Fault-Tolerant Rendezvous in Networks

Jérémie Chalopin[1], Yoann Dieudonné[2], Arnaud Labourel[1], and Andrzej Pelc[3]*

[1] LIF, CNRS & Aix-Marseille University, Marseille, France
[2] MIS, Université de Picardie Jules Verne, France
[3] Département d'informatique, Université du Québec en Outaouais, Gatineau, Québec, Canada

Abstract. Two mobile agents, starting from different nodes of an unknown network, have to meet at the same node. Agents move in synchronous rounds using a deterministic algorithm. Each agent has a different label, which it can use in the execution of the algorithm, but it does not know the label of the other agent. Agents do not know any bound on the size of the network. In each round an agent decides if it remains idle or if it wants to move to one of the adjacent nodes. Agents are subject to *delay faults*: if an agent incurs a fault in a given round, it remains in the current node, regardless of its decision. If it planned to move and the fault happened, the agent is aware of it. We consider three scenarios of fault distribution: random (independently in each round and for each agent with constant probability $0 < p < 1$), unbounded adversarial (the adversary can delay an agent for an arbitrary finite number of consecutive rounds) and bounded adversarial (the adversary can delay an agent for at most c consecutive rounds, where c is unknown to the agents). The quality measure of a rendezvous algorithm is its cost, which is the total number of edge traversals.
For random faults, we show an algorithm with cost polynomial in the size n of the network and *polylogarithmic* in the larger label L, which achieves rendezvous with very high probability in arbitrary networks. By contrast, for unbounded adversarial faults we show that rendezvous is not feasible, even in the class of rings. Under this scenario we give a rendezvous algorithm with cost $O(n\ell)$, where ℓ is the smaller label, working in arbitrary trees, and we show that $\Omega(\ell)$ is the lower bound on rendezvous cost, even for the two-node tree. For bounded adversarial faults, we give a rendezvous algorithm working for arbitrary networks, with cost polynomial in n, and *logarithmic* in the bound c and in the larger label L.

Keywords: rendezvous, deterministic algorithm, mobile agent, delay fault.

* Partially supported by NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

1 Introduction

1.1 The background

Two mobile entities, called agents, starting from different nodes of a network, have to meet at the same node. This task is known as *rendezvous* and has been extensively studied in the literature. Mobile entities may represent software agents in computer networks, mobile robots, if the network is composed of corridors in a mine, or people who want to meet in an unknown city whose streets form a network. The reason to meet may be to exchange data previously collected by the agents, or to coordinate a future network maintenance task. In this paper we study a fault-tolerant version of the rendezvous problem: agents have to meet in spite of delay faults that they can incur during navigation. Such faults may be due to mechanical reasons in the case of robots and to network congestion in the case of software agents.

1.2 The model and the problem

The network is modeled as an undirected connected graph. We seek deterministic rendezvous algorithms that do not rely on the knowledge of node identifiers, and can work in anonymous graphs as well (cf. [3]). The importance of designing such algorithms is motivated by the fact that, even when nodes are equipped with distinct identifiers, agents may be unable to perceive them because of limited sensory capabilities (a robot may be unable to read signs at corridor crossings), or nodes may refuse to reveal their identifiers to software agents, e.g., due to security or privacy reasons. Note that, if nodes had distinct identifiers visible to the agents, the agents might explore the graph and meet at the node with smallest identifier, hence rendezvous would reduce to graph exploration. On the other hand, we assume that edges incident to a node v have distinct labels (visible to the agents) 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 rendezvous would be often impossible, as the adversary could prevent an agent from taking some edge incident to the current node. Security and privacy reasons for not revealing node identifiers to software agents are irrelevant in the case of port numbers, and port numbers in the case of a mine or labyrinth can be made implicit, e.g., by marking one edge at each crossing (using a simple mark legible by the robot), considering it as corresponding to port 0 and all other port numbers increasing clockwise.

Agents start at different nodes of the graph and traverse its edges in synchronous rounds. They cannot mark visited nodes or traversed edges in any way. The adversary wakes up each of the agents in possibly different rounds. Each agent starts executing the algorithm in the round of its wake-up. It has a clock measuring rounds that starts at its wake-up round. In each round an agent decides if it remains idle or if it chooses a port to move to one of the adjacent nodes.

Agents are subject to *delay faults* in rounds in which they decide to move: if an agent incurs a fault in such a round, it remains at the current node and is aware of the fault. We consider three scenarios of fault distribution: random (independently in each round and for each agent with constant probability $0 < p < 1$), unbounded adversarial (the adversary can delay an agent for an arbitrary finite number of consecutive rounds) and bounded adversarial (the adversary can delay an agent for at most c consecutive rounds, where c is unknown to the agents). Agents do not know the topology of the graph or any bound on its size. Each agent has a different positive integer label which it knows and can use in the execution of the rendezvous algorithm, but it does not know the label of the other agent nor its starting round. When an agent enters a node, it learns its degree and the port of entry. When agents cross each other on an edge, traversing it simultaneously in different directions, they do not notice this fact. We assume that the memory of the agents is unlimited: from the computational point of view they are modeled as Turing machines.

The quality measure of a rendezvous algorithm is its *cost*, which is the total number of edge traversals. For each of the considered fault distributions we are interested in deterministic algorithms working at low cost. For both scenarios with adversarial faults we say that a deterministic rendezvous algorithm works at a cost at most C for a given class of graphs if for any initial positions in a graph of this class both agents meet after at most C traversals, regardless of the faults imposed by the adversary obeying the given scenario. In the case of random faults the algorithm is also deterministic, but, due to the stochastic nature of faults, the estimate of its cost is with high probability.

1.3 Our results

For random faults, we show an algorithm which achieves rendezvous in arbitrary networks at cost polynomial in the size n of the network and *polylogarithmic* in the larger label L, with very high probability. More precisely, our algorithm achieves rendezvous with probability 1, and its cost exceeds a polynomial in n and $\log L$ with probability inverse exponential in n and $\log L$. By contrast, for unbounded adversarial faults, we show that rendezvous is not feasible, even in the class of rings. Under this scenario we give a rendezvous algorithm with cost $O(n\ell)$, where ℓ is the smaller label, working in arbitrary trees, and we show that $\Omega(\ell)$ is the lower bound on rendezvous cost, even for the two-node tree. For bounded adversarial faults we give a rendezvous algorithm working for arbitrary networks, with cost polynomial in n, and *logarithmic* in the bound c and in the larger label L.

1.4 Related work

The problem of rendezvous has been studied both under the randomized and the deterministic scenarios. An extensive survey of randomized rendezvous in various models can be found in [3], cf. also [1, 2, 4, 8, 24]. Deterministic rendezvous in networks has been surveyed in [31]. Several authors considered the geometric

scenario (rendezvous in an interval of the real line, see, e.g., [8, 9], or in the plane, see, e.g., [5, 6]). Gathering more than two agents has been studied, e.g., in [22, 24, 29, 34].

For the deterministic setting many authors studied the feasibility of synchronous rendezvous, and the time required to achieve this task, when feasible. For instance, deterministic rendezvous of agents equipped with tokens used to mark nodes was considered, e.g., in [28]. Deterministic rendezvous of two agents that cannot mark nodes but have unique labels was discussed in [17, 26, 33]. Since this is our scenario, these papers are the most relevant in our context. All of them are concerned with the time of rendezvous in arbitrary graphs. In [17] the authors show a rendezvous algorithm polynomial in the size of the graph, in the length of the shorter label and in the delay between the starting times of the agents. In [26, 33] rendezvous time is polynomial in the first two of these parameters and independent of the delay.

Memory required by the agents to achieve deterministic rendezvous has been studied in [23] for trees and in [12] for general graphs. Memory needed for randomized rendezvous in the ring is discussed, e.g., in [27].

Apart from the synchronous model used in this paper, several authors investigated asynchronous rendezvous in the plane [11, 22] and in network environments [7, 13, 16, 20]. 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.

Fault-tolerant aspects of the rendezvous problem have been investigated in [10, 14, 15, 19, 21]. Faulty unmovable tokens were considered in the context of the task of gathering many agents at one node. In [14, 21] the authors considered gathering in rings, and in [15] gathering was studied in arbitrary graphs, under the assumption that an unmovable token is located in the starting node of each agent. Tokens could disappear during the execution of the algorithm, but they could not reappear again. Byzantine tokens which can appear and disappear arbitrarily have been considered in [18] for the related task of network exploration. A different fault scenario for gathering many agents was investigated in [19]. The authors assumed that some number of agents are Byzantine and they studied the problem of how many good agents are needed to guarantee meeting of all of them despite the actions of Byzantine agents. To the best of our knowledge rendezvous with delay faults considered in the present paper has never been studied before.

2 Preliminaries

Throughout the paper, the number of nodes of a graph is called its size. In this section we recall two procedures known from the literature, that will be used as building blocks in some of our algorithms. The aim of the first procedure is graph exploration, i.e., visiting all nodes and traversing all edges of the graph by

a single agent. The procedure, based on universal exploration sequences (UXS) [25], is a corollary of the result of Reingold [32]. Given any positive integer m, it allows the agent to traverse all edges of any graph of size at most m, starting from any node of this graph, using $P(m)$ edge traversals, where P is some 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 $R(m, v)$ the trajectory obtained by Reingold's procedure followed by its reverse. (Hence the trajectory starts and ends at node v.) 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 $R(m, v)$ is integral if it is obtained by Reingold's procedure applied in any graph of size at most m starting at any node v.

The second auxiliary procedure is the Algorithm `RV-asynch-poly` from [20] that guarantees rendezvous of two agents under the *asynchronous* scenario. Unlike in the synchronous scenario used in the present paper, in the asynchronous scenario each agent chooses consecutive ports that it wants to use but the adversary controls the speed of the agent, changing it arbitrarily during navigation. Rendezvous is guaranteed in the asynchronous scenario, if it occurs for any behavior of the adversary. 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. Recall that in our synchronous scenario, agents crossing each other on an edge traversing it simultaneously in different directions, not only do not meet but do not even notice the fact of crossing.

Algorithm `RV-asynch-poly` works at cost polynomial in the size n of the graph in which the agents operate and in the length of the smaller label. Let A be a polynomial, such that if two agents with different labels λ_1 and λ_2 execute Algorithm `RV-asynch-poly` in an n-node graph, then the agents meet in the asynchronous model, after at most $A(n, \min(\log \lambda_1, \log \lambda_2))$ steps.

3 Random faults

In this section we consider the scenario when agents are subject to random and independent faults. More precisely, for each agent and each round the probability that the agent is delayed in this round is $0 < p < 1$, where p is a constant, and the events of delaying are independent for each round and each agent. Under this scenario we construct a deterministic rendezvous algorithm that achieves rendezvous in any connected graph with probability 1 and its cost exceeds a

polynomial in n and $\log L$ with probability inverse exponential in n and $\log L$, where n is the size of the graph and L is the larger label.

The intuition behind the algorithm is the following. Since the occurrence of random faults represents a possible behavior of the asynchronous adversary in Algorithm `RV-asynch-poly` from [20], an idea to get the guarantee of a meeting with random faults at polynomial cost might be to only use this algorithm. However, this meeting may occur either at a node or inside an edge, according to the model from [20]. In the synchronous model with random faults considered in this section, the second type of meeting is not considered as rendezvous, in fact agents do not even notice it. Hence we must construct a *deterministic* mechanism which guarantees a legitimate meeting at a node, with high probability, soon after an "illegitimate" meeting inside an edge. Constructing this mechanism and proving that it works as desired is the main challenge of rendezvous with random faults.

3.1 The algorithm

Before describing the algorithm we define the following transformation of the label λ of an agent. Let $\Phi(0) = (0011)$ and $\Phi(1) = (1100)$. Let $(c_1 \dots c_k)$ be the binary representation of the label λ. We define the *modified label* λ^* of the agent as the concatenation of sequences $\Phi(c_1), \dots, \Phi(c_k)$ and (10). Note that if labels of two agents are different, then their transformed labels are different and none of them is a prefix of the other.

We first describe the procedure `Dance` (λ, x, y) executed by an agent with label λ located at node y at the start of the procedure. Node x is a node adjacent to y.

Procedure `Dance` (λ, x, y)
Let $\lambda^* = (b_1, \dots, b_m)$.
Stage 1.
Stay idle at y for 10 rounds.
Stage 2.
for $i = 1$ **to** m **do**
    if $b_i = 0$
    then stay idle for two rounds
    else go to x and in the next round return to y.
Stage 3.
Traverse the edge $\{x, y\}$ 12 times (i.e., go back and forth 6 times across the edge $\{x, y\}$). ◇

Note that procedure `Dance` (λ, x, y) has cost $O(\log \lambda)$.

We will also use procedure `Asynch`(λ) executed by an agent with label λ starting at any node x_0 of a graph. This procedure produces an infinite walk $(x_0, x_1, x_2, \dots)$ in the graph resulting from applying Algorithm `RV-asynch-poly` by a single agent with label λ.

Using these procedures we now describe Algorithm `RV-RF` (for rendezvous with random faults), that works for an agent with label λ starting at an arbitrary node of any connected graph.

Algorithm `RV-RF`

The algorithm works in two phases interleaved in a way depending on faults occurring in the execution and repeated until rendezvous. The agent starts executing the algorithm in phase Progress.

Phase Progress

This phase proceeds in stages. Let $(x_0, x_1, x_2, \dots)$ be the infinite walk produced by the agent starting at node x_0 and applying `Asynch`(λ). The ith stage of phase Progress, for $i \geq 1$, is the traversal of the edge $\{x_{i-1}, x_i\}$ from x_{i-1} to x_i, followed by the execution of `Dance` (λ, x_{i-1}, x_i). The agent executes consecutive stages of phase Progress until a fault occurs.

If a fault occurs in the first round of the ith stage, then the agent repeats the attempt of this traversal again, until success and then continues with `Dance` (λ, x_{i-1}, x_i). If a fault occurs in the tth round of the ith stage, for $t > 1$, i.e., during the execution of procedure `Dance` (λ, x_{i-1}, x_i) in the ith stage, then this execution is interrupted and phase Correction is launched starting at the node where the agent was situated when the fault occurred.

Phase Correction

Let e denote the edge $\{x_{i-1}, x_i\}$ and let w be the node at which the agent was situated when the last fault occurred during the execution of `Dance` (λ, x_{i-1}, x_i). Hence w is either x_{i-1} or x_i.

Stage 1.
Stay idle at w for 20 rounds.
Stage 2.
Traverse edge e 20 times.
Stage 3.
If the agent is not at w, then go to w.

If a fault occurs during the execution of phase Correction, then the execution of this phase is dropped and a new phase Correction is launched from the beginning, starting at the node where the agent was situated when the fault occurred. Upon completing an execution of the phase Correction without any fault the agent is at node w. It resumes the execution of the tth round of the ith stage of phase Progress. ◇

3.2 Correctness and analysis

This section is devoted to the proof of correctness and analysis of performance of Algorithm `RV-RF`. It is split into a series of lemmas. The first lemma is straightforward.

Lemma 1. *In every segment of 10 consecutive rounds without any faults of execution of Stage 2 of procedure* `Dance` *the agent is idle at least once and moves at least once.*

Lemma 2. *In every segment of 20 consecutive rounds without any faults of execution of procedure* `Dance` *the agent moves at least once.*

Proof. Since Stage 1 of procedure `Dance` consists of 10 rounds in which the agent is idle and in Stage 3 the agent is never idle, the lemma follows from the second part of Lemma 1. □

In our reasoning we will consider an auxiliary model $\mathcal{M}$ of the behavior of agents and of the type of faults. Agents move in synchronous rounds of constant duration T. An edge traversal is always performed at a constant speed and so that the destination node is reached exactly at the end of the round involving the traversal. The agents do not notice when they meet, neither at a node nor inside an edge. Hence, when they execute Algorithm `RV-RF`, they do so indefinitely in an independent way. Faults in model $\mathcal{M}$ are unbounded adversarial, i.e., the adversary can delay an agent at a node for an arbitrary finite number of consecutive rounds. When an agent executes procedure `Asynch`(λ) in model $\mathcal{M}$, it attempts to make the next step of the procedure in each round, but can be delayed by the adversary at each step. For rounds $t < t'$ we denote by $[t, t']$ the time interval between the beginning of round t and the end of round t'. We use (t, t') instead of $[t+1, t'-1]$. For convenience, we will sometimes use the phrase "in round t" instead of "at the end of round t" and "by round t" instead of "by the end of round t". Considerations in this auxiliary model will serve us to draw conclusions about rendezvous in the random fault model.

A meeting in model $\mathcal{M}$ is defined as both agents being at the same node at the same time or being in the same point inside an edge at the same time. We use the word *meeting* in the auxiliary model $\mathcal{M}$ to differentiate it from *rendezvous* in our principal model: the first may occur at a node or inside an edge and agents do not notice it, and the second can occur only at a node, agents notice it and stop. Notice that five types of meetings are possible in model $\mathcal{M}$.

Type 1. The agents cross each other inside an edge $\{u, v\}$ in some round, one agent going from u to v and the other going from v to u.

Type 2. The agents stay together inside an edge during its traversal in the same round in the same direction.

Type 3. The agents meet at node v coming from the same node u, not necessarily in the same round.

Type 4. The agents meet at node v coming from different nodes u and w, not necessarily in the same round.

Type 5. The agents meet at node v, such that one of them has never moved from v.

Hence when there is a meeting in model $\mathcal{M}$, only one of the following 9 situations can occur:

Situation A1. The agents cross each other inside an edge $\{u, v\}$ in some round, one agent going from u to v and the other going from v to u.

Situation A2. Agents meet at a node v, such that one of them has never moved yet.

Situation A3. Agents meet at a node v, both coming from the same node u in different rounds.

Situation A4. Agents a and b meet at a node v, such that:

1. agent a comes from a node u to v in round $k_{1,a}$ and after the meeting goes to a node $w \neq u$ in round $k_{2,a}$;
2. agent b comes from node w to v in round $k_{1,b}$ and after the meeting goes to node u in round $k_{2,b}$.
3. rounds $k_{1,a}$ and $k_{1,b}$ are not necessarily the same; rounds $k_{2,a}$ and $k_{2,b}$ are not necessarily the same.

Situation A5. Agents a and b meet at a node v, such that:

1. agent a comes to v from u in its rth edge traversal and goes to node $w \neq u$ in its $(r+1)$th edge traversal;
2. agent b comes to v from w in its sth edge traversal and goes to a node $z \neq u$ in its $(s+1)$th edge traversal;
3. the rth edge traversal of agent a and the sth edge traversal of agent b are executed not necessarily in the same round;
4. agent a makes its $(r+1)$th edge traversal in the same round or before the round when b makes its $(s+1)$th edge traversal.

Situation B1. Agents meet at a node v, both coming from the same node u in the same round.

Situation B2 Agents meet inside an edge $\{u, v\}$, both coming from u and going to v in the same round. Since they move at the same constant speed, they are simultaneously at each point inside the edge $\{u, v\}$ during the traversal. We will consider this as a single meeting.

Situation B3. Agents meet at a node v, such that:

1. agent a comes from a node u to v in round $k_{1,a}$ and after the meeting goes to a node w in round $k_{2,a}$;
2. agent b comes from node p to v in round $k_{1,b}$ and after the meeting goes to node q in round $k_{2,b}$,
3. $p \neq w$, $q \neq u$; rounds $k_{1,a}$ and $k_{1,b}$ are not necessarily the same; rounds $k_{2,a}$ and $k_{2,b}$ are not necessarily the same.

Situation B4. Agents meet at node v such that

1. agent a comes to v from u in its rth edge traversal and goes to node $w \neq u$ in its $(r+1)$th edge traversal;
2. agent b comes to v from w in its sth edge traversal and goes to a node $z \neq u$ in its $(s+1)$th edge traversal;
3. the rth edge traversal of agent a and the sth edge traversal of agent b are executed not necessarily in the same round;
4. agent a makes its $(r+1)$th edge traversal after the round when b makes its $(s+1)$th edge traversal.

The next lemma shows that one of the situations A1 – A5 is unavoidable when applying Asynch in the model $\mathcal{M}$.

Lemma 3. *Consider two agents a and b, with different labels λ_1 and λ_2, respectively, starting at arbitrary different nodes of an n-node graph, where n is unknown to the agents. In the model $\mathcal{M}$, if agent a applies procedure* `Asynch`*(λ_1) and agent b applies procedure* `Asynch`*(λ_2), then, for every behavior of the adversary at least one of the situations A1 – A5 must occur after a total of at most $A(n, \min(\log \lambda_1, \log \lambda_2))$ steps of both agents.*

Proof. Suppose, for contradiction, that there exists a behavior of the adversary in model $\mathcal{M}$ such that all situations A1 – A5 can be avoided during the first $A(n, \min(\log \lambda_1, \log \lambda_2))$ steps of both agents. Denote by S the scenario truncated to the first $A(n, \min(\log \lambda_1, \log \lambda_2))$ steps of both agents resulting from the above behavior. Since Algorithm `RV-asynch-poly` guarantees a meeting under any behavior of an adversary in the asynchronous model after a total of at most $A(n, \min(\log \lambda_1, \log \lambda_2))$ steps of both agents (cf. [20]), this means that in scenario S, which takes place in the model $\mathcal{M}$, at least one of the situations B1 – B4 must occur. We will show that based on scenario S in model $\mathcal{M}$ it is possible to construct a scenario AS in the asynchronous model from [20] with no meetings after a total of at most $A(n, \min(\log \lambda_1, \log \lambda_2))$ steps, which contradicts the result from [20].

Consider the first meeting ρ in scenario S. This meeting cannot be in situation B1 or B2 because then the agents would meet previously at node u. Hence either situation B3 or B4 must occur.

First suppose that situation B4 occurred during the meeting ρ. Split the scenario S into two parts S_1 and S_2, such that S_2 is the part of scenario S that consists of all rounds after the round in which agent b makes its $(s+1)$th edge traversal. In particular, in the beginning of S_2 agent a has already made its rth edge traversal but not yet its $(r+1)$th edge traversal. S_1 is the part of scenario S preceding S_2. Notice that S_1 contains only one meeting, the meeting ρ.

We can modify the behavior of the adversary in scenario S_1 to produce scenario S_1' in which there is no meeting. In order to do so, consider 3 cases.

Case 1. The meeting ρ occurs in the round when both a and b arrive at v.

In the round preceding the execution of its rth edge traversal agent a is at node u, while b is at node $w \neq u$. According to scenario S_1, in the next round both agents go to v. The modification to obtain scenario S_1' is as follows. The adversary delays agent a at u for one round and releases agent b to make its sth edge traversal to v. Then the adversary releases agent a to go to v and agent b to make its $(s+1)$th edge traversal to z. This avoids the meeting.

Case 2. The meeting ρ occurs in a round in which b is idle at v (it came there in a previous round) and a comes to v from u executing its rth edge traversal.

In the round preceding this traversal agent a was at u. The modification to obtain scenario S_1' is as follows. The adversary releases b to make its $(s+1)$th edge traversal to z and in the same round releases a to make its rth edge traversal. This avoids the meeting.

Case 3. The meeting ρ occurs in a round in which a is idle at v (it came there in a previous round) and b comes to v from w executing its sth edge traversal.

Let t be the round in which agent a makes its rth edge traversal and $t' > t$ the round in which agent b makes its sth edge traversal. In the time interval $[x, y]$, where x is the beginning of round t and y is the end of round $t' - 1$, agent b does not traverse edge $\{u, v\}$ because otherwise the agents would previously meet inside this edge or at v. The modification to obtain scenario S_1' is as follows. The adversary starts moving agent a from u to v in the beginning of round t, but blocks it inside the edge $\{u, v\}$ until the end of the round t' in which b makes its sth edge traversal to v, which it is released to do. In the next round the adversary releases b to make its $(s + 1)$th edge traversal to z and releases a to finish its rth edge traversal to v (these two actions finish simultaneously). This avoids the meeting.

Hence in all cases we obtain a scenario S_1' which does not contain any meeting. Notice that scenario S_1' is still a scenario in model $\mathcal{M}$ in cases 1 and 2, but is *not* a scenario in this model in case 3. Indeed, in this case agent a does not travel with constant speed inside the edge $\{u, v\}$. However, in all cases this is a legitimate scenario for the asynchronous adversary. It ends at the end of a round when a has made its rth edge traversal and when b has made its $(s+1)$th edge traversal but before b has started its $(s+2)$th edge traversal. Hence scenario S' consisting of scenario S_1' followed by S_2 is possible for the asynchronous adversary, has one fewer meeting than scenario S and all situations A1 – A5 are still avoided.

In a similar way it can be shown that if situation B3 occurred during the meeting ρ, then a scenario avoiding the first meeting of S can be produced. Notice that scenario S_2 remained unchanged and it is still a scenario in model $\mathcal{M}$. The same reasoning can be now applied to scenario S_2, again transforming it into a scenario with one fewer meeting (avoiding the first meeting in S_2), the last part of which (containing other meetings, if any, except the first meeting of scenario S_2) is still a scenario in model $\mathcal{M}$.

By induction on the number of meetings it follows that a scenario AS for some behavior of the asynchronous adversary (not for model $\mathcal{M}$ any more) without any meeting after a total of $A(n, \min(\log \lambda_1, \log \lambda_2))$ steps can be produced. This, however, contradicts the fact that Algorithm `RV-asynch-poly` guarantees a meeting under any behavior of the asynchronous adversary after a total of at most $A(n, \min(\log \lambda_1, \log \lambda_2))$ steps of both agents. □

Lemma 4. *Consider two agents a and b, with different labels λ_1 and λ_2, respectively, starting at arbitrary different nodes of an n-node graph, where n is unknown to the agents. In the model $\mathcal{M}$, if the agents have executed at least $A(n, \min(\log \lambda_1, \log \lambda_2))$ stages of phase Progress of Algorithm* `RV-RF`*, then at least one of the following events must have occurred:*

Event 1. There exists a round k during which the agents cross each other inside an edge $\{u, v\}$, one agent going from u to v and the other going from v to u, each of the agents applying a step of procedure `Asynch`*.*

Event 2. There exists a round k during which one agent arrives at node v applying a step of procedure `Asynch` *and the other agent has never moved yet.*

Event 3. There exists a round k during which one agent arrives at node v from u applying a step of procedure `Asynch`*, and the other agent has also arrived*

at v from u applying a step of procedure `Asynch` *in some round $k' < k$ and has not made any further step of procedure* `Asynch` *until the end of round k.*

Event 4. There exists a round k during which agent b arrives at node v from w applying a step of procedure `Asynch` *and such that:*

- *agent a has arrived at v from $u \neq w$ applying a step of procedure* `Asynch` *in some round $k' \leq k$ and has not made any further step of procedure* `Asynch` *during the time interval $[k', k]$;*
- *the next step of procedure* `Asynch` *after round k brings agent a to node w;*
- *the next step of procedure* `Asynch` *after round k brings agent b to node u.*

Event 5. There exists a round k_1 during which agent a arrives at node v from u applying a step of procedure `Asynch` *and such that:*

- *the next step of procedure* `Asynch` *after round k_1 performed by agent a in a round $k_2 > k_1$, brings agent a to node $w \neq u$;*
- *agent b arrives at v from w applying a step of procedure* `Asynch` *in some round $p_1 < k_2$ and goes from v to some node $z \neq u$ applying the next step of procedure* `Asynch` *in some round $p_2 \geq k_2$.*

Proof. Consider a scenario S in model $\mathcal{M}$ in which none of the events 1 – 5 occurred by the time the agents completed $A(n, \min(\log \lambda_1, \log \lambda_2))$ stages of phase Progress of Algorithm `RV-RF`. Replace in scenario S each action that does not correspond to a step of procedure `Asynch` by (a tentative of) performing by the agent the next step of procedure `Asynch` and imposing in this round a delay fault by the adversary. The obtained scenario S' is a legitimate scenario in the execution of procedure `Asynch` in model $\mathcal{M}$ by agents with labels λ_1 and λ_2 in which, after $A(n, \min(\log \lambda_1, \log \lambda_2))$ steps of both agents executed in this procedure none of the situations A1 – A5 took place. This contradicts Lemma 3. □

In the sequel we will need the following notions. The *home* of an agent is the last node at which it arrived applying procedure `Asynch` and the *cottage* of an agent is the previous node that it reached applying procedure `Asynch` (or the starting node of the agent, if the home is reached in the first step of procedure `Asynch`). The home of an agent a is denoted by $Home(a)$ and its cottage by $Cot(a)$. Hence if agent a arrives at y from x applying a step of procedure `Asynch`, then $x = Cot(a)$ and $y = Home(a)$.

We say that a fault occurring during the execution of Algorithm `RV-RF` is *repaired* if either the execution of phase Correction following this fault has been completed without any occurrence of a fault, or the execution of phase Correction following this fault has been interrupted by a fault that has been repaired. Intuitively, this recursive definition says that a fault is repaired if it is followed by a series of partial executions of phase Correction, each except the last one interrupted by a fault, and the last one not interrupted by a fault and executed completely.

The following two lemmas show that after the occurrence of event 1 or event 3 from Lemma 4 a meeting of the agents at some node must happen within $O(\max(\log \lambda_1, \log \lambda_2))$ steps in phase Progress.

Lemma 5. *Consider two agents a and b, with different labels λ_1 and λ_2, respectively, starting at arbitrary different nodes of an n-node graph, where n is unknown to the agents. Suppose that in some round t' of the execution of Algorithm* `RV-RF` *in the model $\mathcal{M}$ we have $Home(a) = Cot(b)$ and $Home(b) = Cot(a)$. Then a meeting occurs at some node after $O(\max(\log \lambda_1, \log \lambda_2))$ steps of the phase Progress executed after round t'.*

Proof. Suppose, without loss of generality, that round t' is the first round for which $Home(a) = Cot(b) = y$ and $Home(b) = Cot(a) = x$. This implies that in some round $t \leq t'$ one of the agents, say a, came from x to y by applying a step of procedure `Asynch` and the following conditions are satisfied:

- agent a has not performed the next step of procedure `Asynch` in the time interval $[t, t']$;
- agent b came from y to x by applying a step of procedure `Asynch` in round t'.

If agent a completed the procedure `Dance` following its last step of `Asynch` before round t', then a meeting occurs in round $t' - 1$, as both agents are at node y in this round. Hence in the sequel we suppose that agent a has not yet completed this procedure `Dance` in round $t' - 1$.

Let $t'' \geq t'$ be the first round in which one of the agents completes its procedure `Dance`. First suppose that t' and t'' are rounds in which none of the agents is subject to a fault and in which none of the agents executes a step of phase Correction. This means that if a fault occurred in the time interval (t, t'), then it must have been repaired in this time interval.

Claim 1. If no fault occurs in the time interval (t', t''), then a meeting occurs at some node by round t''.

In order to prove the claim, first suppose that $t = t'$. In this case the agents start executing procedure `Dance` simultaneously. By the definition of modified labels, there is an index i such that bits at position i of the modified labels of the agents differ. Hence by round t'' one of the agents is inert at one of the nodes x or y while the other traverses the edge $\{x, y\}$. This implies a meeting at x or at y.

Next suppose that $t < t'$ and consider 3 cases.

Case 1. In round t' agent a has not yet finished Stage 1 of procedure `Dance`.

In round $t' - 1$ both agents were at node y, hence the meeting occurred.

Case 2. In round t' agent a has already finished Stage 1 of procedure `Dance` but has not yet finished Stage 2.

Between round $t' + 1$ and round $t' + 10$ agent b executes Stage 1 of procedure `Dance` and hence is inert at x.

Subcase 2.1. Agent a spends all the time interval $[t' + 1, t' + 10]$ in Stage 2 of procedure `Dance`.

By Lemma 1, in at least one of these rounds agent a traverses the edge $\{x, y\}$, while agent b is inert at x during all these rounds. Hence the meeting must occur at x.

Subcase 2.2. Agent a starts Stage 3 of procedure `Dance` in the time interval $[t'+1, t'+10]$.

Hence during round $t'+10$ agent a executes Stage 3 of procedure `Dance` and hence traverses edge $\{x, y\}$, while agent b is inert at x during this round. Hence the meeting must occur at x.

Case 3. In round t' agent a has already finished Stage 2 of procedure `Dance`.

Subcase 3.1. In round t' agent a has not yet finished Stage 3 of procedure `Dance`.

If in round t' agent a is at x, then the meeting occurs in this round. Otherwise, agent a moves to x in round $t'+1 \leq t''$ and the meeting occurs at x in this round.

Subcase 3.2. In round t' agent a finished Stage 3 of procedure `Dance`.

Agent b started Stage 3 of its preceding procedure `Dance` at node y one round before agent a started Stage 3 at y. When agent b starts treating the last bit 0 of its transformed label and hence waits at node y in round $t'-13$, agent a has not finished treating its penultimate bit 1 and hence traverses edge $\{x, y\}$ in this round. If agent a is at x in round $t'-14$, this implies a meeting at y in round $t'-13$. If agent a is at y in round $t'-14$, this implies a meeting at y in round $t'-14$, as b was at y in this round.

This concludes the proof of Claim 1.

Claim 2. If some faults occur in the time interval (t', t''), then a meeting occurs at some node by round t''.

Consider two cases.

Case 1. If an agent is subject to a fault in some round in the time interval (t', t''), then the other agent is also subject to a fault in this round.

In this case the rounds of executing the phase Correction are the same for both agents, hence we can "delete" them and reduce the situation to the case when no faults occur in the time interval (t', t''). The claim follows from Claim 1.

Case 2. There exists a round in the time interval (t', t''), in which exactly one agent is subject to a fault.

Let r be the last round in the time interval (t', t''), in which exactly one agent is subject to a fault. Denote by B the time interval $[r+1, r+20]$ and let f be the agent subject to a fault in round r. Note that B is necessarily included in the time interval (t', t''), since we first assume in the beginning of the proof of this lemma that if a fault occurred in the time interval (t, t'), then it must have been repaired in this time interval.

During the time interval B the agent f is inert because these are the first 20 rounds of phase Correction, and hence it is not subject to any faults. B is included in the time interval (t', t'') because any fault occurring in the time interval (t', t'') must be repaired in this time interval. If the other agent f' executes procedure `Dance` during all rounds of B, then a meeting must occur at some node, because this agent must move at least once in the time interval B by Lemma 2. Hence we may assume that there exist rounds in B in which f' does not execute procedure `Dance`.

Agent f' cannot be subject to a fault in the time interval B because it would be subject to such a fault alone, contradicting the definition of round r. Moreover, agent f' cannot be subject to a fault in round r by the definition of this round. Hence agent f' must execute during some rounds of B a part of phase Correction caused by a fault occurred before round r. This implies that agent f' executes some round of Stage 2 or Stage 3 of phase Correction during the time interval B. Since Stage 2 and Stage 3 contain only steps consisting in traversing edge $\{x, y\}$, the agents must meet at some node by the end of time interval B.

This concludes the proof of Claim 2.

The two claims imply that if t' and t'' are rounds in which none of the agents is subject to a fault and in which none of the agents executes a step of phase Correction, then a meeting must occur at some node by round t''. It remains to consider the case when this condition is not satisfied. Note that in rounds t' and t'' at least one of the agents moves. Hence if a fault occurs in one of these rounds, this means that one of the agents traverses edge $\{x, y\}$ while the other agent is idle at x or at y. This implies a meeting at some node by round t''.

Hence we may assume that an agent executes a step of phase Correction either in round t' or in round t''.

Case 1. An agent executes a step of phase Correction in round t'.

In this case agent a executes a step of phase Correction in round t' while agent b comes to x from y executing a step of procedure `Asynch`. Suppose, for contradiction, that the agents do not meet at some node by round t''. In round t' agent a must move, otherwise a meeting occurs at some node either in round $t'-1$ or in round t'. Moreover, in round $t'+1$ agent a must attempt to move. Indeed, either agent a has not finished the phase Correction in round t', in which case it continues a moving attempt in round $t'+1$, or it finished the phase Correction in round t', in which case in round $t'+1$ it resumes phase Progress where it was interrupted by the last fault, i.e., it also attempts to move.

If agent a is not subject to a fault in round $t'+1$, then it traverses edge $\{x, y\}$ in this round. In round $t'+1$ agent b executes the first round of procedure `Dance`, hence it is idle at x. This implies that a meeting occurs at some node either in round t' or in round $t'+1$. Hence we may assume that agent a is subject to a fault in round $t'+1$. In the time interval $[t'+2, t'+21]$ agent a executes the first 20 rounds of phase Correction following this fault, i.e., it remains idle. On the other hand, agent b completes Stage 1 of procedure `Dance` (which is a waiting period) in round $t'+10$ and makes an attempt to move in round $t'+11$. If it is not subject to a fault in this round, a meeting at some node must occur. Hence we may assume that agent b is subject to a fault in round $t'+11$. This implies that in the time interval $[t'+12, t'+31]$ agent b executes the first 20 rounds of phase Correction following this fault, i.e., it remains idle. On the other hand agent a completes Stage 1 of phase Correction in round $t'+21$ and makes an attempt to move in round $t'+22$. If it is not subject to a fault in this round, a meeting at some node must occur, because b is idle in this round. Hence we may assume that a is subject to a fault in round $t'+22$ and executes the first 20

rounds of phase Correction following this fault in the time interval $[t'+23, t'+42]$, i.e., it remains idle in this time interval. Continuing this reasoning we conclude that none of the agents can finish its procedure `Dance` by round t'', which is a contradiction.

Case 2. An agent executes a step of phase Correction in round t''.

Let f be the agent finishing its procedure `Dance` in round t'' and let f' be the agent executing a step of phase Correction in round t''. Suppose, for contradiction, that the agents do not meet at some node by round t''. Agent f moves in each round of the time interval $[t''-11, t'']$. Indeed, agent f could not be subject to a fault in one of these rounds or execute Stage 1 of the phase Correction, because then it could not finish procedure `Dance` in round t''. For the same reasons, agent f could not execute any steps of Stage 1 or Stage 2 of procedure `Dance`. Hence in these rounds agent f either executes the entire Stage 3 of procedure `Dance`, or executes some rounds of Stage 2 or 3 of phase Correction and then at least one round of procedure `Dance`. Hence in all these rounds the agent must move. It follows that in each round of the time interval $[t''-11, t'']$ agent f' also moves, for otherwise there would be a meeting at some node by round t''. Hence agent f' executes rounds of Stage 2 or Stage 3 of phase Correction in this time interval. It follows that in round $t''-21$ agent f' is idle executing a step of Stage 1 of phase Correction. There are two subcases.

Subcase 2.1. Agent f does not terminate any phase Correction in the time interval $[t''-15, t'']$.

During this time interval agent f executes exclusively steps of procedure `Dance`, which it finishes in round t''. Hence in round $t''-12$ agent f executes the last step of Stage 2 of this procedure, in which it is idle. Agent f' must be also idle during this round, otherwise a meeting occurs at some node. Since in round $t''-11$ agent f' moves executing a step of the phase Correction, this implies that in round $t''-12$ agent f' executes the last step of Stage 1 of phase Correction. Hence f' is idle in round $t''-15$. In this round agent f treats the penultimate bit of its modified label in Stage 2 of procedure `Dance`. This bit is 1. Hence agent f traverses edge $\{x, y\}$ in round $t''-15$. This implies a meeting at some node in round $t''-16$ or $t''-15$, which is a contradiction.

Subcase 2.2. Agent f terminates a phase Correction in the time interval $[t''-15, t'']$.

If this happens in round $t''-2$ or earlier, then in round $t''-21$ agent f executes a step of Stage 2 or Stage 3 of phase Correction. Hence in this round agent f traverses edge $\{x, y\}$, while agent f' is idle. This implies a meeting at some node in round $t''-22$ or $t''-21$, which is a contradiction. Since agent f cannot end phase Correction in round t'' by definition of this round, we may assume that it terminates phase Correction in round $t''-1$. If f executed 21 rounds of movement in Stage 2 and Stage 3 of phase Correction, then a meeting at some node must occur by round $t''-21$. Otherwise agent f executed exactly 20 moves in phase Correction and hence was subject to a fault in round $t''-41$. Consider two possibilities. If both agents are always subject to faults simultaneously in the time interval $[t', t'']$, then both agents execute steps of phase Correction in the

same rounds in the time interval $[t', t'']$, which contradicts the fact that in round t'' agent f executes the last step of procedure `Dance` and agent f' executes a step of phase Correction in this round. If agents are not always subject to faults simultaneously in the time interval $[t', t'']$, then the contradiction is obtained using an argument similar to that from Case 2 in the proof of Claim 2.

Hence we proved that a meeting at some node must occur by round t''. Since in the time interval $[t', t'']$ both agents executed $O(\max(\log \lambda_1, \log \lambda_2))$ steps of the phase Progress, the proof is complete. □

The proof of the next lemma is analogous to that of Lemma 5, hence we omit it.

Lemma 6. *Consider two agents a and b, with different labels λ_1 and λ_2, respectively, starting at arbitrary different nodes of an n-node graph, where n is unknown to the agents. Suppose that in some round t' of the execution of Algorithm* `RV-RF` *in the model $\mathcal{M}$ we have $Home(a) = Home(b)$ and $Cot(a) = Cot(b)$. Then a meeting occurs at some node after $O(\max(\log \lambda_1, \log \lambda_2))$ steps of the phase Progress executed after round t'.*

The last lemma of this section shows that the meeting between agents in model $\mathcal{M}$ is guaranteed after a polynomial number of steps in phase Progress.

Lemma 7. *Consider two agents a and b, with different labels λ_1 and λ_2, respectively, starting at arbitrary different nodes of an n-node graph, where n is unknown to the agents. There exists a polynomial B such that a meeting at some node is guaranteed after the execution of a total of $B(n, \max(\log \lambda_1, \log \lambda_2))$ steps of phase Progress of Algorithm* `RV-RF` *by both agents in the model $\mathcal{M}$.*

Proof. Each stage of phase Progress of Algorithm `RV-RF` consists of one step of procedure `Asynch` and one execution of procedure `Dance`, and the execution time of procedure `Dance` is logarithmic in the label of the executing agent. Hence after executing a total of $A(n, \min(\log \lambda_1, \log \lambda_2))$ stages of phase Progress the agents with labels λ_1 and λ_2 executed a total of at most

$$B^*(n, \max(\log \lambda_1, \log \lambda_2)) = A(n, \min(\log \lambda_1, \log \lambda_2)) \cdot c \cdot \max(\log \lambda_1, \log \lambda_2)$$

steps of phase Progress, for some positive constant c. By Lemma 4, after executing a total of $A(n, \min(\log \lambda_1, \log \lambda_2))$ stages of phase Progress by both agents, one of the 5 events must have occurred.

If event 1 occurred, the conclusion follows from Lemma 5 for

$$B(n, \max(\log \lambda_1, \log \lambda_2)) = B^*(n, \max(\log \lambda_1, \log \lambda_2)) + O(\max(\log \lambda_1, \log \lambda_2))(*).$$

If event 2 occurred, the conclusion is immediate. If event 3 occurred, the conclusion follows from Lemma 6, for B given by $(*)$. If event 4 occurred, there are two cases.

Case 1. Both agents leave node v in the same round applying a step of procedure `Asynch`.

In this case the agents were together at v in the previous round, hence rendezvous occurred and the conclusion follows from the fact that each agent has executed at most $c \cdot \max(\log \lambda_1, \log \lambda_2)$ steps of procedure `Dance` since it arrived at v.

Case 2. One of the agents leaves node v applying a step of procedure `Asynch` in a round m before the other agent leaves node v applying a step of procedure `Asynch`.

In this case in round $m+1$ we have $Home(a) = Cot(b)$ and $Home(b) = Cot(a)$ and hence the conclusion follows from Lemma 5, for B given by $(*)$.

Finally, if event 5 occurred, there are two cases.

Case 1. Both agents leave node v in the same round applying a step of procedure `Asynch`.

In this case the agents were together at v in the previous round, hence rendezvous occurred and the conclusion follows from the fact that each agent has executed at most $c \cdot \max(\log \lambda_1, \log \lambda_2)$ steps of procedure `Dance` since it arrived at v.

Case 2. Agent a leaves node v applying a step of procedure `Asynch` in a round m before agent b leaves node v applying a step of procedure `Asynch`.

In this case in round $m+1$ we have $Home(a) = Cot(b)$ and $Home(b) = Cot(a)$ and hence the conclusion follows from Lemma 5, for B given by $(*)$. □

We are now ready to prove the main result of this section, showing that Algorithm `RV-RF` achieves rendezvous at polynomial cost with very high probability, under the random fault model.

Theorem 1. *Consider two agents a and b, with different labels λ_1 and λ_2, respectively, starting at arbitrary different nodes of an n-node graph, where n is unknown to the agents. Suppose that delay faults occur randomly and independently with constant probability $0 < p < 1$ in each round and for each agent. Algorithm* `RV-RF` *guarantees rendezvous of the agents with probability 1. Moreover, there exists a polynomial B such that rendezvous at some node occurs at cost $\tau = O(B(n, \max(\log \lambda_1, \log \lambda_2)))$ with probability at least $1 - e^{-O(\tau)}$.*

Proof. An entire execution of phase Correction of Algorithm `RV-RF`, if no fault occurs during it, lasts at most 41 rounds. Hence in a segment of 42 consecutive rounds without a fault an agent makes at least one step of phase Progress. Call a segment of 42 consecutive rounds without a fault occurring to a given agent *clean* for this agent. For any round r the probability that the segment starting at r is clean for both agents is at least $q = (1-p)^{84}$. Let B be the polynomial from Lemma 7. The probability of the existence of $B(n, \max(\log \lambda_1, \log \lambda_2))$ pairwise disjoint clean segments is 1, and this event implies rendezvous by Lemma 7.

Let $c = \lceil 84/q \rceil$. Consider $\tau = c \cdot B(n, \max(\log \lambda_1, \log \lambda_2))$ rounds of execution of Algorithm `RV-RF`. Partition these rounds into $\lceil 2/q \rceil B(n, \max(\log \lambda_1, \log \lambda_2))$ pairwise disjoint segments of length 42. By Chernoff bound, there are at least $B(n, \max(\log \lambda_1, \log \lambda_2))$ clean segments among them, with probability $1 - e^{-O(\tau)}$. This concludes the proof. □

4 Unbounded adversarial faults

In this section we consider the scenario when the adversary can delay each of the agents for any finite number of consecutive rounds. Under this scenario the time (number of rounds until rendezvous) depends entirely on the adversary, so the only meaningful measure of efficiency of a rendezvous algorithm is its cost. However, it turns out that, under this harsh fault scenario, even feasibility of rendezvous is usually not guaranteed, even for quite simple graphs. Recall that we *do not* assume knowledge of any upper bound on the size of the graph.

In order to prove this impossibility result, we introduce the following terminology. For any rendezvous algorithm $\mathcal{A}$ in a graph G, a *solo* execution of this algorithm by an agent A is an execution in which A is alone in the graph. Note that, in an execution of algorithm $\mathcal{A}$ where two agents are present, the part of this execution by an agent before the meeting is the same as the respective part of a solo execution of the algorithm by the respective agent.

A port labeling in a graph is called *homogeneous*, if port numbers at both endpoints of each edge are equal.

Remark 1. Let $\mathcal{F}$ be a family of regular graphs of degree d, with homogeneous port labeling. Consider a solo execution of a rendezvous algorithm $\mathcal{A}$ by an agent A_λ with label λ, in a graph $G \in \mathcal{F}$. In each round the agent may either decide to stay at the current node or try to traverse an edge. This decision depends on the label λ and on the history of the agent. In view of the regularity of the graph, and since the port labeling is homogeneous, the agent can differentiate neither G from any graph $G' \in \mathcal{F}$, nor any two nodes in G, during its navigation. Hence, if the algorithm, the agent's label and the adversary's behavior are fixed, the agent's history, in any round, (which can be coded by the sequence of previous round numbers in which the agent made a move) is necessarily the same in any graph of $\mathcal{F}$ and for any starting node. (The agent made a move in a previous round, if and only if, it tried to traverse an edge and the adversary allowed the move by not imposing a fault in this round.)

Consider the decisions of agent A_λ starting in some round t and before its next move. We say that the agent *attacks* since round t, if, from round t on, it tries an edge traversal in each round until it makes the next move. Note that, within a given attack, the edge which the attacking agent attempts to traverse, is not always necessarily the same. We say that the agent does not attack after round t, if there does not exist a round $t' > t$ in which the agent starts attacking. In other words it means that, after round t, the agent decides to stay at the current node in rounds with arbitrarily large numbers, interleaved with possible attempts at a move, all of which can be prevented by an adversary. Note that an adversary must eventually allow an agent that attacks to make a next move but it is capable of preventing an agent that never attacks after a given round from making any further move by imposing faults in all rounds in which the agent tries an edge traversal.

We first prove the following technical lemma.

Lemma 8. *Let $\mathcal{F}$ be a family of regular graphs of degree d, with homogeneous port labeling, $\mathcal{A}$ any rendezvous algorithm working for graphs in $\mathcal{F}$, and M any positive integer. Then there exists a behavior of the adversary inducing an infinite increasing sequence $\tau_1, \tau_2, \ldots, \tau_i, \ldots$ of positive integers, such that, for any graph $G \in \mathcal{F}$ and for any $\lambda \leq M$, there exists a solo execution of algorithm $\mathcal{A}$ in G by agent A_λ with label λ, such that one of the following conditions is satisfied:*

- *either A_λ attacks only a finite number $k \geq 0$ of times, in which case A_λ either does not move if $k = 0$, or moves exactly in rounds $\tau_1, \tau_2, \ldots, \tau_k$ otherwise;*
- *or A_λ attacks infinitely many times, in which case it moves exactly in rounds τ_i, for all positive integers i.*

Proof. Fix a rendezvous algorithm $\mathcal{A}$, a degree d of graphs in a family $\mathcal{F}$, and a positive integer M. Consider the set S of agents A_λ with label $\lambda \leq M$, and suppose that each of these agents executes $\mathcal{A}$ alone in some graph $G \in \mathcal{F}$. Note that, for a given behavior of an adversary and a given label, this execution will be the same in any graph $G \in \mathcal{F}$ (cf. Remark 1). We construct the sequence $\tau_1, \tau_2, \ldots, \tau_i, \ldots$ and the behavior of an adversary by induction on i. We start by giving a partial description of this behavior. For any agent in S, the adversary imposes a fault in every round in which the agent does not attack but tries to traverse an edge. An adversary behaving in this way will be called *tough*.

If an agent does not attack before its first move, then the agent never moves, in view of the toughness of the adversary. Consider the set $S_0 \subseteq S$ of agents that attack before their first move. Let $s_1 < \cdots < s_r$ be the rounds in which this attack starts for some agent in S_0. Define $\tau_1 = s_r$. The adversary allows a move of all agents from the set S_0 in round τ_1.

For the inductive step, consider the set $S_{i-1} \subseteq S$ of agents that attacked $i > 0$ times and made i moves, exactly in rounds $\tau_1, \tau_2, \ldots, \tau_i$. If an agent from S_{i-1} does not attack after its i-th move in round τ_i, then the agent never moves again, in view of the toughness of the adversary. Consider the set $S_i \subseteq S_{i-1}$ of agents that attack after round τ_i and before the next move. Let $t_1 < \cdots < t_p$, where $t_1 > \tau_i$, be the rounds in which the $(i+1)$-th attack starts for some agent in S_i. Define $\tau_{i+1} = t_p$. The adversary allows a move of all agents from the set S_i in round τ_{i+1}.

This concludes the definition of the sequence $\tau_1, \tau_2, \ldots, \tau_i, \ldots$, and of the behavior of the adversary, by induction on i. In order to prove that one of the conditions in the lemma must be satisfied, consider the solo execution $\mathcal{E}_\lambda$ of an agent A_λ with label $\lambda \leq M$, in some graph $G \in \mathcal{F}$, against the above defined adversary. There are two possible cases: either, in the execution $\mathcal{E}_\lambda$, the agent attacks k times, where k is a non-negative integer, or it attacks infinitely many times. By the definition of the adversary, in the first case the agent does not move or moves exactly in rounds $\tau_1, \tau_2, \ldots, \tau_k$, and in the second case it moves exactly in rounds τ_i, for all positive integers i. □

We now establish the impossibility result for the model with unbounded faults.

Theorem 2. *Rendezvous with unbounded adversarial faults is not feasible, even in the class of rings.*

Proof. Let $\mathcal{F}$ be the family of rings of even size with homogeneous port numbering. Since graphs in $\mathcal{F}$ are regular graphs of degree 2, Lemma 8 applies. Suppose that $\mathcal{A}$ is a rendezvous algorithm working for the family $\mathcal{F}$, and let $M = 3$. Let $X = (\tau_1, \tau_2, \ldots, \tau_i, \ldots)$ be the sequence of integers given by Lemma 8, and consider the solo execution $\mathcal{E}_j$ of agent A_j with label j, for $j = 1, 2, 3$, corresponding to X and satisfying one of the two conditions of Lemma 8. (For a fixed $j \leq 3$, this execution is the same in each ring from $\mathcal{F}$, regardless of the starting node.) Consider two cases.

Case 1. At least two among agents A_j attack infinitely many times in execution $\mathcal{E}_j$.

Without loss of generality, assume that A_1 and A_2 attack infinitely many times, each in their solo execution $\mathcal{E}_1$ (resp. $\mathcal{E}_2$). Consider the execution $\mathcal{E}$ in which agents A_1 and A_2 start simultaneously at an odd distance in the graph, and the adversary acts against each of them as in their respective solo execution $\mathcal{E}_1$ and $\mathcal{E}_2$. In execution $\mathcal{E}$, both agents make moves in exactly the same rounds, i.e., the rounds of sequence X. Since an even ring is bipartite, the parity of their distance never changes, and they remain at an odd distance forever. Hence they never meet, which is a contradiction.

Case 2. At least two among agents A_j attack finitely many times in execution $\mathcal{E}_j$.

Without loss of generality, assume that A_1 and A_2 attack finitely many times, each in their solo execution $\mathcal{E}_1$ (resp. $\mathcal{E}_2$): agent A_1 attacks k times and agent A_2 attacks k' times. Choose as the starting nodes of these agents antipodal nodes in the ring of size $2(k+k'+1)$ (i.e., at distance $k+k'+1$), start them simultaneously, and assume that the adversary acts against each of them as in their respective solo execution $\mathcal{E}_1$ and $\mathcal{E}_2$. By Lemma 8, agent A_1 will make a total number of k moves, and agent A_2 will make a total number of k' moves, in this execution. Hence they never meet, which is a contradiction. □

In view of Theorem 2, it is natural to ask if rendezvous with unbounded adversarial faults can be accomplished in the class of connected graphs not containing cycles, i.e., in the class of trees, and if so, at what cost it can be done. The rest of this section is devoted to a partial answer to this problem. We start with an auxiliary result about rendezvous in oriented rings, under the *additional* assumption that the size of the ring is known to the agents. This result will be used in a special situation when the size of the ring can be *learned* by the agents at small cost. By an oriented ring we mean a ring in which every edge has ports 0 and 1 at its extremities. This provides the agents with the orientation of the ring: they can both go in the same direction by choosing port 0 at each node.

Lemma 9. *If the size n of an oriented ring is known to the agents, then rendezvous with unbounded adversarial faults can be achieved at cost $O(n\ell)$, where ℓ is the smaller label.*

Proof. The rendezvous algorithm for an agent with label λ is the following: start by port 0, perform $2n\lambda$ edge traversals in the same direction, and stop. If two agents execute this algorithm, then, if they have not met before, the agent with larger label does at least one full tour of the ring after the agent with smaller label already stopped. Hence they must meet at cost at most $4(\ell+1)n$, where ℓ is the smaller label. □

Our goal is to present an efficient rendezvous algorithm working for arbitrary trees. We will use the following notion. Consider any tree T. A *basic walk* in T, starting from node v is a traversal of all edges of the tree ending at the starting node v and defined as follows. Node v is left by port 0; when the walk enters a node by port i, it leaves it by port $(i+1)$ mod d, where d is the degree of the node. Any basic walk consists of $2(n-1)$ edge traversals. An agent completing the basic walk knows that this happened and learns the size n of the tree and the length $2(n-1)$ of the basic walk.

The following Algorithm `Tree-RV-UF` (for rendezvous in trees with unbounded faults) works for an agent with label λ, starting at an arbitrary node of any tree T.

Algorithm Tree-RV-UF
Repeat 2λ basic walks starting from the initial position and stop. ◇

Theorem 3. *Algorithm* `Tree-RV-UF` *is a correct rendezvous algorithm with unbounded adversarial faults in arbitrary trees, and works at cost $O(n\ell)$, where n is the size of the tree and ℓ is the smaller label.*

Proof. Fix any nodes v_1 and v_2 of T, which are the starting positions of the agents. Notice that agents repetitively performing a basic walk starting at v_1 and v_2 respectively, traverse all edges of the tree in the same order and in the same direction, with a cyclic shift. Hence performing a repetitive basic walk in a tree of size n can be considered as making tours of a "virtual" oriented ring of length $2(n-1)$ composed of edges in the order and direction imposed by the basic walk. Each edge of the tree is traversed exactly twice in each tour of this virtual ring (once in each direction) and the direction of the walk in this virtual ring is the same, regardless of the node where the basic walk starts. Since, after completing the first basic walk, the agent learns its length, and Algorithm `Tree-RV-UF` is equivalent to the algorithm from the proof of Lemma 9 providing rendezvous in oriented rings of known size (run on the virtual oriented ring given by the basic walk of the tree), the conclusion follows. □

We do not know if Algorithm `Tree-RV-UF` has optimal cost, i.e., if a lower bound $\Omega(n\ell)$ can be proved on the cost of any rendezvous algorithm with unbounded adversarial faults, working in arbitrary trees of size n. However, we prove a weaker lower bound. It is clear that no algorithm can beat cost $\Theta(n)$ for rendezvous in n-node trees, even without faults. Our next result shows that, for unbounded adversarial faults, $\Omega(\ell)$ is a lower bound on the cost of any rendezvous algorithm, even for the simplest tree, that of two nodes.

Proposition 1. *Let T be the two-node tree. Every rendezvous algorithm with unbounded adversarial faults, working for the tree T, has cost $\Omega(\ell)$, where ℓ is the smaller label.*

Proof. Let M be a positive integer, and $\mathcal{A}$ a rendezvous algorithm working in the tree T. For every $j \in \{1, \ldots, M\}$, consider the solo execution $\mathcal{E}_j$ of $\mathcal{A}$, for agent A_j with label j, and for the adversary constructed in Lemma 8. Let $\mathcal{E}(i, j)$ be the execution of $\mathcal{A}$ in T, for agents A_i and A_j starting simultaneously from the two nodes of T, for the same adversary. By Lemma 8, for at most one value of $j \in \{1, \ldots, M\}$, agent A_j attacks infinitely many times in execution $\mathcal{E}_j$, because if there were two such agents A_i and A_j, then they would not meet in execution $\mathcal{E}(i, j)$. Hence, for at least $M - 1$ values of $j \in \{1, \ldots, M\}$, agent A_j attacks a finite number of times in execution $\mathcal{E}_j$. Again by Lemma 8, for all these values of j, agent A_j must attack a different number of times in execution $\mathcal{E}_j$, because if there were two agents A_i and A_j attacking the same number k of times, then they would not meet in execution $\mathcal{E}(i, j)$. Hence, for at least two different labels $\ell < L \leq M$ this number of attacks must be at least $M - 2$. It follows from Lemma 8 that in execution $\mathcal{E}(\ell, L)$ each of the agents makes at least $M - 2$ moves before rendezvous. Since $\ell \leq M - 1$, agent A_ℓ makes at least $\ell - 1$ moves before rendezvous, which completes the proof. □

5 Bounded adversarial faults

In this section we consider the scenario when the adversary can delay each of the agents for at most c consecutive rounds, where c is a positive integer, called the *fault bound*. First note that if c is known to the agents, then, given any synchronous rendezvous algorithm working without faults for arbitrary networks, it is possible to obtain an algorithm working for bounded adversarial faults and for arbitrary networks, at the same cost. Let $\mathcal{A}$ be a synchronous rendezvous algorithm for the scenario without faults, working for arbitrary networks. Consider the following algorithm $\mathcal{A}(c)$ working for bounded adversarial faults with parameter c. Each agent replaces each round r of algorithm $\mathcal{A}$ by a segment of $2c + 1$ rounds. If in round r of algorithm $\mathcal{A}$ the agent was idle, this round is replaced by $2c + 1$ consecutive rounds in which the agent is idle. If in round r the agent left the current node by port p, this round is replaced by a segment of $2c + 1$ rounds in each of which the agent makes an attempt to leave the current node v by port p until it succeeds, and in the remaining rounds of the segment it stays idle at the node adjacent to v that it has just entered.

We associate the first segment of the later starting agent with the (unique) segment of the earlier agent that it intersects in at least $c + 1$ rounds. Let it be the ith segment of the earlier agent. We then associate the jth segment of the later agent with the $(j + i - 1)$th segment of the earlier agent, for $j > 1$. Hence, regardless of the delay between starting rounds of the agents, corresponding segments intersect in at least $c+1$ rounds. If the agents met at node x in the jth round of the later agent, according to algorithm $\mathcal{A}$, then, according to algorithm

$\mathcal{A}(c)$, in the last $c+1$ rounds of its jth segment the later agent is at x and in the last $c+1$ rounds of its $(j+i-1)$th segment the earlier agent is at x. Since these segments intersect in at least $c+1$ rounds, there is a round in which both agents are at node x according to algorithm $\mathcal{A}(c)$, regardless of the actions of the adversary, permitted by the bounded adversarial fault scenario. This shows that algorithm $\mathcal{A}(c)$ is correct. Notice that the cost of algorithm $\mathcal{A}(c)$ is the same as that of algorithm $\mathcal{A}$, because in each segment corresponding to an idle round of algorithm $\mathcal{A}$, an agent stays idle in algorithm $\mathcal{A}(c)$ and in each segment corresponding to a round in which an agent traverses an edge in algorithm $\mathcal{A}$, the agent makes exactly one traversal in algorithm $\mathcal{A}(c)$.

In the rest of this section we concentrate on the more difficult situation when the fault bound c is unknown to the agents. The following Algorithm `Graph-RV-BF` (for rendezvous in graphs with bounded faults) works for an agent with label λ starting at an arbitrary node of any graph.

Algorithm `Graph-RV-BF` is divided into phases. The i-th phase is composed of 2^i stages, each lasting $s_i = 2^{i+4}$ rounds. Hence the i-th phase lasts $p_i = 2^{2i+4}$ rounds. The λ-th stage of the i-th phase consists of two parts: the *busy* part of $b_i = 3 \cdot 2^i$ rounds and the *waiting* part of $w_i = 13 \cdot 2^i$ rounds. During the busy part of the λ-th stage of phase i, the agent tries to explore the graph three times (each exploration attempt lasts at most $e_i = 2^i$ rounds), using a UXS. We say that the agent is *active* during the busy part of the λ-th stage of each phase $i \geq q = \lceil \log(\lambda + 1) \rceil$. In order to explore the graph, the agent keeps estimates of the values of c and $P(n)$. (Recall that the latter is the length of a UXS that allows to traverse all edges of any graph of size at most n, starting from any node). The values of these estimates in phase i are called c_i and u_i, respectively, and grow depending on the strategy of the adversary. For the first phase q in which the agent is active, we set $u_q = 1$ and $c_q = 2^q$. In phase i the agent uses the UXS of length u_i. Call this sequence S. The agent uses this UXS proceeding by steps. Steps correspond to terms of the sequence S. During phase i, the k-th step consists of s_i rounds during which the agent tries to move, using port $(p + S[k] \mod d)$ (where d is the degree of the current node and p is the port by which the agent entered the current node), until it succeeds or until the s_i rounds of the k-th step are over. If it succeeded to move, it waits until the s_i rounds of the step are over. If the agent succeeds to perform all of its three UXS explorations during a phase i, i.e., if it succeeds to move once in each step, then we set $u_{i+1} = 2u_i$ and $c_{i+1} = c_i$. Otherwise, we set $u_{i+1} = u_i$ and $c_{i+1} = 2c_i$. When the agent is not active, it waits at its current node. The agent executes this algorithm until it meets the other agent.

Below we give the pseudocode of the algorithm that works for an agent with label λ starting at an arbitrary node of any connected graph.

Algorithm Graph-RV-BF

$q := \lceil \log(\lambda + 1) \rceil$;
$c_q := 2^q$;
$u_q := 1$;
$i := 0$;

```
while rendezvous not achieved do
    for j := 0 to 2^i − 1 do
        if λ = j then
            success := true;
            for r := 0 to 2 do
                success := (success AND exploration(u_i, c_i));
                /*the value of exploration(u_i, c_i)) may be different
                in different iterations of the loop, due to the actions
                of the adversary*/
            endfor
            if success then
                u_{i+1} := 2u_i; c_{i+1} := c_i;
            else
                u_{i+1} := u_i; c_{i+1} := 2c_i;
            endif
            wait for 13 · 2^i rounds;
        else
            wait for 2^{i+4} rounds;
        endif
    endfor
    i := i + 1;
endwhile                                                          ◇
```

We use the following function for exploration.

```
boolean exploration(integer u, integer c)
success := true;
S := UXS of length u;
for k := 0 to u − 1 do
    moved := false;
    for r := 0 to c − 1 do
        p := port by which the agent entered the current node;
        d := degree of the current node;
        try to move by port (p + S[k]  mod d);
        if move is successful then
            moved := true; break;
        endif
    endfor
    wait for c − 1 − r rounds;
    if moved = false then
        success := false; break;
    endif
endfor
wait until the call of exploration lasts for 2^{cu} rounds;
return success;                                                   ◇
```

Theorem 4. *Algorithm* `Graph-RV-BF` *is a correct rendezvous algorithm with bounded adversarial faults in arbitrary graphs, and works at cost polynomial in*

the size n of the graph, and logarithmic in the fault bound c and in the larger label L.

Proof. Let a with label λ be the first agent to be activated by the adversary. The second agent a' with label λ' is activated after $\delta \geq 0$ rounds. Let S_i (respectively S'_i) be the λ-th (respectively λ'-th) stage of phase i of execution of agent a (respectively agent a'). The stage S_i (respectively S'_i) is called the *active* stage of agent a (respectively a') for phase i. Recall that agent a (respectively agent a') is *active* in round t if it is executing a busy part of an active stage S_i (respectively S'_i). Let t_k (respectively t'_k) be the round in which stage S_k (respectively S'_k) starts. We say that stage S_i of agent a intersects stage S'_j of agent a' if there is a round during the execution when both agents are active, with agent a executing the busy part of stage S_i and agent a' executing the busy part of S'_j. We say that an active stage of an agent is *useless* if it intersects an active stage of the other agent. We denote by u'_i and c'_i the values of the variable u_i and c_i during the execution of Algorithm `Graph-RV-BF` by agent a'. For simplicity, we denote these values for agent a by u_i and c_i. First, we show the following claim.

Claim 1. At most three active stages of agent a are useless.

In order to prove the claim, we can assume that agent a has at least one useless stage, otherwise the claim is proved. Let S_i be the first useless stage of a. We consider the minimal index $j > i+1$ such that the active stage S_j is useless. If j does not exist then the claim is proved, since only stages S_i and S_{i+1} can be useless. We will show that for $r > j$, stage S_r is not useless. This will prove the claim since in this case only the stages S_i, S_{i+1} and S_j can be useless. Observe that $\delta < \sum_{k=0}^{i} p_k$, since the second agent was active during the phase i of the first agent. For all $k \in \mathbb{N}$, we have $p_{k+1} = 4p_k$. Hence, we have $\sum_{k=0}^{i} p_k < \frac{1}{2} p_{i+1}$. Thus $\delta < \frac{1}{2} p_{k-1}$ for any $k > i+1$. Hence, for any $k > i+1$ the phase k of agent a happens between the beginning of the second half of phase $k-1$ and the end of phase k of agent a'. Observe that agent a' was active during some phase $i' \leq i$ since agent a has a useless stage in phase i. Hence $\lambda' \leq 2^i \leq \frac{1}{2} 2^{k-1}$ for any $k \geq i+1$. It implies that for all $k \geq i+1$ agent a' is not active during the second half of phase $k-1$, and so for any $r < k$, stage S'_r cannot intersect stage S_k. Hence, for any $k > i+1$, S_k can only intersect stage S'_k. In particular, this implies that during phase j, the active stage S_j intersects stage S'_j. Hence $\lambda' < \lambda$. We have $t'_j < t_j + b_j$ since S'_j intersects S_j (recall that S'_j intersects S_j if there exists a round in which agents a and a' are both executing the busy part of S_j and S'_j, respectively). Let $k \geq j$. We have $t_{k+1} = t_k + (2^i - \lambda)s_k + \lambda s_{k+1} = t_k + p_k + \lambda(s_{k+1} - s_k)$. Similarly, we have $t'_{k+1} = t'_k + p_k + \lambda'(s_{k+1} - s_k)$ for any $k \geq j$. We obtain that for any $k \geq j$:

$$\begin{aligned} t_{k+1} - t'_{k+1} &= t_k + p_k + \lambda(s_{k+1} - s_k) - (t'_k + p_k + \lambda'(s_{k+1} - s_k)) \\ &= t_k - t'_k + (\lambda - \lambda')(s_{k+1} - s_k) \\ &\geq t_k - t'_k + (s_{k+1} - s_k), \qquad \text{since } \lambda > \lambda'. \end{aligned}$$

By induction on r, we have for any $r \geq j$ that :

$$\begin{aligned} t_{r+1} - t'_{r+1} &\geq t_j - t'_j + \sum_{k=j}^{r}(s_{k+1} - s_k) \\ &\geq t_j - t'_j + s_{r+1} - s_j \\ &\geq t_j - t'_j + \frac{1}{2}s_{r+1} \quad (\text{since } s_j \leq \tfrac{1}{2}s_{r+1}) \\ &\geq \frac{1}{2}s_{r+1} - b_j \quad (\text{since } t'_j < t_j + b_j) \\ &\geq 2b_{r+1} - b_j \quad (\text{since } s_{r+1} > 4b_{r+1}) \\ &> b_{r+1} \quad (\text{since } b_{r+1} > b_j). \end{aligned}$$

For any $r \geq j$, we conclude that the busy part of stage S'_{r+1} of agent a' starts in round t'_{r+1} and ends in round $t'_{r+1} + b_{r+1}$. Stage S'_{r+1} does not intersect S_{r+1} since $t_{r+1} > t'_{r+1} + b_{r+1}$. Hence, for any $r > j$, stage S_r is not useless. This ends the proof of the claim.

In order to bound the number of moves of the two agents, we will show the following claim.

Claim 2. There exists a constant d, such that, for any integer i, either rendezvous occurs by the end of the execution of phase i by one of the agents, or the values of u_i and u'_i are at most $dP(n)$.

In order to prove the claim, first consider the values of u_i (for agent a). Let $j = \min\{k \in \mathbb{N} \mid u_k \geq P(n)\}$. For $i < j$ we have $u_i \leq u_j \leq 2P(n)$. Let $i \geq j$. Consider two cases: either (1) S_i is not useless or (2) S_i is useless. For Case (1), consider two subcases: either (1.1) the agent succeeds to move in each step of S_i, or (1.2) the agent is blocked by the adversary in some step of S_i. In case (1.1), agent a explores all the graph since it performs a UXS of length u_k for $u_k \geq P(n)$. Hence, rendezvous occurs since agent a' does not move during stage S_i. For Case (1.2), we have $u_{i+1} = u_i$. Since Case (2) can only happen three times by Claim 1, in view of Case (1.2) and of the fact that $u_{z+1} \leq 2u_z$ for all $z \geq 0$, we have that for all i, $u_i \leq 8u_j \leq 16P(n)$. This proves the claim for agent a with $d = 16$.

Next, we consider the values of u'_i (for agent a'). Let $x = \min\{k \in \mathbb{N} \mid u'_k \geq P(n)\}$. For $i < x$ we have $u'_i \leq u'_x \leq 2P(n)$. Let $i \geq x$. Consider two cases: either (1) S'_i is not useless or (2) S'_i is useless. Using the same argument as for agent a, we can assume that the value u'_{i+1} is equal to u'_i in Case (1), otherwise rendezvous occurs in phase i of agent a'. Consider Case (2). We have that S'_i intersects S_j for some $j \geq i$. Let $p = \min\{k \in \mathbb{N} \mid S'_k \text{ intersects } S_j\}$ and $q = \max\{k \in \mathbb{N} \mid S'_k \text{ intersects } S_j\}$. We consider two subcases: (2.1) $q - p > 0$ or (2.2) $q - p = 0$. Consider Subcase (2.1). Stages S'_{q-1} and S'_q intersect stage S_j and we have $t_j \leq t'_{q-1} + b_{q-1} \leq t'_q \leq t_j + b_j$. This implies $t'_q - (t'_{q-1} + b_{q-1}) \leq b_j$. We have $t'_q - t'_{q-1} = p_{q-1} + \lambda'(s_{q+1} - s_q) \geq p_{q-1}$. Hence:

$$
\begin{aligned}
p_{q-1} &\leq b_j + b_{q-1} \\
2^{2(q-1)+4} &\leq 3 \cdot 2^j + 3 \cdot 2^{q-1}, \quad \text{since } p_i = 2^{2i+4} \text{ and } b_i = 3.2^i \text{ for all } i \in \mathbb{N}. \\
2^{2(q-1)+4} &\leq 3 \cdot 2^j + 3 \cdot 2^{j-1}, \quad \text{since } S'_q \text{ intersects } S_j \text{ and } q \leq j. \\
2^{2q} &\leq \frac{9}{8} \cdot 2^j \\
2q &\leq j + \log\left(\frac{9}{8}\right) \\
2q &\leq j + 0.06 \\
2q &\leq j, \quad \text{since both } q \text{ and } j \text{ are integers.}
\end{aligned}
$$

For any $p \leq k \leq q$, we have $2k \leq j$. Now we show that for k such that $p \leq k \leq q$, if $u'_k \geq 48P(n)$, then rendezvous occurs. Assume that $u'_k \geq 48P(n)$. We have :

$$
\begin{aligned}
2^{2k} &\leq 2^j \quad \text{since } 2k \leq j \\
(e_k)^2 &\leq e_j \quad \text{since } \forall i \in \mathbb{N}, e_i = 2^i \\
e_k(c'_k u'_k) &\leq c_j u_j \quad \text{since } \forall i \in \mathbb{N}, e_i = u'_i c'_i = u_i c_i \\
48P(n)e_k c'_k &\leq 16P(n)c_j \quad \text{since } u'_k \geq 48P(n) \text{ and } u_j \leq 16P(n) \\
3e_k &\leq c_j \quad \text{since } c'_k \geq 1.
\end{aligned}
$$

Hence, during a period of time of duration less or equal to c_j, agent a' explores three times the graph. Agent a moves at most twice during this period of time, since this period of time intersects at most two steps and agent a moves at most once in each step. Hence, agent a' explores the graph at least once while agent a is not moving and thus rendezvous occurs. It follows that for k such that $p \leq k \leq q$, if $u'_k \geq 48P(n)$, then rendezvous occurs. Hence, for k such that $p \leq k \leq q$, if rendezvous does not occur by the end of phase k, we have $u'_k \leq 96P(n)$. (By definition of p and q, in Subcase (2.1) it is enough to restrict attention to k in the interval $[p, q]$.)

Finally, let us consider Subcase (2.2). The stage S'_i is the only active stage of a' to intersect S_j. This means that Subcase (2.2) can only occur three times by Claim 1. Hence the value of u_i is less than $\max\{2^3 u_j, 2^2 96P(n)\} \leq 384P(n)$. This proves the claim for agent a' with $d = 384$ and concludes the proof of Claim 2.

We show that rendezvous occurs by the end of the execution of phase $\rho(r) = \log(P(n)) + \log(c) + \log(L) + r$ by agent a, for some constant r. Note that for any phase $i \geq \lceil \log(L) \rceil \geq \lceil \log(\lambda) \rceil$, agent a has an active stage S_i. At the end of each phase, either $u_{i+1} = 2u_i$ or $c_{i+1} = 2c_i$. For all i, if rendezvous has not occurred by the end of phase i, we have $u_i \leq dP(n)$ for some constant d (cf. Claim 2).

Moreover, observe that if $c_j \geq c+1$ then $u_{j+1} = 2u_j$, since the adversary cannot prevent the move of the agent more than c times. Hence, there exists a constant y, such that if rendezvous has not occurred by the beginning of phase $\rho(y)$, then we have $u_{\rho(y)} \geq P(n)$ and $c_{\rho(y)} \geq c+1$. This means that during any active stage S_j with $j \geq \rho(y)$, agent a explores the graph. At least one of the stages among $S_{\rho(y)}$, $S_{\rho(y)+1}$, $S_{\rho(y)+2}$ or $S_{\rho(y)+3}$ is not useless by Claim 1. Hence, rendezvous occurs by the end of the execution of phase $\rho(y)+3 = \rho(y+3) = \rho(r)$ by agent a, with $r = y+3$.

Now we can conclude the proof of the theorem. Since agent a is activated before or at the same time as agent a', agent a' cannot execute more than the first $\rho(r)$ phases before rendezvous occurs. Hence, at most $\rho(r)$ phases are executed by each of the agents a and a'. Since, in view of Claim 2, each agent makes at most $dP(n)$ moves during each phase, the total number of moves of the agents is in $O((\log(P(n)) + \log(c) + \log(L))P(n))$. Since P is a polynomial, the theorem follows. □

Notice that in the bounded fault scenario (as opposed to the unbounded fault scenario) it makes sense to speak about the time of a rendezvous algorithm execution (i.e., the number of rounds from the start of the earlier agent until rendezvous), apart from its cost. Indeed, now the time can be controlled by the algorithm. Our last result gives an estimate on the execution time of Algorithm `Graph-RV-BF`.

Theorem 5. *Algorithm* `Graph-RV-BF` *works in time polynomial in the size n of the graph, in the fault bound c and in the larger label L .*

Proof. From the proof of Theorem 4 we know that rendezvous occurs by the time when the agent activated earlier executes its ρ-th phase with $\rho = \log(P(n)) + \log(c) + \log(L) + O(1)$. Thus the number of rounds until rendezvous (since the activation of the first agent) is $\sum_{i=0}^{\rho} p_i$. We have:

$$\begin{aligned}\sum_{i=0}^{\rho} p_i &\leq 2p_\rho, \quad \text{since } \forall i \in \mathbb{N}, p_{i+1} = 4p_i \\ &\leq 2^{2\rho+5}, \quad \text{since } \forall i \in \mathbb{N}, p_{i+1} = 2^{2i+4} \\ &= O((P(n) \cdot c \cdot L)^2).\end{aligned}$$

Since P is a polynomial, Algorithm `Graph-RV-BF` works in time polynomial in the size n of the graph, in the fault bound c and in the larger label L. □

Notice the difference between the estimates of cost and of time of Algorithm `Graph-RV-BF`: while we showed that cost is polylogarithmic in L and c, for time we were only able to show that it is polynomial in L and c. Indeed, Algorithm `Graph-RV-BF` relies on a technique similar to "coding by silence" in the time-slice algorithm for leader election [30]: "most of the time" both agents stay idle,

in order to guarantee that agents rarely move simultaneously. It remains open whether there exists a rendezvous algorithm with bounded adversarial faults, working for arbitrary graphs, whose both cost and time are polynomial in the size n of the graph, and polylogarithmic in the fault bound c and in the smaller label ℓ .

6 Conclusion

We presented algorithms for rendezvous with delay faults under various distributions of faults. Since we assumed no knowledge of any bound on the size of the graph, for unbounded adversarial faults rendezvous is impossible, even for the class of rings. Hence it is natural to ask how the situation changes if a polynomial upper bound m on the size of the graph is known to the agents. In this case, even under the harshest model of unbounded adversarial faults, a simple rendezvous algorithm can be given. In fact this algorithm mimics the asynchronous rendezvous algorithm (without faults) from [16]. An agent with label λ, starting at node v of a graph of size at most m, repeats $(P(m)+1)^\lambda$ times the trajectory $R(m, v)$, which starts and ends at node v, and stops. Indeed, in this case, the number of integral trajectories $R(m, v)$ performed by the agent with larger label is larger than the number of edge traversals by the other agent, and consequently, if they have not met before, the larger agent must meet the smaller one after the smaller agent stops, because the larger agent will still perform at least one entire trajectory afterwards. The drawback of this algorithm is that, while its cost is polynomial in m, it is exponential in the smaller label ℓ. We know from Theorem 1 that the cost of any rendezvous algorithm must be at least linear in ℓ, even for the two-node tree. Hence an interesting open problem is:

> Does there exist a deterministic rendezvous algorithm, working in arbitrary graphs for unbounded adversarial faults, with cost polynomial in the size of the graph and in the smaller label, if a polynomial upper bound on the size of the graph is known to the agents?

References

1. S. Alpern, The rendezvous search problem, SIAM J. on Control and Optimization 33 (1995), 673-683.
2. S. Alpern, Rendezvous search on labelled networks, Naval Reaserch Logistics 49 (2002), 256-274.
3. S. Alpern and S. Gal, The theory of search games and rendezvous. Int. Series in Operations research and Management Science, Kluwer Academic Publisher, 2002.
4. E. Anderson and R. Weber, The rendezvous problem on discrete locations, Journal of Applied Probability 28 (1990), 839-851.
5. E. Anderson and S. Fekete, Asymmetric rendezvous on the plane, Proc. 14th Annual ACM Symp. on Computational Geometry (1998), 365-373.
6. E. Anderson and S. Fekete, Two-dimensional rendezvous search, Operations Research 49 (2001), 107-118.

7. E. Bampas, J. Czyzowicz, L. Gasieniec, D. Ilcinkas, A. Labourel, Almost optimal asynchronous rendezvous in infinite multidimensional grids, Proc. 24th International Symposium on Distributed Computing (DISC 2010), 297-311.
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. J. Chalopin, S. Das, P. Widmayer, Deterministic symmetric rendezvous in arbitrary graphs: Overcoming anonymity, failures and uncertainty, In "Search Theory: A Game Theoretic Perspective", S. Alpern et al. (eds.), Springer, 175-195, 2013.
11. M. Cieliebak, P. Flocchini, G. Prencipe, N. Santoro, Distributed computing by mobile robots: Gathering, SIAM J. Comput. 41 (2012), 829-879.
12. J. Czyzowicz, A. Kosowski, A. Pelc, How to meet when you forget: Log-space rendezvous in arbitrary graphs, Distributed Computing 25 (2012), 165-178.
13. J. Czyzowicz, A. Labourel, A. Pelc, How to meet asynchronously (almost) everywhere, ACM Transactions on Algorithms 8 (2012), article 37.
14. S. Das, Mobile agent rendezvous in a ring using faulty tokens, Proc. 9th International Conference on Distributed Computing and Networking (ICDCN 2008), 292-297.
15. S. Das, M. Mihalak, R. Sramek, E. Vicari, P. Widmayer, Rendezvous of mobile agents when tokens fail anytime, Proc. 12th International Conference on Principles of Distributed Systems (OPODIS 2008), 463-480.
16. 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.
17. A. Dessmark, P. Fraigniaud, D. Kowalski, A. Pelc. Deterministic rendezvous in graphs. Algorithmica 46 (2006), 69-96.
18. Y. Dieudonné, A. Pelc, Deterministic network exploration by a single agent with Byzantine tokens, Information Processing Letters 112 (2012), 467-470.
19. Y. Dieudonné, A. Pelc, D. Peleg, Gathering despite mischief, Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), 527-540.
20. Y. Dieudonné, A. Pelc, V. Villain, How to meet asynchronously at polynomial cost, Proc. 32nd Annual ACM Symposium on Principles of Distributed Computing (PODC 2013), 92-99.
21. P. Flocchini, E. Kranakis, D. Krizanc, F. L. Luccio, N. Santoro, C. Sawchuk, Mobile agents rendezvous when tokens fail, Proc. 11th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO 2004), 161-172.
22. P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of asynchronous robots with limited visibility, Theoretical Computer Science 337 (2005), 147-168.
23. P. Fraigniaud, A. Pelc, Delays induce an exponential memory gap for rendezvous in trees, ACM Transactions on Algorithms 9 (2013), article 17.
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. M. Koucký, Universal traversal sequences with backtracking, Journal of Computer and System Sciences 65 (2002), 717-726.
26. D. Kowalski, A. Malinowski, How to meet in anonymous network, Proc. 13th Int. Colloquium on Structural Information and Communication Complexity, (SIROCCO 2006), 44-58.

27. E. Kranakis, D. Krizanc, and P. Morin, Randomized Rendez-Vous with Limited Memory, Proc. 8th Latin American Theoretical Informatics (LATIN 2008), 605-616.
28. 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), 592-599.
29. W. Lim and S. Alpern, Minimax rendezvous on the line, SIAM J. on Control and Optimization 34 (1996), 1650-1665.
30. N.L. Lynch, Distributed algorithms, Morgan Kaufmann Publ. Inc., San Francisco, USA, 1996.
31. A. Pelc, Deterministic rendezvous in networks: A comprehensive survey, Networks 59 (2012), 331-347.
32. O. Reingold, Undirected connectivity in log-space, Journal of the ACM 55 (2008).
33. 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.
34. L. Thomas, Finding your kids when they are lost, Journal on Operational Res. Soc. 43 (1992), 637-639.