An Autonomous Traceability Mechanism for a Group of RFID Tags

Yann Glouche
INRIA, Unité de Recherche Rennes-Bretagne-Atlantique
Campus de Beaulieu,
35042 Rennes Cedex, France
email: yann.glouche@inria.fr

Paul Couderc
INRIA, Unité de Recherche Rennes-Bretagne-Atlantique
Campus de Beaulieu,
35042 Rennes Cedex, France
email: paul.couderc@inria.fr

***Abstract*—Coupled Objects are an innovative way to ensure integrity of group of objects, or complex objects made of parts. This principle can be used in various applications such as logistic or security. The main property of coupled objects is that integrity checking is autonomous and does not depend on external information systems: all the necessary data are self contained in radio-frequency identification tags associated to the objects. This avoids important issues such as scalability and privacy, but the self-contained approach makes error diagnostics difficult when an integrity check fails. In this paper, we propose a solution to this problem, with a resilient data structure supporting the identification of missing elements in a coupled object. When some elements among the coupled objects are missing, it is possible to detect if the group is corrupted. Moreover, our approach also allows to identify the missing elements.**

***Keywords*-RFID; checking; integrity; data structure; graph.**

I. Introduction

In recent years, we have seen increasing adoption of the radio-frequency identification (RFID) technology in many application domains, such as logistic, inventory, public transportation, and security. In this paper, we focus on an innovative integrity checking approach, which uses a collection of RFID tags distributed over a set of object, discussed in [1], [2]. In the coupled objects approach, a set of physical entities are associated in a logical group by writing integrity data in an RFID tag on each object. The integrity of the group can then be checked when needed by reading the tag data and verifying integrity properties.

For example, package containing a group of tagged items could be transported in a secured way by different parties to the final recepient. Liability transfer and trust between parties would be ensured by checking the package integrity at each transfer step.

Unlike typical RFID systems, the tags store data, which are self sufficient for the integrity checking application, without needing access to an external data base or information system. Autonomous operation is an important feature as it avoids many issues associated with typical RFID systems: network access and scalability, database scalability, privacy and security.

However, the coupled objects architecture also has some limitations: when an integrity error is detected in a group of objects, characterizing the nature of the error is challenging. Typically, when some parts are missing in the group we would like to identify them, or to recover the application data associated with them. Unfortunately, as RFID tags provide very limited memory capacity, it is difficult to implement redundant storage and robust data structures distributed over a set of RFID tags. It is this problem that we address in this paper. We propose a resilient data structure for coupled objects, which can be used in particular for improving traceability when parts of an object group are lost.

The article is organized as follows. The next section presents the background on coupled objects and discusses some related works. The third section details our resilient data structure for coupled objects, while the fourth section describes the building and verification algorithms. Finally, Section V concludes the paper.

II. Background

The coupled objects concept finds an application in many RFID approaches. However, due to some memory limitations in the RFID tags, retrieving the data stored in missing elements is difficult whithout using an external information system.

A. Coupled Objects principles

Coupled objects consist in groups of physical objects that are logically associated together, meaning that they carry digital information referencing other objects of the set, or representing their membership to the group.

The physical objects are associated to an RFID tag. Each member of a group is represented by a tag. The group representation uses some data distributed over the set of tags. The structure is self-contained in data stored in the memory of a set of tags. Because the memory size in the tag is limited and the integrity check should be fast, the group will be represented by a digest, computed by a hash code function. The digest will be stores in the memory of each tag of the group. This approach enables full autonomous operation of both the association points and the checkpoints.

The group building phase works as follows. Let us consider a set of n tags with unique identifiers $t_1, t_2, ..., t_n$. The idenfiers are ordered in a determined sequence (using a chosen order relation). Then, a hash function is applied to this information to compute the digest: $d = hash(t_1, t_2, ..., t_n)$. This hash value is used as a group identifier gid, stored in each tag of the set.

This hash value makes possible the integrity checking phase. Once a group is formed, integrity checks can be performed. The principle is to read all the tag identifiers t_i of the objects of a given group (sharing the same group id gid), and verifying that the $hash(t_1, t_2, ..., t_n) = gid$. If the computed hash does not match the gid stored in the tags, the group is considered as invalid.

B. Ubicheck application

A simple application scenario of coupled objects is *Ubi-Check* [1]: consider someone at the airport who is going to cross the security gate. The person is required to remove his jacket, belt, and put in a tray mobile phone, music player, remove laptop from his bag, and may be other objects... All that in hurry, with other people in the queue doing the same. Obviously, personal objects are vulnerable to get lost in this situation: objects can get stuck inside the scanner, can stack up on each other at the exit of the scanner, and it is easy to forget something while being stressed to get a flight. A coupled objects solution would secure the process by associating all personal items into a group, and checking it at the exit gate of the screening area. The same application could also work when leaving an hotel, a train, a plane, etc. We will use the Ubi-Check scenario in the rest of the paper to illustrate the problem of integrity failure raised by lost objects.

C. Dealing with integrity errors

When an integrity error is detected, this mean that one or more objects are missing from the group. In UbiCheck, this would trigger an alarm or a visual signal to notify the user. However, the system is not able to characterize the nature of the error, by listing the missing parts. A simple solution would be to store in an external information system a detailed list of the object group, but this would ruin all the advantages of the coupled objects architecture: autonomous operation, independance of external information system, privacy, etc.

Our goal is to design a data structure for the tags that would help to charaterize the missing element from the group, with the two following properties :

1) Autonomy: the necessary data to characterize the integrity error should be self-contained in the tags,
2) Resilience: the data structure should survive losing one or more objects.

The exact nature of the data used to interpret the integrity error can depend on the application. In the context of this paper, we will consider that objects of a group have interpretable identifiers, such as names, that would be used to enumerate the missing objects when an integrity failure is detected. For example, in Ubi-Check, each personal item would be associated to a user defined shortname such as 'wallet', 'phone', 'passport', etc. These names constitute the data to be protected by the distributed data structure stored in the tags, and we should be able to recover them when some objects are lost.

D. Related works

The problem we are addressing shares some similarities with data resilience methods that have been widely studied for storage devices and file systems. Related to the FRD concept [3], the RAID-6 approach protects disk storage systems from any disk failures. Unlike RAID-1 through RAID-5, which detail exact techniques for storing and encoding data to survive single disk failures, RAID-6 is merely a specification. The exact technique for storage and encoding is up to the implementor. Various techniques for implementing RAID-6 protect disk storage systems from two disks failures, such as EVENODD coding [4], Row Diagonal Parity coding [5], Liberation Codes coding [6]. These techniques for implementing RAID-6 have been developed and are based on erasure codes. There is no one standard for RAID-6 encoding.

The RAID implementation proposed in [7], [8] can restore some data lost after at most two disks failures. For restoring the data of two disk, this approach is based on the resolution of an equation system. This equation system is build by the definition of two parity functions, with two unknown variables, which are the two pieces of data lost on each of the two defective disks. It can be generalized to restore the data of n disks by considering n parity functions, to build an equation system of n equations and n unknown variables. Then, n disks are used to store the parity data to restore the disk failures. Like the data disk, the RFID tags store some data.

The RAID-5 approach distributes parity data along with the data. It is possible to distribute the parity data over the memory of each tag. The impact on the memory consumption of each tag becomes less important. It is simple to use a RAID approach to restore the data of the tags. This approach can be easily adapted for restoring some lost data on the RFID memory banks. The RAID approach can be used to rescue the data of n tags lost. However, when more than n tags are lost, nothing can be said about the missing tags: nothing can be deduces about the n first lost tags.

The RAID approach is used to secure the data, minimizing the overhead on storage space. The goal of our approach is not exactly the same. We want to able to recover some data that characterize the physical objects that have been lost, or the parts of a group that have been lost.

In the domain of the communication network, some approaches are based on the graph theory [9] to enforce the connectivity between network stations. In [10], the graph optimally connected is used to enforce the links between a set of stations, which be can viewed as a set of vertices of a graph. This approach enforces the connectivity in the network, when the disconnection of some stations occurs. Close to our RFID problem, a link between stations can be seen like the capacity to describe each tag data by the other tags. Enforcing the link between stations is a similar problem to enforce the link between tags.

III. Resilient data structure for coupled objects

As explained previously, when parts of a coupled object are lost, integrity failure is detected, but we need a resilient data structure to improve the traceability of the lost parts. This structure will be distributed over all the parts of the group, stored in tags memory. We assume that each part is described by an application-specific data item. For the purpose of simplicity, in the examples we will consider the Ubi-Check case where objects are described by short names.

The structure is based on the following design principles. First, the traceability mechanism should be able to identify the missing parts, so the structure should implement data redundancy. Second, the same memory space is allocated on each tag of the structure: the data distribution is balanced over the set of tags. Third, the robustness of the traceability mechanism should be independent of particular tags that may be lost; this means that if we want the structure to be able to resist to k tags lost in a coupled object of n parts, any of the k out of the n tags could be lost.

Our approach is based on the notion of regular graph from graph theory [9].

A. A graph representation for the data of RFID tags

In this model, a group of tags is modeled by a graph. Each RFID tag represents a vertex of a graph. Each tag stores in its memory some data about the neighbors: for example a nickame, which can be considered, as a short description. Each tag knows the nickname of its neighbor in the graph representation. These data are used to store the edges of the graph representation in the memory of the tags of the group.

Definition 1 (Graph): A graph G is a pair (V, E) comprising a set V of vertices, and a set E of edges, which is a set of pairs of vertices belonging to V.

Definition 2 (Regular graph): A regular graph is a graph where each vertex has the same number of neighbors: every vertex has the same degree. Let $k \in \mathbb{N}$. A regular graph with vertices of degree k is called a k-regular graph or regular graph of degree k.

The regular graphs of degree at most 2 are easy to classify: A 0-regular graph (Figure 1) consists of disconnected vertices, a 1-regular (Figure 2) graph consists of disconnected edges, and a 2-regular graph consists of a connected cycle (Figure 5) or a set of disconnected cycles (Figure 3). A 3-regular graph (Figure 4) is also called a *cubic* graph.

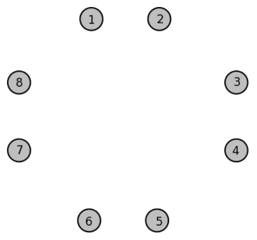

Figure 1. A 0-regular graph.

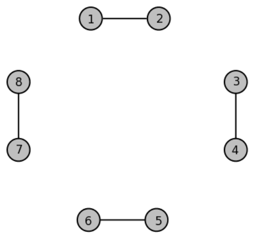

Figure 2. A 1-regular graph.

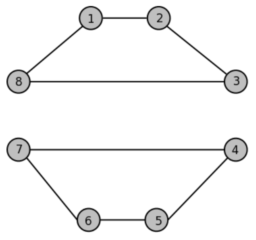

Figure 3. A 2-regular graph.

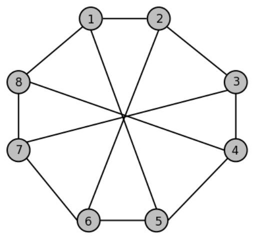

Figure 4. A 3-regular graph.

Let us consider the 2-regular graph G composed of 8 vertices presented on Figure 3. Let G_1 be the subgraph of G composed of the vertices 1, 2, 3, 8, and G_2 the subgraph of G composed of the vertices 4, 5, 6, 7. If the subgraph G_1 is deleted, then in the subgraph G_2, it does not exist a vertex for which one of its neighbors is missing. It is impossible to determine something about the missing vertices. In fact, it is impossible to deduce something by only considering the vertices 4, 5, 6, 7, because the subgraph G_2 is completly disconnected of the subgraph G_1. The graph structure should be connected to ensure a better tracability of the vertices by their neighbors in a graph representation.

Definition 3 (Path): In a graph, a *path* is a sequence of vertices such that from each of its vertices there is an edge to the next vertex in the sequence.

Definition 4 (Connectivity in graph): In a graph, the vertices of a path are said to be connected. Consider a graph $G = (V, E)$. If, for all vertices u and v of V, there exists a path from u to v, then G is connected.

Let us consider a graph $G = (V, E)$. This property ensures that, for all subgraph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$, there exists two vertices $v \in V'$ and $u \in V \setminus V'$, such that $(u, v) \in E \setminus E'$ (there is an edge of G between a vertex of V' and a vertex of $V \setminus V'$). A 2-regular connected graph is shown on Figure 5. The regular graph presented on Figures 1 and 2 are also not connected. The 2-regular graph shown in Figure 3 is not connected. A $k-$regular graph can be connected if $k \geq 2$.

A connected graph ensures that, when some vertices are deleted, at least one edge from one missing vertex to a present vertex is deleted. When a vertex is missing, it becomes easy to ensure that it is missing by using a simple graph exploration algorithm. The $k-$vertex-connected graph enforces the connection property in the $k-$regular graph.

Definition 5 ($k-$vertex-connected graph): A graph $G = (V, E)$ is said to be $k-$vertex-connected if the graph remains connected when you delete fewer than k vertices from the graph.

A a graph is $k-$vertex-connected, if k is the size of the smallest subset of vertices such that the graph becomes disconnected if you delete them. So, it is sure that when less than k vertices are deleted in the graph, it is still connected. A $k-$vertex-connected graph ensures that: when some vertices are deleted, at least k edges from the set of missing vertices to the set of present vertices are deleted. By the following Property 1, a $k-$regular graph structure can at most ensure a $k-$vertex-connectivity.

Property 1: A $k-$regular graph is at most $k-$vertex-connected.

Proof (sketch): In the example presented on Figure 5, the $2-$regular graph is also $2-$vertex-connected.
In a $k-$regular graph, each vertex can be disconnected from the graph by deleting at least k neighbors. In this case, the graph becames disconnected, and one subgraph is defined by only one vertex. ■

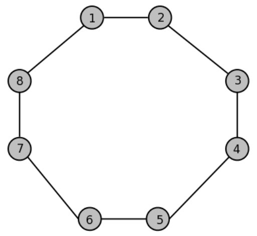

Figure 5. A 2-regular graph connected.

This structure of $k-$vertex-connected graphs induces the property of k-edge-connected.

Definition 6 (k-edge-connected graph): Let $G = (V, E)$ be an arbitrary graph. If $G' = (V, E \setminus X)$ is connected for all $X \subseteq E$, where $\mid X \mid < k$, then G is $k-$edge-connected. Trivially, a graph that is $k-$edge-connected is also $(k - 1)-$edge-connected.

Property 2: Let's $G = (V, E)$ a k-vertex-connected graph. Then, G is a k-edge-connected graph.

Proof: Let's a graph $G = (V, E)$, such that G is not $k-$edge-connected.

- $\Rightarrow$ There exists a set of edges $E' \subseteq E$, such that $\mid E' \mid < k$, and the subgraph $(V, E \setminus E')$ is not connected.
- $\Rightarrow$ There exists a set $V' \subseteq V$ such that $\mid V' \mid = \mid E' \mid$ and $V' = \{s \in V \mid \exists t \in (V \setminus V'), (s, t) \in E'\}$ for which the subgraph $(V', \{(u, v) \in E \mid (u \in V \setminus V') \vee (v \in V \setminus V')\}))$ is not connected.
- $\Rightarrow$ There exists a set of vertices V', such that $\mid V' \mid < k$ and the subgraph $(V \setminus V', \{(u, v) \in E \mid (u \in V \setminus V') \vee (v \in V \setminus V')\})$ is not connected.
- $\Rightarrow$ G is not $k-$vertex connected.

Thus, if a graph G is not $k-$edge-connected, then it is not $k-$vertex-connected.
It is equivalent to say that a graph G is always $k-$edge-connected or not $k-$vertex-connected (by definition of the implication relation).
Thus, if a graph G is $k-$vertex-connected, then it is $k-$edge-connected.

■

Then, the structure of $k-$regular graph k-vertex-connected is also $k-$edge-connected.

Definition 7 (Optimally connected graph): Let G be a regular graph of degree k. If G is indeed $k-$vertex-connected and $k-$edge-connected, then it is called an *optimally connected graph*.

Thus, the structure of $k-$regular graph k-vertex-connected and $k-$edge-connected is also called an optimally connected graph of degree k. Then, the graph presented on Figure 5 is a optimally connected graph of degree 2, and the graph presented on Figure 6 is an optimally connected graph of degree 3.

By using this representation of graph $k-$vertex-connected and $k-$regular (with $k \geq 2$) to the RFID application presented in Section II, each tag is modeled by a vertex of the graph. Each association of two tags is modelized by an edge of the graph. Two tags are associated and they define an edge, when each of them contained the nickname describing the other. So, each vertex (or tag) knows who are its neighbors. With this representation of regular graph, the same quantity of memory is used in each tag, and the existence of each tag is equivalently stored in the others. By the $k-$vertex-connection property of this graph, this structure ensures that in a graph $k - regular$:

- when a set of n tags is lost, with $n \in \mathbb{N}, n \leq k$, the structure ensures that exactly n tags are missing, and the n nickames are known,
- when some tags (some vertices of the graph representation) are missing, at least k neighbors can give the nickname of some missing neighbors.

When a neighbor of a tag in the graph representation is missing, it is possible to determine explicitly that it is missing by using the knowledge of each vertex on the identity of its neighbors. This knowledge is stored in the RFID tags memory.

For example on Figure 5, it is possible to ensure that:

- if the tag 1 is missing, then the remaining tags 2 and 8 lost a neighbor.
- If the tags 1 and 2 are missing, then:
  - the remainging tag 8 looses the tag 1 as neighbor,
  - the remaining tag 3 looses the tag 2.
- If the tags 1, 2 and 3 are missing, then:
  - the remaingings tag 8 looses the tag 1 as neighbor,
  - the remaining tag 4 looses the tag 3,
  - Nothing can be deduces about the tag 2, because its neighbors are also missing.
- If the tags 1, 2 and 5 are missing, then:
  - the remainging tag 8 looses the tag 1 as neighbor,
  - the remaining tag 3 looses the tag 2,
  - the remaining tag 4, 6 looses the tag 5 as neighbor.

B. Robustness of the structure and memory cost

The greater the degree of a graph is the more robust is the structure. More robust is the structure, the easier it is to rescue some data when some tags are lost. It is also easier to determine the missing tags. The greater the degree of a graph is, in consequence, the more costly this representation is costly in tag memory. Let us consider the scenario application of Ubi-Check of the traveler in an airport presented in Section II-B. A traveler have 8 objects: Phone, Wallet, Bag, Belt, Jacket, Passport, Watch, Laptop, that he considers as very important. The traveler decides to group this set of objects with the Ubicheck application based on a RFID solution. Each object is associated to an RFID tag. In the example presented on Figure 6, the traceability mechanism links the tags by using an optimally connected graph of degree 3.

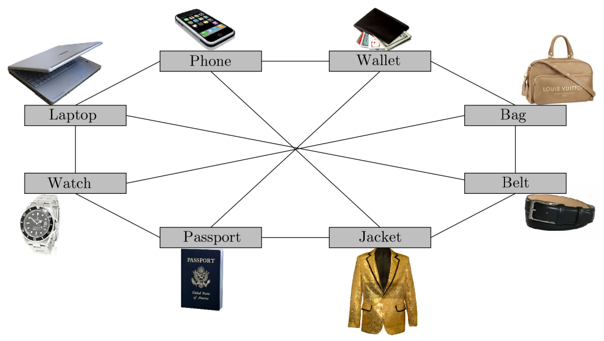

Figure 6. A 3-regular graph connected.

Each tag stores a nickname, which characterizes (for the owner) the object associated to it. To store the graph structure over the set of tags, each tag stores the nicknames of all its neighbors. Figure 7 represents the data stored in the memory bank of the tag associated to the phone (in the example of a $3-$regular graph presented on Figure 6).

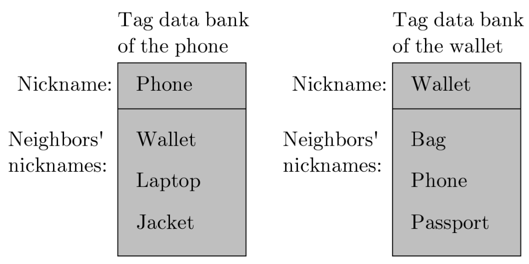

Figure 7. 3-regular graph representation in the tag data bank of the phone and wallet.

When the degree k of the graph representation increases, the robustness of the information also increases. But, increasing the degree also increases the memory space used in each tag for storing the graph structure. In fact, each tag stores an information about all of its k neighbors. So, the space used by the traceability mechanism is proportionnally increased when k increases.

With this optimal graph representation of degree 3, when at most 3 objects are lost, it is possible to explicitly list them. When more than 3 objects are lost, 3 of them can be listed. More generally, by using a graph representation of degree k, when at most k objects are lost, it is possible to explicitly list them. When more than k objects are lost, k of them can be listed.

For this example, the traceability mechanism can ensure that:

- if the phone is missing, then its absence is detected by the tag of laptop, wallet, and jacket.
- If the phone and wallet are missing, then:
  - the absence of the phone is detected by the tags of laptop, and jacket,

  - the absence of the wallet is detected by the passport, and the bag.
- If the phone, wallet and bag are missing, then:
  - the absence of the phone is detected by the tags of laptop and jacket,
  - the absence of the wallet is detected by the passport,
  - the absence of the bag is detected by the tags of watch and belt.
- If the phone, wallet, bag and passport are missing, then:
  - the absence of the phone is detected by the tags of laptop and jacket,
  - the absence of the bag is detected by the passport,
  - the absence of the passport is detected by the tags of watch and jacket,
  - nothing can be deduces about the missing wallet because in this 3-regular graph representation the three neighbors of the wallet are also lost.

As shown on Figure 8, when the traceability mechanism uses a regular graph of degree 3, four memory fields are used. For example, in the graph structure presented on Figure 6, the tag associated to the phone stores a nickname $phone$ that characterizes the object associated to it, and it also stores the nicknames of its neighbors: $Wallet$, $Laptop$, $Jacket$. In the same way, the tag associated to the wallet stores its nickname, and the nicknames of its neighbors.

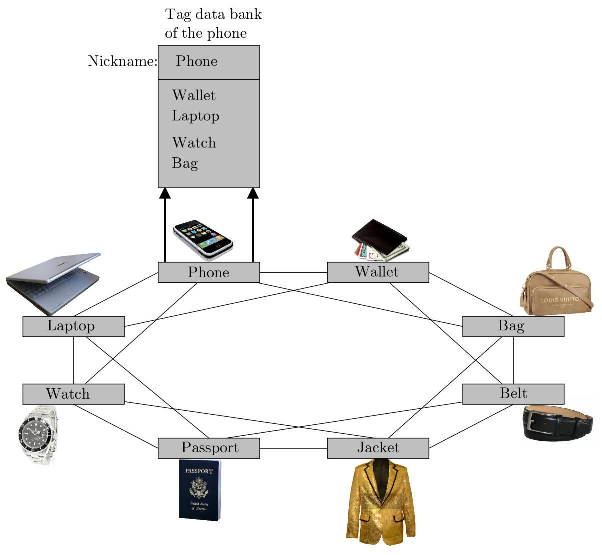

Figure 8. 4-regular graph representation in the tag data bank of the phone.

The characters can be translated in a hexadecimal value on two bytes. Let us consider, that the nicknames have at most 10 characters. In a 3-regular graph representation, each tag must store the three nicknames of these three neighbors. Then, it is necessary to use 60 bytes to store the nickname of the neighbors in 2-regular graph. More generally, for a $k-$regular graph representation, when the maximum length (in number of characters) of the nickname is equal to l, the memory space used to store the nickname of the neighbors of a tag is equal to: $2 * l * k$. Here, each tag is identified by a nickname, this a good way to extend the Ubicheck application. For other applications, it is also possible to save another information than a nickname.

IV. Building and verification algorithms

In this section, we present the algorithm of graph building, and the algorithm of graph structure checking.

A. Graph building

The graph building algorithm is used during the group creation phase of RFID tags, to store information about their neighbors in the graph structure.

It is essential to determine the necessary conditions to build a $k-$regular graph with a set of tags.

Property 3: Let $d \in \mathbb{N}$, and V a set a vertices. A regular graph $G = (V, E)$ of degree d exists, if and only if there exists $e \in \mathbb{N}$, such that $d <\mid V \mid$ and $d* \mid V \mid= 2 * e$.

$\Longrightarrow$:

Each vertex has at most $\mid V \mid -1$ potential neighbors. Then in a regular graph $G = (V, E)$ of degree d, $d <\mid V \mid$.

Let $d : V \mapsto \mathbb{N}$ the function that associates a vertex of V to its degree. In a simple graph each edge links two vertices. When the sum of the degrees is done, each vertices is counted twice: one time when the degree of the first vertex is counted, and a second time when the degree of the other vertex is counted. Then, in a simple graph: $\sum_{v \in V} d(v) = 2* \mid E \mid$.

In a regular graph of degree d, all the vertices have the same degree d, then $d* \mid V \mid= 2* \mid E \mid$. Thus, if a regular graph $G = (V, E)$ of degree d exists, then there exists $e \in \mathbb{N}$, such that $d* \mid V \mid= 2 * e$ and $d <\mid V \mid$. ■

$\Longleftarrow$:

Let a set of vertices V, and $d, e \in \mathbb{N}$ such that $d <\mid V \mid$ and $d* \mid V \mid= 2 * e$. Let us consider the set of vertices V as points regularly placed on a circle (as the vertices of a regular $n-$gon, with $n =\mid V \mid$). Let a set of edges E defined as follow:

- if d is even (the assumption: $d * n$ is even, is satisfied), then each vertex is connected to $d/2$ vertices that are after it as well as $d/2$ vertices that are before it,
- if d is odd and $\mid V \mid$ is pair (that satisfies the assumption: $d* \mid V \mid$ is pair), then each vertex is connected to $(d-1)/2$ vertices that are after it as well as $(d-1)/2$ vertices that are before it, and all the diagonals (those connecting two diametrically opposite points) of the $n-$gon are adding.

Then there exists a regular graph $G = (V, E)$ of degree d. ■

The algorithm 1 builds a regular graph of degree d from a set of vertices V, with $n =\mid V \mid$. To ensure the existence of the regular graph, it is assumed that $d <\mid V \mid$ and $d* \mid V \mid= 2 * e$ with $e \in \mathbb{N}$ (Property 3). The algorithm 1 works as follow. Let's us consider n vertices as points regularly placed on a circle (as the vertices of a regular $n-$gon). So, if d is even then each vertex is connected to $\lfloor d/2 \rfloor$ vertices that are after it as well as $\lfloor d/2 \rfloor$ vertices that are before it. And if d is odd, all the diagonals (those connecting two diametrically opposite points) of the $n-$gon are adding. In the case where d is odd, it is necessary that n is pair to satisfy the existence condition: $d * n$ is pair. In this case, each vertex is also connected to $\lfloor d/2 \rfloor$ neighbors before it, and to $\lfloor d/2 \rfloor$ neighbors before it.

Algorithm 1 Algorithm of building a graph optimally connected.

```
  $d$ : **degree of the graph**
  $V$ : **a set of vertex**
**Ensure:** $(d <\mid V \mid)$ **and** $((d* \mid V \mid)$ is pair)
  $set$ : **Array of tags**
  $nbTags$ : **Integer**
  $j$ : **Integer**
  $E$ : **a set of edge**
  $V \leftarrow \emptyset$
  $E \leftarrow \emptyset$
  $set \leftarrow HW\_read()$
  $nbTags \leftarrow set.length$
  **for** $i = 0$ **to** $nbTags$ **do**
    {//for each tag a vertex is added in $V$}
    add a new vertex $set[i]$ in $V$
    **for all** $j$ such that $j \neq i$ **and** $j \in [i - \lfloor \frac{d}{2} \rfloor; i + \lfloor \frac{d}{2} \rfloor]$ **do**
      {//all the edges between the vertex $i$ and the $d/2$ vertices that are after him are added to $E$}
      {//all the edges between the vertex $i$ and the $d/2$ vertices that are before him are added to $E$}
      **if** the edge $(set[j \textbf{ modulo } nbTags], set[i]) \notin E$ **then**
        add the new edge $(set[i], set[j \textbf{ modulo } nbTags])$ in $E$
      **end if**
    **end for**
    **if** $d$ is odd **then**
      {//the edge between the vertex $i$ and its diametrically opposite points is added to $E$}
      add the new edge $(set[i], set[(i + \lfloor \frac{nbTags}{2} \rfloor) \textbf{ modulo } nbTags])$ in $E$
    **end if**
  **end for**
  **return new Graph(**$V$,$E$**)**
```

$HW\_read()$ represents the inventory method provided by the RFID reader to return the set of RFID tags detected by the reader.

Let's a $k-$regular graph $G = (V, E)$. With the building principle of algorithm 1, it is impossible to delete less vertices for disconnecting a set of vertices of the $n-$gon, than for deleting only one vertex. The algorithm 1 builds a regular graph of degree d. Then, the d neighbors of a vertex must be deleted to disconnect one point. Thus, a regular graph of degree d built with the algorithm 1 is $d-$vertex-connected. Thus, the algorithm 1 builds some optimal graphs.

B. Checking graph

Algorithm 2 checks the integrity of an RFID structure modelized by a $k-$regular graph. For each vertex in a set $\mid V \mid$, the algoritm search all its neighbors in the same set $\mid V \mid$. The missing neighbors of all tags are stored in the set $\mid S \mid$. This set stores some information to depict the missing vertex. If S is empty, then no vertex is missing, the integrity of the graph is alright.

More formally, the algorithm is described as follows:

Algorithm 2 Algorithm of checking.

```
  $S$ : **set of nicknames**
  $V$ : **set of tags or set of vertices in the graph representation**
  $E$ : **a set of edges**
  $S \leftarrow \emptyset$
  **for all** $v \in V$ **do**
    {//The presence of all the neighbors of each vertex $v$ is tested}
    **for all** $n \in neighbors(v, E)$ **do**
      **if** $n \notin V$ **then**
        {//The nickname of the missing neighbor is added to $S$}
        add the nickname of $n$ in $S$
      **end if**
    **end for**
  **end for**
  {//If $S = \emptyset$ then no object is missing,}
  {//else $S$ contains the nicknames of the missing objects.}
  **return** $S$
```

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a resilient data structure for coupled objects to help integrity error diagnosis: in particular, it can be used to identify the missing elements of a group of objects. The structure is stored in tags memory, in line with the idea of coupled objects to support autonomous operation. The robustness of the structure can be increased at the expense of memory overhead, making the structure configurable to application requirements.

Beside the Ubi-Check application described in the paper, we are considering other application scenarios where integrity checking of coupled objects should be diagnosed in case of errors. For example, it could be used to secure a medical prescription with a set of drugs. In case of an integrity error, it would be important to charaterize the nature of the error, such as identifying a missing drug. Some perspectives to this work include supporting dependency properties between elements inside a coupled object, in order to better charaterize integrity errors, or to characterize the global properties of a composite object when some elements are missing.

References

[1] M. Banâtre, F. Allard, and P. Couderc, "Ubi-check: A pervasive integrity checking system," in *NEW2AN*, 2009, pp. 89–96.

[2] F. Allard, M. Banâtre, F. Ben Hamouda, P. Couderc, and J.-F. Verdonck, "Physical aggregated objects and dependability," Avalaible: http://hal.inria.fr/inria-00556951 [Last accessed in June 28, 2012], INRIA, Research Report RR-7512, Jan. 2011.

[3] J.-C. Fabre, Y. Deswarte, and L. Blain, "Tolérance aux fautes et sécurité par fragmentation-redondance-dissémination," *Technique et Science Informatiques (TSI)*, vol. 15, no. 4, pp. 405 –427, 1996.

[4] M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: an optimal scheme for tolerating double disk failures in raid architectures," *SIGARCH Comput Archit News*, vol. 22, no. 2, pp. 245–254, 1994.

[5] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proceedings of the 3rd USENIX Symposium on File and Storage Technologies (FAST'04*, 2004, pp. 1–14.

[6] J. S. Plank, "The raid-6 liberation codes," in *FAST-2008: 6th Usenix Conference on File and Storage Technologies*, 2008, pp. 97–110.

[7] H. Anvin, "The mathematics of raid-6," 2011. [Online]. Available: http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf

[8] I. S. Reed and S. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[9] F. Harary, *Graph Theory*. Reading, MA: Addison-Wesley, 1969.

[10] B. Myers, "Optimally connected communication networks with maximum diameters," *Electronic Circuits and Systems, IEE Proceedings G*, vol. 128, no. 6, pp. 289 –292, December 1981.