Rendezvous of Mobile Agents When Tokens Fail Anytime

Shantanu Das, Matúš Mihalák, Rastislav Šrámek,
Elias Vicari*, and Peter Widmayer*

Department of Computer Science, ETH Zurich, Zurich, Switzerland
{dass,mmihalak,rsramek,vicariel,widmayer}@inf.ethz.ch

Abstract. We consider the problem of Rendezvous or gathering of multiple autonomous entities (called mobile agents) moving in an unlabelled environment (modelled as a graph). The problem is usually solved using randomization or assuming distinct identities for the agents, such that they can execute different protocols. When the agents are all identical and deterministic, and the environment itself is symmetrical (e.g. a ring) it is difficult to break the symmetry between them unless, for example, the agents are provided with a token to mark the nodes. We consider fault-tolerant protocols for the problem where the tokens used by the agents may disappear unexpectedly. If all tokens fail, then it becomes impossible, in general, to solve the problem. However, we show that with any number of failures (less than a total collapse), we can always solve the problem if the original instance of the problem was solvable. Unlike previous solutions, we can tolerate failures occurring at arbitrary times during the execution of the algorithm. Our solution can be applied to any arbitrary network even when the topology is unknown.

Keywords: Rendezvous, Mobile Agents, Asynchronous, Anonymous Networks, Symmetry-breaking, Fault Tolerance, Faulty Token.

1 Introduction

We consider one of the fundamental problems in computing with autonomous mobile agents—the problem of gathering the agents, called *Rendezvous* [1]. The typical setting is when there is a communication network represented by a graph of n nodes and there are k mobile entities called agents, that are dispersed among the nodes of the network. The objective of the Rendezvous problem is to make all the agents gather together in a single node of the network. Solving Rendezvous is a basic step when performing some collaborative task with multiple distributed autonomous entities and it is known that many other problems such as network decontamination, intruder detection and distributed search, are easier to solve if the agents are able to rendezvous. When the nodes of the network are labelled

* The authors are partially supported by the National Competence Center in Research on Mobile Information and Communication Systems NCCR-MICS, a center supported by the Swiss NSF under grant number 5005 – 67322.

T.P. Baker, A. Bui, and S. Tixeuil (Eds.): OPODIS 2008, LNCS 5401, pp. 463–480, 2008.

with distinct labels from a totally ordered set, it is possible to gather at a predetermined location, for instance, the node having the smallest label. The problem is however more interesting when such unique labels are not present, i.e. when the network is anonymous. In this case, the agents have to reach an agreement among themselves about the node where to meet.

The problem of Rendezvous in an anonymous network has been studied for both synchronous [10,17,16] and asynchronous systems [9,14,15]; in this paper we focus on the more difficult case of asynchronous systems. In such systems, there is no common notion of time for the agents, and the speed of the agents traversing the network can be arbitrary. In fact, Rendezvous is not always deterministically solvable in asynchronous systems. However, Rendezvous can be achieved under certain conditions, by using a simple marking device called a *token* (also called a *pebble* or *marker*) [4,7,11,12,19]. Under this model, each agent has a token that it can put on a node to mark it. However, the tokens of all agents are identical (i.e., an agent cannot distinguish its token from a token of another agent). In [12], it was shown how to achieve Rendezvous in an anonymous ring network, when every agent puts its token on its starting location. Often it is assumed that the environment in which the agents are operating is hostile towards them i.e. it does not help in the task of achieving Rendezvous. For example, when an agent leaves a token, it is not guaranteed that the token would remain there until the agent returns (we consider asynchronous systems where there are no time bounds on the steps taken by an agent). Flocchini *et al.* [11] consider the model where some of the tokens are faulty, i.e. a token may suddenly disappear after an agent leaves it on a node. The solution given in the above paper depends on the very strong condition that the values of n and k are such $\gcd(n, k') = 1$, $\forall$ $k' \leq k$. This condition was later removed in [7]. However both these solutions work only in the special case when the tokens fail either immediately on being placed or they never fail at all. In other words, they do not solve the real problem of coordination between the agents using tokens that are unreliable.

In this paper, we first show how Rendezvous can be achieved in a ring network assuming that the tokens may fail at arbitrary times during the execution of the algorithm. Our protocol solves the problem under the exact conditions that are necessary for solving Rendezvous in the fault-free scenario. Further if Rendezvous is impossible in the given instance, the agents are able to detect it and terminate within a finite time. Thus, we solve the problem of *Rendezvous with Detect*. We only require that at least one token always survives (which is a necessary condition for feasibility of Rendezvous). Our algorithm has the same asymptotic cost as the previous solutions, in terms of the number of agent moves.

For arbitrary (and unknown) networks, there were no previous solutions for Rendezvous tolerating token failures. We show that Rendezvous can be solved in general graphs tolerating up to $k - 1$ failures, for any instance where the problem is solvable in absence of failures. Thus, the conditions for solvability of Rendezvous do not depend on the occurrences of token failures (excluding the case when all tokens fail). Even for this case, our algorithm can tolerate failures occurring at arbitrary times during the execution of the algorithm.

1.1 Our Model

We consider a network consisting of n nodes which is represented by an undirected connected graph $G = (V, E)$ (we shall use the words network and graph interchangeably). There are k *mobile agents* initially located in distinct nodes of the network. The node in which an agent initially resides is called its *homebase*. The agents are allowed to move along the edges of the graph. Each such move may take a finite but arbitrarily long time (i.e. the system is strongly asynchronous). The nodes of the network are anonymous (i.e. without any identifiers). At each node v of the graph, the incident edges are locally labelled, so that an agent arriving at the node v can distinguish among them. Additionally, the agent can identify the edge through which it arrived at the node[1]. The edge labelling of the graph G is given by $\lambda = \{\lambda_v : v \in V\}$, where for each vertex v of degree d, $\lambda_v : \{e(v, u) : u \in V \text{ and } e \in E\} \rightarrow \{1, 2, ..., d\}$ is a bijection specifying the local labelling at v.

The agents are exactly identical to each other and they follow the same (deterministic) algorithm. Each agent has a token which is a simple marking device that can be released at a node in order to mark it. Any agent visiting a node would be able to "see" the token left at that node, but would not be able to identify who left that token (i.e. the tokens are also anonymous like the agents). Each agent has a finite amount of memory that can be used to carry information. Two agents can communicate (exchange information) with each other, only when they are at the same node. An agent can also detect how many agents are present at a node. However, if two agents are traversing the same edge (from the same or opposite direction), they may not communicate or even see each other. It is also possible that an agent passes another agent on the same edge (without being aware of it). Each agent knows n, the size of the network, and k, the number of present agents. The objective is to reach an agreement about a unique node in the graph where all the agents should meet. The cost of a solution in this model is measured as the total number of moves (edge traversals) performed by all the agents combined.

In our model some of the tokens may be *faulty*. A faulty token is one which disappears at some instant during the execution of the algorithm and never appears again. We make the important assumption that at most $k - 1$ tokens may fail. In other words, at least a token is always present. Otherwise we would be in the situation as when no tokens were available and Rendezvous is not deterministically possible.

1.2 Our Results

We show that it is possible to achieve Rendezvous in asynchronous anonymous environments, even if the tokens that are used to break symmetry may fail. Unlike previous attempts which solved only very restricted special cases of the

[1] This ensures that the agent do not keep moving back and forth on the same edge forever.

problem, we present solutions which work in the most general setting, i.e. in an arbitrary unknown network where tokens may fail independently and at any time. Our algorithm works for any number of failures $0 \leq f \leq k-1$. In particular, it works also for the case when $f = 0$, i.e. when no tokens fail. We note that it is easier to solve Rendezvous for either only the fault-free scenario (i.e. $f = 0$) or only the faulty cases (i.e. $f \geq 1$) [2]. However, it is difficult to construct an algorithm which works for both scenario (or, when the switch from the fault-free to the faulty case can occur anytime during the execution of the algorithm). Also note that at best such an algorithm can achieve Rendezvous for only those instances where Rendezvous is possible in the absence of failures. Not only do our algorithms achieve this, but the agents can also detect the instances where Rendezvous is impossible and thus, they can terminate explicitly under all scenarios. Such an algorithm is said to solve *Rendezvous with Detect*.

In Section 2.2 we present solutions for the ring network which is the only case that has been studied before. Our algorithm solves *Rendezvous with Detect* when n and k are known, using at most $O(k\ n)$ moves. Based on the ideas of the solution for ring networks, we also present a (more complicated) solution for the general case, when the network topology is arbitrary and unknown. For an unknown anonymous network, there are no known sub-exponential cost algorithms for even traversing the network (i.e. visiting every node). Our algorithm for solving rendezvous in this case requires $O(k\ \Delta^{2n})$ moves for graphs of maximum degree Δ.

2 Rendezvous in a Ring Network

For problems involving symmetry-breaking, the ring networks represent a topologically simple but highly symmetrical (and hence difficult to solve) instance of the problem. As it is easier to visualize the problem in a ring network, we will first consider the case of ring networks, before proposing a solution for graphs of general topology. It was shown in [7] that it is not possible to solve *Rendezvous with Detect* in an asynchronous anonymous ring in the presence of token failures, if the value of k is unknown to the agents. In this paper, we assume that the agents have prior knowledge of the values of both n and k.

2.1 Properties and Conditions

If there are no failures, then the solvability of the Rendezvous problem depends on the initial locations of the agents. We can represent the initial location of the agents in a ring of size n by the sequence S_A of inter-agent distances, starting from an arbitrary agent A, in any direction (clockwise or counterclockwise), i.e., by the sequence $S_A := (d_0, d_1, \cdots, d_{k-1})$, where d_i denotes the distance between the i-th and $(i+1)$-th agent as they appear, in the order starting from agent A, on the ring (in the clockwise or counterclockwise direction). Observe

[2] If at least one failure is guaranteed then the failures can be used to break symmetry between the agents.

that for every other agent A' and every direction (clockwise or counterclockwise), the sequence $S_{A'}$ is just a "rotation" of S_A or a "rotation" of the *reversed sequence* of S_A (where the *reversed sequence* of S_A is $(d_{k-1}, d_{k-2}, \cdots, d_1, d_0)$). For any arbitrary sequence $S = (d_0, d_1, \ldots, d_{r-1})$ of r integers, we define the following. For any i, $0 \leq i \leq r$, $Sum_i(S)$ is defined as $\sum_{j=0}^{i-1} d_j$ (i.e. the sum of the first i elements of S). The *reversal* of S is defined as the sequence $Rev(S) = (d_{r-1}, d_{r-2}, \ldots, d_0)$. For any i, $0 \leq i < r$, the *i-th rotation* of S is defined as the sequence $Rot_i(S) = (d_i, d_{i+1}, \ldots, d_{r-1}, d_0, \ldots, d_{i-1})$. A sequence S is periodic if $\exists i$, $0 < i \leq (r/2)$, such that $Rot_i(S) = S$. A sequence S is called *rotation-reversal free*, if for every i, $0 < i < r$, $Rot_i(S) \neq S$ and $Rev(Rot_i(S)) \neq S$. Observe that if S is rotation-reversal free, then also $Rot_i(S)$ and $Rev(Rot_i(S))$ are rotation-reversal free, for any i, $0 < i < r$.

Lemma 1 ([7]). *Rendezvous of k agents can be solved in a ring, in absence of failures, if and only if the sequence $S = (d_0, d_1, \ldots, d_{k-1})$ of initial inter-agent distances (starting from any agent) satisfies the following conditions:*

(i) S is rotation-reversal free, or
(ii) S is not periodic, and there exists i, $0 < i \leq k-1$, such that $S = Rev(Rot_i(S))$ and at least one of $Sum_i(S)$ and $n - Sum_i(S)$ is even.

If $S = (d_0, d_1, d_2, \ldots, d_{k-1})$ is the sequence of inter-agent distances and it satisfies the conditions of Lemma 1, then we can define a unique node as the Rendezvous location, denoted by RV-point(S), in the following way[3]. If condition (i) of the above lemma holds, then there is a unique i, $0 \leq i \leq k-1$, such that either $Rot_i(S)$ or $Rev(Rot_i(S))$ is the lexicographically smallest sequence obtained by applying any number of reversions and rotations on S. In this case, we define RV-point(S) as the location of the i-th agent. Otherwise, if condition (ii) holds, then there exist unique i and j, $0 \leq i, j \leq k-1$, $i \neq j$, such that $Rot_i(S) = Rev(Rot_j(S))$ is the lexicographically smallest sequence (obtained by rotations and reversals on S). In this case, at least one of the segments (of the ring) between the i-th and the j-th agent is of even size (the *even* segment). Let u and v respectively be the homebase of the i-th and the j-th agent, and x be the node exactly in the middle of the even segment[4]. If the labelled path (using the labels of the edges) from x to u is lexicographically smaller than the path from x to v, then u is defined as RV-point(S), otherwise RV-point(S) is v. Since there is a local ordering on the edges incident at each node (in particular node x), the labels along the two paths would not be identical. We point out some important properties of the RV-point(S):

- RV-point is defined if and only if the instance of the Rendezvous problem is solvable (i.e. if and only if the conditions of Lemma 1 are met).
- For the sequence S of inter-agent distances of a solvable instance, RV-point(S) is a unique location in the ring and it is the homebase of some agent.

[3] To be precise, RV-point(S) depends on the edge-labelling of the ring as well.
[4] If both segments from u to v are even-sized, we pick the lexicographically smaller one.

A simple strategy for solving Rendezvous with Detect in the absence of failures is the following [12]. Each agent puts its token at the starting location and goes around the ring once (i.e. moves n steps in the same direction), to compute the sequences S. The agent now moves to the location defined by RV-point(S), or detects that the instance is not solvable. Notice that the agents may move in different directions and the sequences obtained by them would be distinct (but where one can be transformed into another by using rotation and/or a reversal operation), they would still gather in the same location. However, this strategy would fail in the presence of token failures (unless some strong conditions are imposed on the values of n and k). We note that the (exact) conditions for solving Rendezvous (in ring networks) in presence of arbitrary token failures were, until this paper, not known. We show that the conditions of Lemma 1 are sufficient to solve Rendezvous with Detect in a ring even in the faulty scenario when tokens may fail at arbitrary times, provided that the agents know the values of n and k (and at least one token does not fail).

2.2 Solution Protocol

We know that Rendezvous in a ring (with no faults on tokens) is possible only if the initial sequence of inter-agent distances satisfies the conditions of Lemma 1. We show that even if some of the tokens disappear, the agents are still able to reconstruct the initial sequence of inter-agent distances and thus to decide whether the instance is solvable, and to eventually meet at the Rendezvous location.

The main idea behind our solution of reconstructing the sequence of the inter-agent distances is the following. Agents whose tokens have failed (called RUNNERS) "run" around and inform other agents about their location, while those agents whose token did not fail (called OWNERS), wait at the homebase to receive the "running" agent. The difficulty of this approach is when no token disappears (as then no agent is "running", and thus every agent waits in its homebase – a deadlock). On the other hand, if we know that no token disappeared, and the instance is solvable, we can elect a leader agent – the agent with its homebase as the Rendezvous location RV-point(S), who starts "running" and informs others about the Rendezvous location. Combining both cases in one algorithm is, in our opinion, the most interesting part of the solution.

More precisely, the agents do the following. In the beginning, each agent puts its token on its homebase, and walks once around the ring for n steps (thus, it comes back to its homebase), making notice of inter-agent distances as induced by the observed tokens during the walk[5]. We call this walk the *initial walk*. Comparing k with the number of found tokens during the walk (the agent considers its own token at the end of its walk), the agent knows whether some of the tokens disappeared (during the agent's walk). If this is the case, then the agent (1) either waits at its homebase, if its token is present (and stays there even if the token disappears later on – in this case the agent acts as a token and informs other

[5] The walk of each agent is determined by the first edge the agent uses when starting from its homebase, which can be e.g. the edge with the smallest label. This also determines whether the agent runs clockwise or counterclockwise.

passing-by agents about this), or (2) walks around the ring to the next token (or to an OWNER who acts as a token) – the agent associates itself with the owner of this token (the agent may need to wait for the OWNER which might still be on its initial walk around the ring), and starts "running" around the ring to inform every OWNER about the location of the failed token. When a RUNNER gets back to the OWNER it is associated to, it attempts to reconstruct the original sequence of inter-agent distances. If this information is complete, it goes around and informs all OWNERS about the meeting point (and the total number of RUNNERS). The informed owners then subsequently inform their associated RUNNERS about this, and walk "together" to the Rendezvous location.

In the other case, when the initial walk of an agent results in encountering all k tokens, and thus computing the correct sequence S of inter-agent distances of k agents, we have to modify the above algorithm. To ensure that even in the case when no token fails (after the initial walk of every agent), there is a special agent – a "leader" – which walks once around the ring and informs all other OWNERS about the situation. The special agent is the agent whose homebase is the Rendezvous location RV-point(S). If a token should fail after the election of a "leader" but before that all agents know about the Rendezvous location, an agent becomes RUNNER and the algorithm reverts to the aforementioned case.

Thus, the agents may have one of the following roles in our protocol. In case, the agent's own token fails during the initial walk, the agent becomes a RUNNER. In case the agent finds all tokens during the initial walk, and its homebase is the Rendezvous location, the agent becomes a special agent called the LEADER. RUNNERS and the LEADER move around the ring collecting information and exchanging this information with OWNERS, leading into informing all OWNERS (and thus subsequently all RUNNERS) about the (computed) Rendezvous location. An OWNER is an agent that does not identify itself as a RUNNER or the LEADER. The role of OWNERS is to help RUNNERS in coordinating with each other.

We now present the complete algorithm for solving the Rendezvous problem in a ring when tokens may fail anytime (and at least one token does not fail).

Algorithm *RVring* :

1. Put token at starting location;
   Move n steps, and compute inter-token sequence S (we call it the *initial walk*);
2. If own token disappeared then become RUNNER;
   Else become OWNER;
   If k tokens were found
   If S satisfies solvability condition,
   If own location is RV-point(S) then become LEADER;
   Else (S does not satisfy solvability conditions)
   set Status to FAILURE;
3. A LEADER agent executes the following:

   (The LEADER*'s checking walk*)
   Go around the ring, waiting at each token for the OWNER to return;

If all $k-1$ OWNERS were found,
(The LEADER*'s gathering walk*)
Collect all OWNERS to RV-point(S) and set Status to SUCCESS;
Else (some token disappeared)
If own token disappeared then become RUNNER;
Else become OWNER;

4. An OWNER agent simply waits at its homebase, communicates with passing agents and follows their instructions. Each OWNER stores the information about its associated RUNNERS. The OWNER moves only when the LEADER tells to move, or if the Rendezvous location is known, it has met every RUNNER in their teaching walk (the OWNER counts the number of RUNNERS that it met in the teaching walk) and all its associated RUNNERS have been informed.
5. A RUNNER agent executes the RUNNER algorithm (explained below).

Remarks:

- After step 2 of the algorithm *RVRing*, only LEADER can change its role. Especially, an OWNER agent never becomes RUNNER even if its token disappears later on.
- If a node v contains an OWNER agent then it is assumed that there is also a token at node v. This means that if a token disappears after its OWNER has returned, this has no effect on the execution of the algorithm. The OWNER simulates the presence of the token communicating it to any other agent that visits v.

Runner's Algorithm:

(1) *A* RUNNER *agent associates with exactly one* OWNER *agent.*
An agent becomes a RUNNER only if it finds that the token at its homebase has disappeared. Such an agent moves to the next node that contains a token (or contains an OWNER) and waits for the owner of that token. In case the token disappears before the owner of the token comes to the node, then the RUNNER moves to the next token (or OWNER) and repeats the same. Once the RUNNER meets the awaited OWNER of the token, it *associates* with this OWNER. The RUNNER agent remembers the distance and direction from its homebase to the homebase of the associated OWNER. This information is communicated to the associated OWNER. OWNER keeps track of the number of its associated RUNNERS and of their homebase locations.
(2) *A* RUNNER *tells the other* OWNERS *about its homebase.*
After associating with an OWNER, the RUNNER agent goes once around the ring (we call it the RUNNER*'s teaching walk*), stopping at each token, waiting for the OWNER of that token and then communicating the information about the position of its homebase to that OWNER.
(3) When the RUNNER reaches its associated OWNER again, it learns the sequence S' (possibly incomplete) that the OWNER has gathered so far. If S' is the correct sequence and Rendezvous is solvable for this instance, then the

RUNNER agent computes the RV-point and communicates it to all OWNERS. The RUNNER agent can check if the sequence is complete, i.e., if $|S'| = k$. If the sequence is complete, the agent walks around again (we call it a RUNNER*'s informing walk*), stops at every OWNER and informs the OWNER about the complete sequence S' and about the number of RUNNERS (determined by k minus the number of OWNERS). In this walk the RUNNER does not wait at unoccupied nodes with a token on it. If S' satisfies the conditions of Lemma 1 (i.e., the instance of the Rendezvous problem is solvable), the status of the OWNER is set to SUCCESS, otherwise the status is set to FAILURE.

(4) A RUNNER agent waits at the homebase of its associated OWNER until the status of this OWNER is changed to SUCCESS or FAILURE, upon which the RUNNER learns the original sequence of inter-agent distances S. If the status is SUCCESS, it moves to the Rendezvous location RV-point(S).

2.3 Proof of Correctness

We need to show that every agent learns the correct sequence of the original inter-agent distances S, and that (in case the instance is solvable) every agent moves to the Rendezvous location RV-point(S).
We proceed with small (reassuring) observations.

1. *Every agent finishes its initial walk.* This is a rather trivial observation.
2. *If the instance is not solvable and there is no* RUNNER*, then every agent detects this after the initial walk.* This is again obvious, as in case there is no RUNNER, every agent computes the correct sequence S and thus sets its status to FAILURE.
3. *After the initial walk, every* RUNNER *(if there are some) associates with an* OWNER*.* This follows from the assumption that there is at least one token which does not disappear. Thus, the RUNNER either finds an OWNER immediately, or it finds a token. The token belongs to an agent in its initial walk, or to a LEADER in its checking walk. In both cases the agent comes back.
4. *Every* RUNNER *finishes its teaching walk.* The RUNNER may wait for an agent to come to its token from its initial walk, but we know that every agent finishes this phase. If the token disappears before the OWNER of the token returns, the RUNNER continues. Further, if the RUNNER waits at a token of the LEADER, the LEADER finds out that there is a RUNNER and thus the LEADER goes back home and becomes OWNER.
5. *No* OWNER *leaves its homebase before meeting every* RUNNER*.* If the OWNER moved because the LEADER said so, then there is no RUNNER; Otherwise (see the algorithm) the OWNER waits until it sees all RUNNERS in their teaching walk. The OWNER checks this by keeping track of how many RUNNERS it met in the teaching walk, and by comparing this count with the total number of RUNNERS (which it learns from a RUNNER in an informing walk). The RUNNERS inform an OWNER in which walk they are.

6. *Every* Owner *meets every* Runner *in the* Runner*'s teaching walk.* Indeed, consider any Owner and any Runner. We know that the Owner does not move if it did not meet every Runner (see previous observation). As every Runner goes for its teaching walk, it eventually meets the considered Owner.

We now prove the first lemma which helps us to reach our goal.

Lemma 2. *If there is no* Leader*, or the* Leader *did not meet* $k-1$ Owners *in its checking walk, then there is a* Runner *which is informed of the initial sequence* S *of inter-agent distances after its teaching walk.*

Proof. It should be obvious that if there is no Leader, then there exist Runners. We now show that at least one Runner obtains the original sequence S of inter-agent distances. Consider a Runner agent which was the last among all Runners to finish its teaching walk (let us call it the *last* Runner[6]). Thus, at this time, every other Runner has already communicated its homebase information. When the *last* Runner reaches the Owner which it is associated to, this Owner possesses all information about the homebases of all Runners (and knows the position of all Owners from the initial walk). Hence, the original sequence S can be reconstructed by this Runner.

Lemma 3. *Every agent learns the original sequence* S *of inter-agent distances.*

Proof. Let us first consider the situation when all agents became either Owner or Leader. In this case they all learn the correct S, as they all met all tokens during the initial walk.

Otherwise, there exists at least one Runner. Thus, by the previous lemma, there is a Runner R which learns in its teaching walk the original sequence S. Thus, this Runner R begins its informing walk, in which it informs all Owners it meets about S. We claim that after the Runner R finishes its informing walk, every Owner knows S. Indeed, when R arrives during its informing walk at a homebase of an Owner, the Owner is either present there, and thus learns S from R, or the Owner is not present there, which means that the Owner left, which is only possible if the Owner knows S. (We note that this may happen e.g. if another Runner with complete information about S already finished its informing walk, and thus left together with its associated Owner to the Rendezvous-location.)

Lemma 4. *If the instance of the Rendezvous problem is not solvable, then all agents will learn this information, and will stop moving at some time.*

Proof. By the previous lemma, every agent will learn the initial sequence S of inter-agent distances, from which it finds out that the instance of the problem is not solvable. Notice that every agent stops after walking at most three times around the ring in case that the instance is not solvable.

[6] There could possibly be multiple such agents. For the sake of arguments, we arbitrarily choose one of them.

Lemma 5. *If the instance of the Rendezvous problem is solvable, then the agents meet at the Rendezvous location RV-point(S), where S is the initial sequence of inter-agent distances.*

Proof. Due to Lemma 3 we know that every agent learns S and thus knows where is the Rendezvous location RV-point(S). We now show that every agent eventually walks to the Rendezvous location. We consider first the case when an agent becomes LEADER and in its checking walk it meets $k-1$ OWNERS. Clearly, in this case, after the LEADER finishes its gathering walk, all OWNERS walk to the Rendezvous location.

Let us now consider the case when there is no LEADER, or the LEADER does not meet $k-1$ OWNERS. In this case, there are only OWNERS and RUNNERS present (eventually, after the LEADER ends its checking walk and becomes RUNNER or OWNER). Let us consider a RUNNER agent. By Lemma 3 we know that the RUNNER learns the initial sequence S and whenever this happens the RUNNER simply walks to the Rendezvous location RV-point(S). Let us now consider an OWNER. Again by Lemma 3 we know that it learns the initial sequence S. The OWNER walks to the Rendezvous location RV-point(S) only when it knows that all RUNNERS met the OWNER in their teaching walks. As every RUNNER will eventually finish its teaching walk (see the observations above), the OWNER will have met all RUNNERS and thus walks to the Rendezvous location RV-point(S).

Theorem 1. *The algorithm* RVring *solves the Rendezvous problem with detect whenever there are at most $k-1$ token failures, where k is the number of agents. The total number of moves made by the agents is $O(k\ n)$ and the memory requirement for each agent is $O(k \log n)$.*

Proof. Due to Lemmas 4 and 5, we know that the algorithm always terminates explicitly. Further, whenever Rendezvous is solvable, the algorithm achieves Rendezvous of k agents and otherwise detects unsolvability. This proves correctness of the algorithm. To bound the number of total movements of the agent, observe that in the algorithm *RVring* each agent (RUNNER, LEADER or OWNER) performs a constant number of traversals of the ring (at most five in case of a RUNNER – the initial walk, the association of RUNNERS to OWNERS, the checking walk, the informing walk, and the walk to the Rendezvous location) where each traversal takes exactly n moves. The memory required to store the sequence S' of inter-token distances is $O(g \log n)$ where $g \leq k$ is the number of OWNER agents.

Our algorithm compares favorably with previous results which solve the Rendezvous problem in fault-free scenario (or the restricted faults scenario) with the same cost and the same memory requirements. In fact, it is easy to show that any algorithm for solving *Rendezvous with Detect* requires $\Omega(k\ n)$ moves in the worst case.

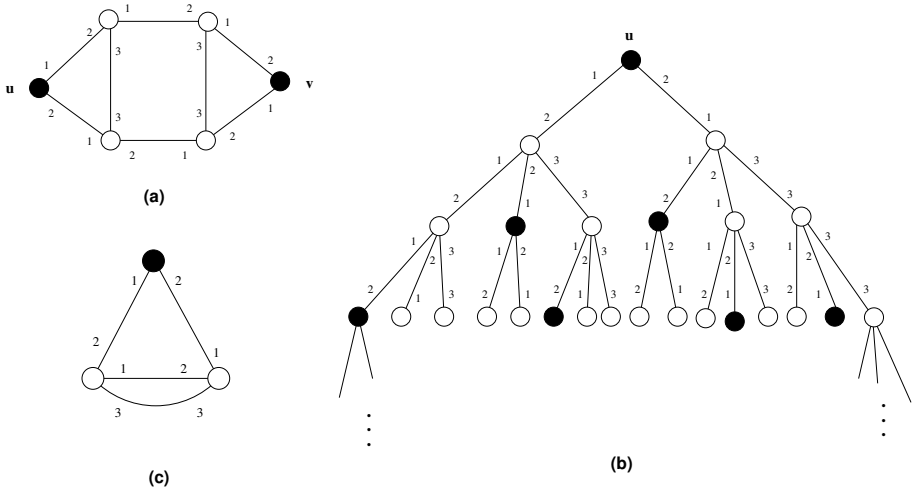

Fig. 1. (a) A network with $n = 6$ and two agents at nodes u and v. (b) The bi-colored view of node u. (c) The quotient graph of the network.

3 Rendezvous in Arbitrary Networks

3.1 Conditions for Solving Rendezvous

In the general case of arbitrary graphs, the initial location of the agents in the graph $G(V, E)$ is specified using a placement function $p : V \rightarrow \{0, 1\}$, where $p(v) = 1$ if the node v is a homebase of an agent and $p(v) = 0$ otherwise[7]. The function p is a bi-coloring on the set of vertices of G (We say that a vertex v is black if $p(v) = 1$ and white if $p(v) = 0$). An instance of the Rendezvous problem is defined by the tuple (G, λ, p) where λ is the edge labelling on the graph G and p defines the initial location of agents in G. The conditions for solving Rendezvous in arbitrary graphs are related to the conditions for leader election in the anonymous message-passing network model, which has been studied extensively [2,5,20]. The concept of a *view* of a node in a network was introduced by Yamashita and Kameda [20] with respect to the symmetry-breaking problem in message passing systems. The *view* of a node v in the labelled graph (G, λ) is an infinite edge-labelled rooted tree that contains all (infinite) walks starting at v. For the Mobile Agent model, the concept of view can be extended to that of a bi-colored view, where the vertices in the view are colored black or white depending on whether or not they represent the homebase of some agent (recognizable by the presence or not of a token).

In the network represented by (G, λ, p), the *bi-colored view* of a node v is an infinite edge-labelled rooted tree T whose root represents the node v, and for each neighboring node u_i of v in G, there is a vertex x_i in T (colored as u_i) and

[7] Recall that the agents start from distinct nodes, so each node contains at most one agent initially.

an edge from the root to x_i with the same label as the edge from v to u_i in G. The subtree of T rooted at x_i is (recursively) the view of the node u_i.
The following result is known for Rendezvous of agents in arbitrary graphs:

Lemma 6 ([2,3,20]). *When the agents do not have a-priori knowledge of the network topology, the Rendezvous problem is solvable in an arbitrary network (G, λ, p) if and only if each node in G has a distinct bi-colored view.*

Notice that the view of a node $u \in G$ contains multiple occurrences of each vertex in G (see Figure 1). In fact the view of a node u truncated to depth $2n-2$ (denoted by T_u^{2n-2}) contains the view of any other node up to depth of at least $n-1$. From the results of Norris [18], it is known that the views of two nodes are identical if and only if their views up to depth $n-1$ are identical. Thus, from the view T_u^{2n-2} of any node u, it is possible to obtain a so-called *closed view* where vertices with identical views are merged into a single vertex. The (multi)-graph H thus obtained (which may contain self-loops and parallel edges), is called the quotient-graph [20] of the network. In the notion of *graph coverings*, one can say the graph G covers the graph H and this covering preserves the labelling of the edges and the bi-coloring on the vertices. If the view of each node is distinct, then the quotient graph H is isomorphic to G and we say that (G, λ, p) is covering minimal [5]. Hence, an alternate (but equivalent) condition for solving the Rendezvous problem is that the network is covering minimal with respect to coverings that preserve the edge-labelling and the vertex-coloring. In this case, we can define a unique rendezvous location in G by ordering the bi-colored views (according to some fixed total order) and choosing the node whose bi-colored view is the smallest.

3.2 Rendezvous Algorithm for Unknown Graphs

In the case of an unknown arbitrary graph, we have the following strategy for solving Rendezvous. Each agent can perform a (blind) depth-first traversal up to depth $2n-2$ to construct the bi-colored view. (This is the initial walk of the agent.) It is possible that tokens may disappear during the traversal. So, if a token at a node v disappears between two consecutive visits to v by the agent, then the agent would obtain an inconsistent coloring of the view (without being able to detect this inconsistency by just looking at the view). We say that a bi-coloring of a view is *inconsistent* if multiple occurrences of the same vertex in the view are colored differently. However, it is easy to check for consistency by simply repeating the view computation one more time and then comparing the results.[8] If the view is inconsistent, then we know there must be at least one token failure, and in that case the agent does not need to construct the correct bi-colored view in its initial walk (as explained below). Otherwise, given the view of the network and any consistent bi-coloring of the view, each agent can compute the closed view (H, λ_H, P_H), where H is a multi-graph, λ_H is an edge labelling

[8] If there is any inconsistency due to disappearance of tokens, then the bi-colored views obtained in two consecutive computations would be distinct.

and P_H is a vertex bi-coloring of H. We define $k' = |\{v \in H : P_H(v) = 1\}|$ as the agent-count of H. Observe the following:

- The labelled graph (G, λ, p) covers the labelled multigraph (H, λ_H, P_H). Thus $|H|$ divides $|G|$.
- The number of tokens detected is $k' \times (n/|H|)$. Thus, $k' \times (n/|H|) \leq k$. If P_H is the correct bi-coloring then $k/k' = n/|H|$. Thus, an agent can determine whether the bi-coloring that it computed is the correct bi-coloring or not.
- If P_H is the correct bi-coloring and Rendezvous is solvable in (G, λ, p), then H is isomorphic to G and $k' = k$. This implies that if an agent obtains the correct bi-coloring, then it can detect whether Rendezvous is solvable or not. Further, if Rendezvous is solvable, the agent can traverse G using the map H and it can also determine the unique Rendezvous location as defined earlier.

We now show how to extend the algorithm for rings to obtain a Rendezvous algorithm for arbitrary graphs, using the above ideas.

Each agent computes the bi-colored view in the first round of the algorithm. As mentioned before an agent needs to perform the view computation twice to check for consistency. In case the results of the two computations differ, then the agent knows that some token must have failed and thus it returns to its home-base and becomes either OWNER or RUNNER (depending on whether its own token is present). On the other hand, if no tokens fail, then the computed bi-coloring would be the correct one and we proceed as in the previous algorithm, by electing a LEADER agent. In case of token failures, there would exist some RUNNER agents and each RUNNER would try to compute the correct bi-colored view. This done in two rounds as follows. First each RUNNER traverses the view and communicates to each OWNER the (labelled) path it traversed to reach this OWNER. In the second round the RUNNER traverses the view again to obtain information from each OWNER (about the paths leading to other RUNNERS). This information can be used to obtain the bi-coloring of the view. As before, we can show that at least one RUNNER would compute the correct bi-coloring. In the case when Rendezvous is solvable, this agent can build a map of the graph and determine the unique Rendezvous location; Thus, we can proceed in a manner similar to the previous algorithm. In the other case, when Rendezvous is not solvable, this agent can detect this and inform each OWNER about the impossibility of Rendezvous. The new algorithm, called *RV-General* is described below.

Algorithm *RV-general* :

1. Put token at starting location ;
   Construct bi-colored view up to depth $2n - 2$;
   Check for consistency of the bi-coloring;

2. If own Token disappeared then become RUNNER;
   Else become OWNER;
   If the bi-colored view is consistent,
   Construct the quotient-graph H

(let n' be the size of H and k' be the number of black nodes);
If $n' = n$ and $k' = k$,
If own location is RV-location(H) then become LEADER;
Else if $n/n' = k/k'$ declare Failure; (set Status to FAILURE)

3. A LEADER agent executes the following:

Traverse H, waiting at each token (for the OWNER to return);
If all $k-1$ OWNERS were found,
Collect all OWNERS to RV-location(H) and set Status to SUCCESS;
Else (some token disappeared)
If own Token disappeared then become RUNNER;
Else become OWNER;

4. An OWNER agent simply waits at its homebase, communicates with passing agents and follows their instructions. Each OWNER stores (communicates) the information about paths to RUNNERS. The OWNER moves only when the LEADER tells to move, or if the Rendezvous location is known, it has met every RUNNER in their teaching walk (the OWNER counts the number of RUNNERS that it met in their teaching walk and compares this with the RUNNER-Number) and all its associated RUNNERS have been informed.
5. A RUNNER agent executes the RUNNER algorithm (explained below).

Runner's Algorithm:

(1) A RUNNER agent perform a depth-first traversal of the view, and it associates with the first OWNER agent that it meets (as before, the RUNNER waits at each token for the OWNER to return). During the depth-first traversal, the RUNNER agent remembers the labelled path it traversed from its homebase to the current node and if the current node has a token, the agent communicates the path information to the OWNER of this token. This is the RUNNER's *teaching walk*.
(2) A RUNNER agent, after completing its *teaching walk*, returns to its associated OWNER and collects information from the OWNER about paths to other RUNNERS. Based on this information, it computes a bi-coloring of the view and builds the quotient graph H. Note that this may not be the correct quotient-graph of G (due to missing information about some RUNNERS).
(3) Let n' be the size of H and k' be the number of black nodes in H. If $n' = n$ and $k' = k$, then H is identical to G and this represents a solvable instance of the Rendezvous problem. In this case, the RUNNER computes the RV-location and then traverses the graph G, stopping at each OWNER to communicate the RV-location and the RUNNER-Number (This is simply a count of the number of vertices in the bi-colored view of this OWNER, which corresponds to some RUNNER homebase). The RUNNER also changes the status of each OWNER to SUCCESS.
Otherwise, if $H < G$, but $n/n' = k/k'$, then this represents an unsolvable instance of the Rendezvous problem. In this case, the RUNNER traverses a

spanning tree of H and changes the status of each OWNER (that it meets) to FAILURE.

(4) A RUNNER agent waits at the homebase of its associated OWNER until the status of this OWNER is changed to SUCCESS or FAILURE. If the status is SUCCESS, it moves to the Rendezvous location RV-point(S).

The correctness of algorithm *RV-General* can be proved in a similar manner as for the previous algorithm. We first show that the agents are able to compute the original bi-coloring and thus construct the minimum-base of the network.

Lemma 7. *Every* OWNER *agent is able to obtain the minimum-base of the network. Further if there are some* RUNNERS *then at least one* RUNNER *also obtains this information.*

Proof. We first show that each OWNER obtains the correct bi-coloring of the view. If there are no failures, then the result holds trivially, so we consider only the case when some tokens failed. Consider an OWNER agent A sitting at its homebase node v. If there is a path p (of length less than $2n-2$) from some node x to v, and there was a token on node x that failed, then there must be a RUNNER agent corresponding to this failed token and this RUNNER would traverse all paths (of length at most $2n-2$). In particular, this RUNNER would traverse path p and reach node v. Thus, for each path starting at node v, agent A could determine if the end point of the path is colored black or white. Notice that an OWNER waits at its homebase until it is visited by every RUNNER during its teaching walk. Thus, once every RUNNER has completed its traversal of the view, every OWNER would obtain the correct bi-coloring of the view. Consider now the last RUNNER to complete its traversal of the view. When this RUNNER returns to its associated OWNER, the OWNER would at that time have obtained the complete bi-colored view. Thus, this RUNNER would be able to construct the minimum-base of the network.

From the previous discussion and due to Lemma 7, the following result holds.

Theorem 2. *Algorithm* RV-General *solves the Rendezvous in any arbitrary network (G, λ, p) whenever the conditions of Lemma 6 are satisfied. Otherwise, every agent terminates with FAILURE.*

Proof. If there is a LEADER and $k-1$ OWNERS, then the LEADER informs every OWNER about the Rendezvous-location and moves them to the Rendezvous-location. Thus, the theorem holds. Consider now the situation when there some RUNNERS (this implies that there is no LEADER). Due to the previous lemma, at least one RUNNER constructs the minimum-base H of the network. If the conditions of Lemma 6 are satisfied then H is a map of the network and there is a unique Rendezvous-location. The RUNNER traverses the network and changes the status of every OWNER to SUCCESS (and informs them about the Rendezvous-location). Thus, every RUNNER learns about the Rendezvous-location from its associated OWNER and moves to the Rendezvous-location. An OWNER moves

to the Rendezvous-location after all its associated RUNNERS have learned the Rendezvous-location. Thus, all agents meet at the Rendezvous-location. In the other case when the conditions of Lemma 6 are not satisfied, at least one RUNNER learns about this (after computing the minimum-base). This RUNNER changes the status of every OWNER to FAILURE. Every RUNNER eventually returns to its associated OWNER and waits until the status is changed. Thus, every agent terminates with status FAILURE.

Theorem 3. *During the algorithm* RV-General*, the agent performs* $O(k\Delta^{2n})$ *moves in total, where* Δ *is the maximum degree of any node in* G*.*

Proof. Each view computation requires $O(\Delta^{2n})$ agent moves. Each agent initially performs view-computation two times. Every RUNNER agent performs two more traversals of the view T. Other than that each agent performs a constant number of traversals of the quotient graph H which has size at most n.

4 Conclusions

We presented algorithms for solving Rendezvous with unreliable tokens in a strongly asynchronous environment, when the tokens may fail at any arbitrary time. As long as at least one token remains, we were able to solve the problem under the same conditions as required for Rendezvous without failures. Thus, our results show that the occurrence of $f < k$ faults has no effect on the solvability of the Rendezvous problem. Our algorithm for the case of ring networks requires $O(k\ n)$ agent moves and has therefore the same asymptotic cost as in the absence of failures. Our solution can also be applied to the more general case of an unknown graph, albeit with an exponential cost in terms of agent moves. This cost is same as that of the known solution for the fault-free case [8]. Improving this cost involves finding a more efficient way of traversing and exploring an unknown unlabelled graph.

References

1. Alpern, S., Gal, S.: The Theory of Search Games and Rendezvous. Kluwer, Dordrecht (2003)
2. Angluin, D.: Local and global properties in networks of processors. In: Proc. of 12th Symposium on Theory of Computing (STOC 1980), pp. 82–93 (1980)
3. Barrière, L., Flocchini, P., Fraigniaud, P., Santoro, N.: Can we elect if we cannot compare? In: Proc. Fifteenth Annual ACM Symp. on Parallel Algorithms and Architectures (SPAA 2003), pp. 324–332 (2003)
4. Baston, V., Gal, S.: Rendezvous search when marks are left at the starting points. Naval Research Logistics 38, 469–494 (1991)
5. Boldi, P., Shammah, S., Vigna, S., Codenotti, B., Gemmell, P., Simon, J.: Symmetry breaking in anonymous networks: Characterizations. In: Proc. 4th Israel Symp. on Th. of Comput. and Syst., pp. 16–26 (1996)

6. Chalopin, J., Das, S., Santoro, N.: Rendezvous of Mobile Agents in Unknown Graphs with Faulty Links. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 108–122. Springer, Heidelberg (2007)
7. Das, S.: Mobile Agent Rendezvous in a Ring using Faulty Tokens. In: Rao, S., Chatterjee, M., Jayanti, P., Murthy, C.S.R., Saha, S.K. (eds.) ICDCN 2008. LNCS, vol. 4904, pp. 292–297. Springer, Heidelberg (2008)
8. Das, S., Flocchini, P., Nayak, A., Santoro, N.: Effective elections for anonymous mobile agents. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 732–743. Springer, Heidelberg (2006)
9. De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous deterministic rendezvous in graphs. Theor. Comput. Sci. 355(3), 315–326 (2006)
10. Dessmark, A., Fraigniaud, P., Kowalski, D.R., Pelc, A.: Deterministic Rendezvous in Graphs. Algorithmica 46(1), 69–96 (2006)
11. Flocchini, P., Kranakis, E., Krizanc, D., Luccio, F.L., Santoro, N., Sawchuk, C.: Mobile Agents Rendezvous When Tokens Fail. In: Kralovic, R., Sýkora, O. (eds.) SIROCCO 2004. LNCS, vol. 3104, pp. 161–172. Springer, Heidelberg (2004)
12. Flocchini, P., Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Multiple mobile agent rendezvous in a ring. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 599–608. Springer, Heidelberg (2004)
13. Flocchini, P., Roncato, A., Santoro, N.: Computing on anonymous networks with sense of direction. Theor. Comput. Sci. 301, 1–3 (2003)
14. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theor. Comput. Sci. 390(1), 27–39 (2008)
15. Kosowski, A., Klasing, R., Navarra, A.: Taking Advantage of Symmetries: Gathering of Asynchronous Oblivious Robots on a Ring. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 446–462. Springer, Heidelberg (2008)
16. Kowalski, D.R., Pelc, A.: Polynomial Deterministic Rendezvous in Arbitrary Graphs. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 644–656. Springer, Heidelberg (2004)
17. Kranakis, E., Krizanc, D., Markou, E.: Mobile Agent Rendezvous in a Synchronous Torus. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 653–664. Springer, Heidelberg (2006)
18. Norris, N.: Universal covers of graphs: isomorphism to depth $n-1$ implies isomorphism to all depths. Discrete Applied Math 56, 61–74 (1995)
19. Sawchuk, C.: Mobile agent rendezvous in the ring. PhD Thesis, Carleton University (2004)
20. Yamashita, M., Kameda, T.: Computing on anonymous networks: Parts I and II. IEEE Trans. on Parallel and Distributed Systems 7(1), 69–96 (1996)