(\nu)/\delta$. $\qquad\square$

Let $\nu^* = 2\rho(\mu)$. ν^* is an upper bound on the values of ν calculated by all agents. Let δ^* be the minimum of the values of δ calculated by all agents. Let $\phi = 21\rho(\nu^*)/\delta^* + 7\sum_{i=1}^{\nu^*} P(i)/\delta^* + 8\Pi(\nu^*, \Lambda)/\delta^* + 8(\nu^*)^2 P(\nu^*)/\delta^*$. The value ϕ is an upper bound on the termination of the red periods of all agents.

Lemma 3.7 *By the time $\phi + 2\phi\kappa/\epsilon$ any good agent S is at the starting node of the smallest agent S_{min} and starts its final waiting period.*

Proof. Let T be the trajectory of agent S until the end of its red period, and let T' be the trajectory of agent S_{min} until the first meeting with agent S. The length of each of the trajectories T and T' is at most $\phi\kappa$. After meeting all agents, agent S knows both these trajectories. Hence it can backtrack to the meeting point and then backtrack to the starting node of agent S_{min}. This takes at most $2\phi\kappa$ edge traversals, and hence it takes time at most $2\phi\kappa/\epsilon$. Hence by the time $\phi + 2\phi\kappa/\epsilon$ agent S is at the starting node of the smallest agent and starts its final waiting period. $\square$

Lemma 3.8 *By the time $2\phi + 4\phi\kappa/\epsilon$ all good agents are at the starting node of the smallest agent, all of them know it, and correctly declare that gathering is completed.*

Proof. We first show that at the beginning of its final waiting period, every good agent can compute the length $\tau' = \phi + 2\phi\kappa/\epsilon$ of this period. Indeed this value depends on five parameters: Λ, κ, ϵ, ν^* and δ^*. At the beginning of its final waiting period, every good agent has met all agents and knows all their parameters. Hence it can compute the largest length Λ of all labels, the largest speed κ, and the smallest speed ϵ. It learns all values δ computed by all agents, and finds their minimum δ^*. It learns all values ν computed by all agents, and finds their maximum ν^*. Hence it can compute ϕ and finally τ'.

In view of Lemma 3.7, by the time $\phi + 2\phi\kappa/\epsilon$, every good agent is at the starting node of the smallest agent and starts its final waiting period. Hence at the end of its final waiting period, which has the same length $\tau' = \phi + 2\phi\kappa/\epsilon$, every good agent knows that all good agents started their final waiting period. Hence every good agent correctly declares that gathering is completed by the time $2\phi + 4\phi\kappa/\epsilon$. $\qquad\qquad\square$

We are now ready to prove the main result of this section.

Theorem 3.1 *Algorithm* `Gathering with motion faults` *is a correct algorithm for the task of gathering with motion faults. It works in time polynomial in the size n of the graph, in the length Λ of the largest label, in the inverse of the smallest speed ϵ, and in the ratio r between the largest and the smallest speed.*

Proof. The correctness of the algorithm follows immediately from Lemma 3.8. It remains to verify that the time $2\phi + 4\phi\kappa/\epsilon$ is polynomial in n, Λ, $1/\epsilon$ and r. Since $r = \kappa/\epsilon$, it is enough to verify that $\phi = 21\rho(\nu^*)/\delta^* + 7\sum_{i=1}^{\nu^*} P(i)/\delta^* + 8\Pi(\nu^*, \Lambda)/\delta^* + 8(\nu^*)^2 P(\nu^*)/\delta^*$ is polynomial in n, Λ and $1/\epsilon$.

The upper bound m on the size of the graph, calculated by each agent, is at most $2\rho(n)$, in view of Lemma 2.1. Since ρ is a polynomial, it follows that m is polynomial in n, and so is μ. Hence $\nu^* = 2\rho(\mu)$ is also polynomial in n.

Since the value γ computed by each agent is $\frac{\theta}{c}$, where θ is the speed of the agent and $c \leq \mu^2 P(\mu)$, it follows that $1/\gamma$ is polynomial in n and in $1/\epsilon$. Hence the value $1/\delta$ computed by each agent is polynomial in n and in $1/\epsilon$, and thus so is $1/\delta^*$.

It follows that $21\rho(\nu^*)/\delta^*$ is polynomial in n and in $1/\epsilon$. Since P is a polynomial, it follows that $7\sum_{i=1}^{\nu^*} P(i)/\delta^*$ and $8(\nu^*)^2 P(\nu^*)/\delta^*$ are polynomial in n and in $1/\epsilon$. Since Π is a polynomial, it follows that $8\Pi(\nu^*, \Lambda)/\delta^*$ is polynomial in n, Λ, and $1/\epsilon$. Hence ϕ is polynomial in n Λ, and $1/\epsilon$. $\square$

4 Gathering with total faults

In this section we construct a gathering algorithm in the scenario of total faults, where an agent meeting a crashed agent cannot learn anything about it: all crashed agents look identical. We first show that in the presence of these more severe faults, gathering may be impossible if only one agent is good. (Gathering in this case means that this agent should stop and declare that all good agents are with it, i.e., that it is the only good agent.) This is in contrast with the motion faults scenario, in which our algorithm worked correctly for any team of at least two agents, even if only one agent was good.

Proposition 4.1 *Gathering with total faults cannot be guaranteed if there may be only one good agent.*

Proof. Consider the class of rings of size $4k$, for all positive integers k, with port numbers 0 and 1 at every edge. This port numbering orients the rings. Suppose that there exists an algorithm $\mathcal{A}$ that accomplishes the gathering in this class of rings. In the ring of size 4, the adversary places agents in two antipodal nodes and crashes them immediately. Moreover, it places a good agent in one of the remaining nodes, assigning it the speed 1. If this single good agent executing $\mathcal{A}$ has label 1, it must stop after t_1 steps and declare that it is the unique good agent. If it has label 2, it must stop after t_2 steps and declare that it is the unique good agent.

Consider the ring of size $4(t_1 + t_2)$, in which the adversary places agents at every second node, and crashes them immediately. Moreover, it places two good agents at two antipodal nodes that are not yet occupied, assigns speed 1 to each of them, assigns label 1 to one of them, label 2 to the other, and starts them simultaneously. Consider the agent with label 1 executing the algorithm for t_1 steps, and the agent with label 2 executing the algorithm for t_2 steps. By this time, these agents cannot meet because initially they are at distance $2(t_1 + t_2)$. Hence the history of each of them is the same as in the ring of size 4. Consequently, since the algorithm $\mathcal{A}$ is deterministic, each of them must make the same decision as in the ring of size 4 where it was the unique good agent, i.e., stop and declare gathering (declare that it is the unique good agent). Since in the considered scenario there are at this time two good agents at different nodes, algorithm $\mathcal{A}$ is incorrect. $\square$

Hence, in the rest of this section, we assume that there are at least two good agents. This means that the task of gathering with total faults is on the one hand more difficult because a good agent cannot learn anything from a crashed agent, but on the other hand it is easier because a good agent can rely on the existence of another good agent, and plan its moves accordingly.

As before, we first describe the high-level idea of the algorithm, pointing out the differences with respect to the Algorithm `Gathering with motion faults`, and then proceed with the detailed description. We will now have only four states in which an agent can be: `cruiser`, `explorer`, `token`, and `gatherer`.

Algorithm `Gathering with total faults`

An agent starts executing the algorithm in state `cruiser`. The goal in this state is to meet another agent, using procedure *Meeting*. However, as opposed to the previous algorithm, an agent in state `cruiser` executes this procedure until meeting another agent in state `cruiser` or `gatherer`, ignoring all meetings with agents in states `token` or `explorer` and ignoring meetings with crashed agents. As opposed to the previous scenario, we can afford to ignore these meetings because there are at least two good agents.

After meeting an agent in state `cruiser`, agents break symmetry using their labels: the larger agent becomes an `explorer`, the smaller becomes a `token`. The `explorer` tries to find an upper bound on the size of the graph. This is done using the modified procedure $ESST$, where the agent in state `token` is used as the semi-stationary token. Now there are two modifications of $ESST$. One is as before, returning to the token after each phase, and it is made for the same reason. The second modification is that when the `explorer` meets a crashed agent, it returns to the `token` to verify if this `token` did not crash. This has to be done because all crashed agents look the same, so the `explorer` must be sure that the crashed agent it met is not its own crashed `token`. If the `token` survived until the return, the `explorer` goes again to the place where it interrupted exploration and continues it. Whenever the `explorer` realizes that its `token` crashed, or the `token` realizes that all its `explorers` crashed, it transits back to state `cruiser`. (Now trying to find a crashed `explorer` would be impossible, as it cannot be recognized.) If exploration succeeds, the `explorer` and the `token` learn an upper bound on the size of the graph and transit to state `gatherer`. If a `cruiser` meets a `gatherer`, it adopts its upper bound on the size of the graph and transits to state `gatherer`.

After learning the size of the graph, an agent transits to state `gatherer`. In this state, an agent behaves similarly as in the previous algorithm: it first executes procedure R^* to meet all slow agents, and then procedure *Meeting* to meet the remaining good agents. The latter is done using the learned upper bound on the size of the graph to estimate the time after which the agent can be certain to have met all other good agents. However, there is an additional difficulty to be overcome. At the end of this period of executing *Meeting* in state `gatherer` (we will call it again the red period), an agent must be certain that all good agents identify the same agent at whose starting node they will gather. Now this is more difficult because one agent can meet the agent with the smallest label before it crashed, and another agent may meet it after it crashed, and hence may be unable to learn the smallest label. Hence, the red period of all agents must be sufficiently long to ensure that all agents learned (possibly indirectly) the labels, speeds and routes to the initial positions of the same subset of agents. Then they can correctly choose the same (smallest) agent in the subset (even if by this time it already crashed) and meet at its starting node. The actions of a good agent after its red period are similar as for gathering with motion faults.

We now give a detailed description of the algorithm, explaining the actions of the agents in each of the four states. In the following description, the executing agent is called A, has label L and speed θ.

State `cruiser`

Upon wakeup, agent A is in this state. It performs procedure *Meeting* until it meets another agent

in state `cruiser` or an agent in state `gatherer`.

In the first case, let $L_1, \ldots, L_k$ be the agents in state `cruiser` that A met in the first meeting, and suppose that it did not meet any agent in state `gatherer` during this meeting (simultaneous meeting of many agents is possible). Let $L' = \min(L, L_1, \ldots, L_k)$. If $L = L'$, agent A transits to state `token`, otherwise agent A transits to state `explorer`, and considers the agent with label L' as its token.

In the second case, agent A completes the currently executed edge traversal, adopts the upper bound m on the size of the graph from the agent in state `gatherer`, and transits to state `gatherer`.

State `explorer`

Agent A executes the following procedure $ESST^*$ that results from procedure $ESST$ using the two above mentioned modifications.

Procedure $ESST^*$

The agent executes the consecutive steps of the modified procedure $ESST$, as in Algorithm `Gathering with motion faults`. Call these steps *green*. When the agent makes a green step after which it sees a crashed agent, it goes back to its `token` following the trajectory $\overline{T}$, where T is the trajectory consisting of all green steps made in the current phase until this point. (If the meeting with the crashed agent was inside an edge, agent A completes the current edge traversal and then follows $\overline{T}$). If the `token` is crashed, agent A transits to state `cruiser`. Otherwise it follows trajectory T getting back to the crashed agent. All these steps from the crashed agent to the `token` and back are called *blue*. Then the agent makes the next (green) step of the modified procedure $ESST$, as in Algorithm `Gathering with motion faults`.

Upon completion of procedure $ESST^*$, the agent traversed all edges of the graph, knows it, and adopts the total number of edge traversals m as an upper bound on the size of the graph. Agent A gives the integer m to its token and transits to state `gatherer`.

Note that for every green step made in phase i of procedure $ESST^*$, the agent made less than $2g_i$ blue steps, where g_i is the total number of green steps in phase i. Hence the total number of steps made by the agent in phase i is at most $3g_i^2$. Since $g_i = 2\pi_i$, the total number of steps in phase i is at most $12\pi_i^2$.

State `token`

Agent A considers all agents that it met when they were in state `cruiser` at the moment of the first meeting (and that transited to state `explorer` upon this meeting) as its `explorers`. Agent A continues its current edge traversal and waits until one of the following two events happens for each of its `explorers`. Either an `explorer` completes procedure $ESST^*$ giving agent A an upper bound m on the size of the graph, or an `explorer` B is not back after the scheduled time at most $12\pi_i^2/\beta$ in some phase i, where β is the speed of B (learned by A). As before, there may be potentially two different upper bounds given to A by `explorers` that successfully completed procedure $ESST^*$. When one of the above events occurs for every `explorer` of the agent A, there are two cases. The first case is when at least one `explorer` managed to complete procedure $ESST^*$ and gave an upper

bound to A. In this case, agent A adopts the minimum of the (possibly two) obtained upper bounds as its upper bound on the size of the graph and transits to state `gatherer`. The second case is when all `explorers` of A failed to successfully complete the procedure (because they all crashed). Agent A learns this and transits to state `cruiser`.

State `gatherer`

In this state, agent A knows an upper bound m on the size of the graph. It starts by executing the procedure $R^*(m, v)$, where v is the node in which A transited to this state (see the description of state `gatherer` in Section 3). Recall that θ is the speed of the agent, and let c be the number of edge traversals performed by agent A in the execution of this procedure. Let $\gamma = \frac{\theta}{c}$. As before, upon completion of $R^*(m, v)$, agent A has achieved the following two goals: it met all agents whose speed is less than γ, and it learned that all agents are awake. It also follows that at this time, agent A learned the lower bound $\delta = \min(\gamma, \alpha)$ on the speeds of all good agents, where α is the minimum speed of all non-crashed agents it met.

Next, agent A executes procedure $Meeting$ for time $\tau = 4\nu\Pi(\nu, \log L)/\delta + 12\nu\rho^2(\nu)/\delta + \nu^2 P(\nu)/\delta$, where $\nu = 12\rho^2(m)$. Call this period of time the *red* period of agent A. During its red period, agent A maintains a *bag* containing *data* of each agent that agent A met or about which it heard. For each agent in the bag, data concerning it are: its label, its speed, the route to its starting node, and the lowest speed of any agent it is aware of. Moreover, for an agent in the bag in state `gatherer`, the data includes also its adopted upper bound on the size of the graph and the lower bound δ calculated by this agent. In the beginning of the red period the bag of A consists of the data for agents that A met previously, and that were not yet crashed by the time of the meeting. Whenever A meets a non-crashed agent B, it adds the content of the bag of B to its own bag. At each step, agent A updates the routes to the starting node of each agent in its bag, hence at each step these are routes from the current node.

We will show that at the end of its red period, the bag of each good agent contains the data of the same subset Σ of agents (some of which may have already crashed by this time), and that this subset contains all good agents.

Let m^* be the largest bound m calculated by any agent from the set Σ. Hence $\nu^* = 12\rho^2(m^*)$ is an upper bound on the values of ν calculated by all agents from the set Σ. Let δ^* be the minimum over all values δ and all speeds that appear in the common bag of agents from the set Σ. Let $\psi = (12\nu^* + 1)\Pi(\nu^*, \Lambda^*)/\delta^* + 108\nu^*\rho^2(\nu^*)/\delta^* + 5(\nu^*)^2 P(\nu^*)/\delta^*$. We will show that ψ is an upper bound on the termination time of red periods of all agents in the set Σ. Let κ^* be the largest speed that appears in the common bag of agents from the set Σ. Note that at the end of its red period, each good agent can compute ν^*, Λ^*, δ^*, κ^* and ψ, looking at the content of its bag. Agent A goes to the starting node of the agent whose label is the smallest in its bag, and starts its final waiting period of length $\tau' = \psi + 2\psi\kappa^*/\delta^*$.

Notice that this final waiting period has the same length for all good agents, as it does not depend on the label and the speed of an individual agent. Agent A can calculate this waiting time because, when it starts it, it knows all parameters concerning all agents in its bag.

Similarly as before, we will prove that after the final waiting period, all good agents must be at the same node, which is the starting node of the smallest agent in the bag of each of them, and that agent A learns this fact. After waiting time τ' (or at the time when some other agent declares gathering, if this happens earlier), agent A declares that all good agents are gathered, and terminates.

The proof of correctness and the performance analysis of Algorithm `Gathering with total faults` are split into a series of lemmas. Let ϵ^* be the minimum speed of any agent that any good agent ever becomes aware of. Let Λ^* be the length of the longest label of any agent that any good agent ever becomes aware of. (There may exist very slow agents with speed smaller than ϵ^*, or agents with very long labels, longer than Λ^* that crash before meeting any other agent, and thus, in the model of total faults, no good agent would be aware of such a low speed or long label.)

Lemma 4.1 *By the time $2n(2\Pi(n, \Lambda^*) + 24\rho^2(n))/\epsilon^*$ some good agent must transit to state* `gatherer`.

Proof. Consider the earliest woken good agent S_0 and some other good agent S. By the time $\Pi(n, \Lambda^*)/\epsilon^*$, agent S must be woken up, for otherwise, agent S_0 would meet it and wake it up because it would perform at least $\Pi(n, \Lambda^*)$ edge traversals executing procedure $Meeting$. Until one of these agents transits to state `gatherer`, each of them must be in one of the states `cruiser`, `explorer`, or `token`. In state `cruiser`, an agent executes procedure $Meeting$, and in state `explorer` or `token` it is involved in procedure $ESST^*$, either as an explorer or as a token. Each execution of $ESST^*$ in which a good agent is involved lasts at most the time $12\rho^2(n)/\epsilon^*$. Hence before transiting to state `gatherer`, a good agent executes procedure $Meeting$ in state `cruiser` with interruptions of duration at most $12\rho^2(n)/\epsilon^*$. There can be at most n such interruptions by the time a good agent transits to state `gatherer` because the only reason for transiting from state `explorer` or `token` to state `cruiser` is a crash of some new agent, and there are at most n agents altogether. By the time $n(2\Pi(n, \Lambda^*) + 24\rho^2(n))/\epsilon^*$ after the wakeup of agent S (if no good agent transited to state `gatherer` before) there must be a period of length at least $\Pi(n, \Lambda^*)/\epsilon^*$ when both agents S and S_0 execute procedure $Meeting$. Hence they must meet because each of them performed at least $\Pi(n, \Lambda^*)$ edge traversals executing this procedure. Then one of them transits to state `explorer`, and the other one to state `token`. Since they are both good, after an additional time at most $12\rho^2(n)/\epsilon^*$, they both transit to state `gatherer`. This happens by the time $2n(2\Pi(n, \Lambda^*) + 24\rho^2(n))/\epsilon^*$. □

Let m be the upper bound on the size of the graph, adopted by the first good agent that transited to state `gatherer`.

Lemma 4.2 *By the time $4n(2\Pi(n, \Lambda^*) + 24\rho^2(n))/\epsilon^* + 2m^2P(m)/\epsilon^* + \Pi(n, \Lambda^*)/\epsilon^*$ all good agents transit to state* `gatherer`.

Proof. Let S be the first good agent that transited to state `gatherer`. In view of Lemma 4.1, this happened by the time $2n(2\Pi(n, \Lambda^*) + 24\rho^2(n))/\epsilon^*$. After an additional time at most $2m^2P(m)/\epsilon^*$, agent S completed procedure R^*, and started its red period. Also, after this time, all

agents are woken up. Consider any other good agent S'. After time $2n(2\Pi(n, \Lambda^*) + 24\rho^2(n))/\epsilon^* + 2m^2P(m)/\epsilon^* + 2n(2\Pi(n, \Lambda^*) + 24\rho^2(n))/\epsilon^*$, agent S executes procedure $Meeting$ in state `gatherer` and agent S' executes procedure $Meeting$ in state `cruiser` (if it has not transited to state `gatherer` before). Hence, after an additional time at most $\Pi(n, \Lambda^*)/\epsilon^*$ they must meet, and agent S' transits to state `gatherer`. $\qquad\square$

Lemma 4.3 *By the end of its red period a good agent S knows that the bags of all good agents contain the data of the same subset Σ of agents, and that subset Σ contains all good agents. This happens by the time $4\nu(2\Pi(\nu, \Lambda^*) + 24\rho^2(\nu))/\epsilon^* + 4\nu^2P(\nu)/\epsilon^* + \Pi(\nu, \Lambda^*)/\epsilon^* + 4\nu\Pi(\nu, \Lambda^*)/\delta + 12\nu\rho^2(\nu)/\delta + \nu^2P(\nu)/\delta$.*

Proof. Consider a good agent S with label L which started its red period at time t. Suppose that m is the upper bound on the size of the graph, adopted by agent S. Agent S knows the lower bound δ on the speeds of all good agents, and knows some upper bound on the largest upper bound μ on the size of the graph adopted by any agent. Indeed, any agent T adopts as its upper bound the number of edge traversals in its execution of procedure $ESST^*$. This number is at most $12\rho^2(n)$. Hence it is at most $\nu = 12\rho^2(m)$, which S knows. Thus $\mu \le \nu$.

Agent S knows that by the time t all agents are woken up. Hence it knows that by the time $t + \nu(\Pi(\nu, \log L)/\delta + \Pi(\nu, \log L)/\delta + 12\rho^2(\nu)/\delta)$, all good agents transited to state `gatherer`. Indeed, consider a good agent S' that has not yet transited to state `gatherer`. Hence, after time t, it executes procedure $Meeting$ in state `cruiser`, with possible interruptions by being involved in procedure $ESST^*$, either in state `explorer` or in state `token`. Each period when S' executes $Meeting$ in state `cruiser` (between interruptions) lasts at most time $\Pi(\nu, \log L)/\delta$ because by this time S' would meet agent S. Each interruption when S' executes procedure $ESST^*$ in state `explorer` can last time at most $12\rho^2(\nu)/\delta$ because S' has speed at least δ. Each interruption when S' executes procedure $ESST^*$ in state `token` can last at most time $\Pi(\nu, \log L)/\delta$ because during such interruption agent S' is inside a single edge and hence it would meet agent S. Since there are less than ν interruptions, as each of them is due to a crash of some agent, and there are at most ν agents, we conclude that S knows that by the time $t + \nu(\Pi(\nu, \log L)/\delta + \Pi(\nu, \log L)/\delta + 12\rho^2(\nu)/\delta)$, all good agents transited to state `gatherer`.

Hence S knows that by the time $t' = t + \nu(\Pi(\nu, \log L)/\delta + \Pi(\nu, \log L)/\delta + 12\rho^2(\nu)/\delta) + \nu^2P(\nu)/\delta$ every good agent completed the procedure R^* and started its red period. Consider the time $t'' = t' + \nu(2\Pi(\nu, \log L)/\delta)$. The time segment of length $\nu(2\Pi(\nu, \log L)/\delta)$ that starts at time t' can be divided into ν time segments of length $2\Pi(\nu, \log L)/\delta$. In at least one of them no agent crashes. This segment is divided into two segments I_1 and I_2, each of length $\Pi(\nu, \log L)/\delta$. Let Σ be the set of agents that have not crashed by the beginning of segment I_1. During the time segment I_1 agent S meets all agents from the set Σ, and hence its bag at the end of I_1 is the union of the bags of all the agents from Σ. During segment I_2 agent S meets again all agents from the set Σ, and hence the bag of each of them at the end of segment I_2 is equal to the bag of agent S. After the end of segment I_2 the bags of agents do not change. The set Σ contains all good agents. We have $t'' = t + \tau$, where τ is the length of the red period of agent S. Hence by the end of red period

of agent S, i.e., by the time t'', agent S knows that the bags of all good agents contain the data of the same subset Σ of agents, and that subset Σ contains all good agents. Since $t \leq 4\nu(2\Pi(\nu, \Lambda^*) + 24\rho^2(\nu))/\epsilon^* + 4\nu^2 P(\nu)/\epsilon^* + \Pi(\nu, \Lambda^*)/\epsilon^*$ and $\tau \leq 4\nu\Pi(\nu, \Lambda^*)/\delta + 12\nu\rho^2(\nu)/\delta + \nu^2 P(\nu)/\delta$, the time estimate in the lemma follows. $\qquad\square$

Let m^* be the largest bound m calculated by any agent from the set Σ. Hence $\nu^* = 12\rho^2(m^*)$ is an upper bound on the values of ν calculated by all agents from the set Σ. Let δ^* be the minimum over all values δ and all speeds that appear in the common bag of agents from the set Σ. Hence $\delta^* \leq \epsilon^*$. Let $\psi = (12\nu^* + 1)\Pi(\nu^*, \Lambda^*)/\delta^* + 108\nu^*\rho^2(\nu^*)/\delta^* + 5(\nu^*)^2 P(\nu^*)/\delta^*$. It follows from Lemma 4.3 that ψ is an upper bound on the termination of red periods of all agents from the set Σ. (Note that if an agent from the set Σ crashed before completing its red period, this red period could not extend beyond the time established in Lemma 4.3 for good agents.) Let κ^* be the largest speed that appears in the common bag of agents from the set Σ.

Lemma 4.4 *By the time $\psi + 2\psi\kappa^*/\delta^*$ any good agent S is at the starting node of the smallest agent S^* from the set Σ, and starts its final waiting period.*

Proof. Let T be the trajectory of agent S until the end of its red period, and let T' be the trajectory from the initial node of agent S^* to some node x of the trajectory T, learned by agent S from its bag. (Notice that agent S^* might have crashed and this trajectory may sometimes be learned by agent S indirectly.) The length of each of the trajectories T and T' is at most $\psi\kappa^*$. After the end of its red period, agent S knows both these trajectories. Hence it can backtrack to node x and then backtrack to the starting node of agent S^*. This takes at most $2\psi\kappa^*$ edge traversals, and hence it takes time at most $2\psi\kappa^*/\delta^*$. Hence by the time $\psi + 2\psi\kappa^*/\delta^*$ agent S is at the starting node of agent S^* and starts its final waiting period. $\qquad\square$

Lemma 4.5 *By the time $2\psi + 4\psi\kappa^*/\delta^*$ all good agents are at the starting node of the smallest agent from the set Σ, all of them know it, and correctly declare that gathering is completed.*

Proof. We first show that at the beginning of its final waiting period, every good agent can compute the length $\tau' = \psi + 2\psi\kappa^*/\delta^*$ of this period. Indeed this value depends on four parameters: Λ^*, κ^*, ν^* and δ^*. At the beginning of its final waiting period, every good agent can compute these parameters from its bag (that is the same for all good agents). Hence it can compute ψ and finally τ'.

In view of Lemma 4.4, by the time $\psi + 2\psi\kappa^*/\delta^*$, every good agent is at the starting node of the smallest agent S^* from the set Σ, and starts its final waiting period. Hence at the end of its final waiting period, which has the same length $\tau' = \psi + 2\psi\kappa^*/\delta^*$, every good agent knows that all good agents started their final waiting period. Hence, by the time $2\psi + 4\psi\kappa^*/\delta^*$, every good agent correctly declares that gathering is completed. $\qquad\square$

We are now ready to prove the main result of this section.

Theorem 4.1 *Algorithm* `Gathering with total faults` *is a correct algorithm for the task of gathering with total faults. It works in time polynomial in the size n of the graph, in the length Λ of the largest label, in the inverse of the smallest speed ϵ, and in the ratio r between the largest and the smallest speed.*

Proof. The correctness of the algorithm follows immediately from Lemma 4.5. It remains to verify that the time $2\psi + 4\psi\kappa^*/\delta^*$ is polynomial in n, Λ, $1/\epsilon$ and r.

We have $\kappa^* \leq \kappa$. The value $1/\delta$ computed by each agent is polynomial in n and in $1/\epsilon$, and thus so is $1/\delta^*$. Hence κ^*/δ^* is polynomial in $r = \kappa/\epsilon$, in n and in $1/\epsilon$. Thus it is enough to verify that $\psi = (12\nu^* + 1)\Pi(\nu^*, \Lambda^*)/\delta^* + 108\nu^*\rho^2(\nu^*)/\delta^* + 5(\nu^*)^2 P(\nu^*)/\delta^*$ is polynomial in n, Λ and $1/\epsilon$.

The upper bound m on the size of the graph, calculated by each agent, is at most $12\rho^2(n)$. Since ρ is a polynomial, it follows that m is polynomial in n, and so is m^*. Hence $\nu^* = 12\rho^2(m^*)$ is also polynomial in n.

We have $\Lambda^* \leq \Lambda$, and hence each of the three summands in ψ are polynomial in n, Λ and $1/\epsilon$ because Π, ρ and P are polynomials. Hence ψ is polynomial in n, Λ and $1/\epsilon$. $\qquad\square$

5 Conclusion

Our algorithm for motion faults works for any team of at least two agents, regardless of the number of faulty agents, and our algorithm for total faults works for any team of agents, containing at least two good agents; the latter is necessary. Hence, in terms of feasibility of gathering with crash faults, our results are optimal.

We showed that the time of our algorithms is polynomial in the size n of the graph, in the length Λ of the largest label, in the inverse of the smallest speed ϵ, and in the ratio r between the largest and the smallest speed. Optimizing this polynomial seems to be out of the reach at present, as even the optimal rendezvous time of two good agents is not known for arbitrary graphs. However, it remains open if the time of gathering with crash faults really has to depend on all four above parameters. The dependence on n and on $1/\epsilon$ seems natural. The time of gathering should somehow depend on the labels as well, but maybe it could depend on the length of the smallest label (as it is the case for rendezvous without faults) and not the largest. The most problematic seems the dependence on the ratio between the largest and the smallest speed. Is this dependence necessary?

References

[1] N. Agmon, D. Peleg, Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots. SIAM J. Comput. 36 (2006), 56-82.

[2] S. Alpern, The rendezvous search problem, SIAM J. on Control and Optimization 33 (1995), 673-683.

[3] S. Alpern, Rendezvous search on labelled networks, Naval Reaserch Logistics 49 (2002), 256-274.

[4] S. Alpern and S. Gal, The theory of search games and rendezvous. Int. Series in Operations research and Management Science, Kluwer Academic Publisher, 2002.

[5] E. Anderson and R. Weber, The rendezvous problem on discrete locations, Journal of Applied Probability 28 (1990), 839-851.

[6] E. Anderson and S. Fekete, Asymmetric rendezvous on the plane, Proc. 14th Annual ACM Symp. on Computational Geometry (1998), 365-373.

[7] E. Anderson and S. Fekete, Two-dimensional rendezvous search, Operations Research 49 (2001), 107-118.

[8] V. Baston and S. Gal, Rendezvous on the line when the players' initial distance is given by an unknown probability distribution, SIAM J. on Control and Opt. 36 (1998), 1880-1889.

[9] V. Baston and S. Gal, Rendezvous search when marks are left at the starting points, Naval Reaserch Logistics 48 (2001), 722-731.

[10] S. Bouchard, Y. Dieudonné, B. Ducourthial, Byzantine gathering in networks, Distributed Computing 29 (2016), 435-457.

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

[12] M. Cieliebak, P. Flocchini, G. Prencipe, N. Santoro, Distributed Computing by Mobile Robots: Gathering. SIAM J. Comput. 41(2012), 829-879.

[13] J. Czyzowicz, A. Kosowski, A. Pelc, How to meet when you forget: Log-space rendezvous in arbitrary graphs, Distributed Computing 25 (2012), 165-178.

[14] J. Czyzowicz, A. Labourel, A. Pelc, How to meet asynchronously (almost) everywhere, ACM Transactions on Algorithms 8 (2012), article 37.

[15] G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, U. Vaccaro, Asynchronous deterministic rendezvous in graphs, Theoretical Computer Science 355 (2006), 315-326.

[16] A. Dessmark, P. Fraigniaud, D. Kowalski, A. Pelc. Deterministic rendezvous in graphs. Algorithmica 46 (2006), 69-96.

[17] Y. Dieudonné, A. Pelc, Deterministic polynomial approach in the plane, Distributed Computing 28 (2015), 111-129.

[18] Y. Dieudonné, A. Pelc, Anonymous meeting in networks, Algorithmica 74 (2016), 908-946 .

[19] Y. Dieudonné, A. Pelc, D. Peleg, Gathering despite mischief, ACM Transactions on Algorithms 11 (2014), article 1.

[20] Y. Dieudonné, A. Pelc, V. Villain, How to meet asynchronously at polynomial cost, SIAM Journal on Computing 44 (2015), 844-867.

[21] P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of asynchronous robots with limited visibility. Theor. Comput. Sci. 337 (2005), 147-168.

[22] P. Fraigniaud, A. Pelc, Delays induce an exponential memory gap for rendezvous in trees, ACM Trans. Algorithms 9 (2013), article 17.

[23] S. Gal, Rendezvous search on the line, Operations Research 47 (1999), 974-976.

[24] A. Israeli and M. Jalfon, Token management schemes and random walks yield self stabilizing mutual exclusion, Proc. 9th Annual ACM Symposium on Principles of Distributed Computing (PODC 1990), 119-131.

[25] D. Kowalski, A. Malinowski, How to meet in anonymous network, in 13th Int. Colloquium on Structural Information and Comm. Complexity, (SIROCCO 2006), Springer LNCS 4056, 44-58.

[26] E. Kranakis, D. Krizanc, and P. Morin, Randomized Rendez-Vous with Limited Memory, Proc. 8th Latin American Theoretical Informatics (LATIN 2008), Springer LNCS 4957, 605-616.

[27] E. Kranakis, D. Krizanc, N. Santoro and C. Sawchuk, Mobile agent rendezvous in a ring, Proc. 23rd Int. Conference on Distributed Computing Systems (ICDCS 2003), IEEE, 592-599.

[28] A. Pelc, Deterministic rendezvous in networks: A comprehensive survey, Networks 59 (2012), 331-34.

[29] O. Reingold, Undirected connectivity in log-space, Journal of the ACM 55 (2008).

[30] T. Schelling, The strategy of conflict, Oxford University Press, Oxford, 1960.

[31] A. Ta-Shma and U. Zwick. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), 599-608.

[32] X. Yu and M. Yung, Agent rendezvous: a dynamic symmetry-breaking problem, Proc. International Colloquium on Automata, Languages, and Programming (ICALP 1996), Springer LNCS 1099, 610-621.