Online monitoring and diagnosis of RFID readers and tags

Rafik Kheddam, Oum-El-Kheir Aktouf, Ioannis Parissis
Grenoble Institute of Technology, LCIS
Valence, France
firstname.lastname@lcis.grenoble-inp.fr

***Abstract*—The need to fault-tolerance in RFID systems is increasing with the increased use of this technology in critical domains such as real-time processing fields. Although many efforts have been made to make this technology more dependable, it is still unreliable. In this paper, we propose a complementary approach to existing ones. It consists of a probabilistic monitoring that detects failures of RFID system components and a verification process that refines the diagnosis process by finding the causes of the detected failures.**

Keywords—finite state machine; model based diagnosis; fault-tolerance; dependability; probabilistic diagnosis

1. INTRODUCTION

A Radio Frequency Identification (RFID) system is composed of three main components: readers, tags and user applications. Readers are devices that use radio waves to communicate with the tags. Tags are electronic labels with memories that uniquely identifies given items. User applications are programs that use the RFID data according to user predefined rules. In large-scale deployment such as a baggage handling system of an airport, we can find another component called RFID middleware that processes the huge volume of raw data coming from RFID devices and translates it to business events for the user applications.

RFID technology has the advantage of remotely identifying objects "without any need to line of sight". Therefore, it is going to replace the infrared based technology such as barcode system. Figure 1 shows the architecture of an RFID system and its runtime operation.

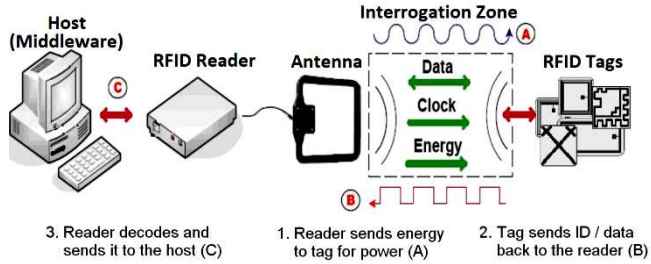

Figure 1. RFID System [1]

In recent years, RFID systems have been used in critical domains such as medical fields or transportation systems. Many techniques have been used to enhance the reliability of this technology. These techniques are summarized in two categories [2]. *Reader Status Monitoring* and *Reader Performance Monitoring*. Reader Status Monitoring is an intrusive monitoring, where the middleware uses SNMP requests such as *ping command* to check if the readers are powered and connected to the network, if its antennas are operating, *etc*. Reader Performance Monitoring is a non-intrusive monitoring, where the middleware processes performance parameters such as Read Error Rate (RER) [3], Read Error to Total Reads (RETR)[1], Average Tag Traffic Volume (ATTV) of a given reader, Error Frequency, *etc*. to detect reader failures using the variation of these processed parameters (this monitoring requires a learning phase).

All these techniques present some drawbacks. Reader Status Monitoring can only detect if the monitored reader is operational or not. But, it cannot detect a failure resulting from damaged reader antennas or disturbed environment whose consequence is an increase of read errors of the reader. Indeed, RFID readers are very sensitive to their environment. So, the data streams produced by the readers are error-prone. It is admitted that the read rate of current RFID systems in a real word deployment is in 60-70% range [4] [5]. On the contrary, Reader Performance Monitoring can detect such failures, but without going further. Hence, when a reader has an abnormal performance parameter value such as high RER, it is not possible to determine whether the problem comes from the reader, the tags or the environment.

This paper presents a new diagnosis approach that proceeds in two steps. The first one combines the Reader Performance Monitoring with a statistical analysis of the read results of all readers that have processed the same tags to identify *faulty readers* or *faulty tags*. In the second step, when a reader is diagnosed as faulty, a model based diagnosis is used in order to find the causes of this failure.

The rest of this paper is organized in two main sections. The first one introduces the proposed RFID monitoring techniques (a probabilistic diagnosis process and a state verification process). The second one presents and discusses different implementation propositions.

2. RFID SYSTEM DIAGNOSIS

The proposed approach consists in two diagnosis mechanisms: *Probabilistic Monitoring* and *Model Based Diagnosis* that can be used together or separately. The first one aims to detect any abnormal behavior of RFID devices

[1] RER (*resp.*, RETR) is the number of erroneous reads made by a reader to identify **each** (*resp.*, **all**) tag(s) located within its field of view.

such as “a reader is not responding”, “a high error frequency of a reader”, “some tags are not read”, *etc.* The second mechanism aims to find the origin and the causes of the failures.

2.1 Probabilistic Monitoring

Probabilistic Monitoring is an approach that monitors the tags and the readers reading the same set of tags by comparing their performance parameter values. To do this, we first associate to each reader a performance value (a learning phase is needed) with some standard deviation that will be used as a basis of comparison of the future results of this reader as it is done in [6]. If a reader does not match its associated performance parameter, it is declared faulty. If most readers don’t match their associated parameters according to a given tag or group of tags, then this or these tags are declared faulty. This failure can be due to (i) an incompatibility between the readers and the tags such as “High Frequency readers that try to read Low Frequency tags”, (ii) a disturbed environment or (iii) faulty tags.

Probabilistic Monitoring consists of three steps as shown in Figure 2. First, all readers that process the same tags are gathered into the same group. Then, results of all these readers are compared in order to detect faulty ones and / or faulty tags. Finally, failure probabilities are determined for each identified faulty component. These three steps are described in the following.

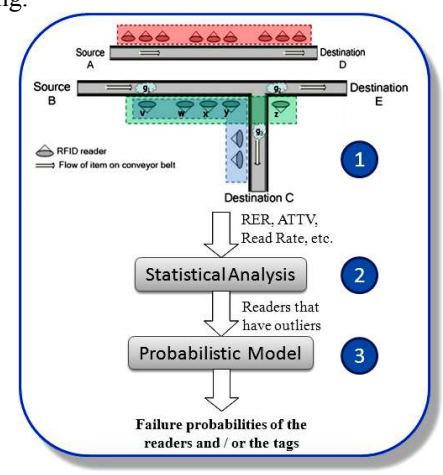

Figure 2. Probabilistic Monitoring Steps

2.1.1 Reader Partition

This step consists of identifying all readers that will analyze the same tags to compare their read results. As an example, Figure 3 emphasizes on how to gather the readers according to the origin and the destination of the tags (luggage) in a baggage handling system. So, the readers $\{R_1,R_2,R_3,R_4,R_5\}$ will be analyzed together according to all the tags that follow the path $\{B,E\}$, the readers $\{R_1,R_2,R_3,R_4,R_6,R_7\}$ will be analyzed according to the tags of the path $\{B,C\}$ and so on.

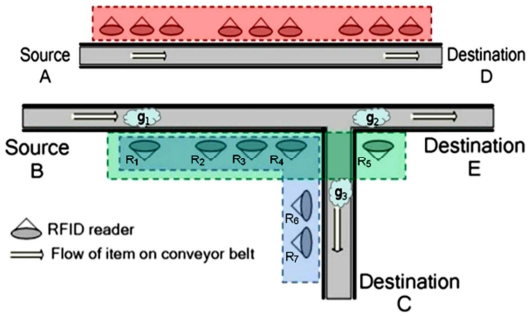

Figure 3. Reader Partition

2.1.2 Reader Results Comparison (RRC)

This step consists of comparing the results of all readers that have analyzed some tags at a given time. Table 1 shows an example of reader results comparison, where readers from different paths are analyzed simultaneously to get global and consistent diagnosis decisions as some readers may belong to different paths. For sake of simplicity, Table 1 indicates if a given reader R_i matches its associated performance parameter ($M(R_i,g_j)=1$) or not ($M(R_i,g_j)=0$) according to each tag or group of tags g_j (details of used performance parameter are not important here). $S(g_j)$ indicates which component (reader or group of tags) is faulty when analyzing the group of tags g_j; $R_i \in S(g_j)$ if R_i has a result ($M(R_i,g_j)$) different from that of the majority of the other readers. For example, $R_2 \in S(g_2)$ because $M(R_2,g_2)=1$ and all the other readers belonging to the same path, *i.e.* $\{R_1,R_3,R_4,R_5\}$, have $M(R_i,g_2)=0$, $i \in \{1,3,4,5\}$. In addition, g_2 is indicated as faulty because of the results of the majority of the readers. $SF(\,)$ indicates which reader is faulty; $R_i \in SF(\,)$, if $\exists\, g_j$, $R_i \in S(g_j)$. $SF(\,)$ can be adjusted so as to minimize false-positives, *e.g.* a reader R_i will be declared faulty only if it has been declared faulty twice according to two different groups of tags. In this example, R_7 will not be declared faulty in $SF(\,)$.

Table 1. Reader results comparison

M	R_1	R_2	R_3	R_4	R_5	R_6	R_7	**$S(g_j)$**
g_1	1	0	1	1	1	-	-	$\{R_2\}$
g_2	0	1	0	0	0	-	-	$\{R_2, g_2\}$
g_3	1	1	1	1	-	1	0	$\{R_7\}$
SF( )								$\{R_2, R_7\}$

At this stage of the monitoring, we can determine the probability of failure of each tag. For example, g_2 is analyzed by five readers and declared faulty by four of them, so its failure probability is *4/5*. In addition, the temporary failure probability of each reader is estimated. These probabilities will be used by the proposed Probabilistic Model to process the real probabilities of the faulty readers; *e.g.*, R_2 is declared faulty twice, each time, there are four readers that disagree with it. So, its temporary failure probability is $(4/5 + 4/5) / 2$.

2.1.3 Probabilistic Model

In this step, a failure probability is associated to each faulty reader. This failure probability is calculated on the basis of the failure probabilities of the other readers, the number of processed groups of tags, and the failure identification during Reader Results Comparison (RRC). This approach is inspired from the probabilistic diagnosis of multiprocessors systems [7].

RRC aims to identify:

- a real fault-free reader as fault-free (Correct Negative).
- a real faulty reader as faulty. In other words, RRC tries to avoid the case where a faulty reader is declared fault-free (False Negative).

We call this ability to correctly identify the state of the readers (faulty or fault-free) by ***Identifiability*.**

a) Correct Negative

Let *n* be the number of readers analyzing the tags, *p* the probability of failure of a reader (we have considered that each reader has the same failure probability for sake of readability of the formula below) and *C(x,y)* the binomial coefficient.

A fault-free reader is identified as such in RRC after processing of *t* tags if:

- At least half of all readers are fault-free and their results are the same as the one of the reader under analysis. This case is represented by the following probability

$$\sum_{i=\lceil \frac{n}{2} \rceil}^{n-1} C(i, n-1) \times (1-p)^i \times p^{n-1-i} \qquad (1)$$

- At least half of all readers are faulty, but for each group of tags, there are at least half of all readers that correctly process this group, and then have the same result than the reader under analysis. This case is represented by the following probability

$$\sum_{i=\lceil \frac{n}{2} \rceil}^{n-1} \left[C(i, n-1) \times p^i \times (1-p)^{n-1-i} \times \left(\sum_{j=i+1-\frac{n}{2}}^{i} C(j, i) \times (1-r)^j \times r^{i-j} \right)^t \right] \qquad (2)$$

With *r* being the probability that a faulty reader will process the tags incorrectly [2] (*i.e.*, the probability that the failure appears on the system).

If *CN*(*t,n,p,r*) is the probability to have a correct negative, then

$$CN(t, n, p, r) = (1) + (2)$$

b) False Negative

A faulty reader will be considered fault-free after it has processed *t* tags if:

- The reader under analysis processes correctly the *t* tags. So, it has the behavior of a fault-free reader. The probability of this case is represented by the following formula

$$CN(t, n, p, r) \times (1-r)^t \qquad (3)$$

- The reader under analysis *x* processes incorrectly *i* tags ($1 \leq i \leq t$), but for each one of them, there are at least half of all readers which are faulty and have the same result as the one of *x*. This case is represented by the following probability

$$\sum_{i=1}^{t} \left[\left(\sum_{j=\lceil \frac{n}{2} \rceil}^{n-1} C(j, n-1) \times (p \times r)^j \times \left((1-p) + p \times (1-r) \right)^{n-1-j} \right)^i \times C(i, t) \times r^i \times (1-r)^{t-i} \times CN(t-1, n, p, r) \right] \qquad (4)$$

If *FN*(*t,n,p,r*) is the probability to have a false negative, then

$$FN(t, n, p, r) = (3) + (4)$$

So, the ***Identifiability*** is processed as follows

$$\boldsymbol{I(t, n, p, r) = (1-p) \times CN(t, n, p, r) + p \times (1 - FN(t, n, p, r))}$$

In the remaining sections, a complementary diagnosis approach is presented to allow a precise identification of reader failure causes.

[2] In some cases, a faulty reader will correctly identify the tags in its field of view (*e.g.*, the damaged antenna is not used, the path that contains the error in its integrated circuit is not used, *etc.*).

2.2 Model Based Diagnosis

This approach is based on the communication standard between RFID readers and RFID middleware to monitor the behavior of the readers. This standard is called Low Level Reader Protocol (LLRP). In this approach, we propose to extend this protocol to make it able to handle RFID failures. Indeed, LLRP is a complex and complete communication protocol with error notifications, but it is not able to detect misconfiguration errors neither to locate the origin and the causes of the failures.

As shown in Figure 4, RFID readers have two main functions: tag inventory (*i.e.*, the identification of all tags located in the reader field of view) and tag memory access. These two functions are provided by LLRP using two main XML based messages: Reader Operation Specification (ROSpec) and Access Specification (AccessSpec). An ROSpec is responsible of triggering AccessSpecs. ROSpec has an identification number, a state (Disabled, Inactive or Active), a priority and a set of commands and parameters required by the reader to perform a correct inventory. AccessSpec has an identification number, a state (Disabled or Active), an ROSpecID that specifies the corresponding ROSpec, a set of conditions (TagSpec) that specify on which tags this AccessSpec is applicable and a set of commands and parameters required by the reader to access correctly the different memories of the tags.

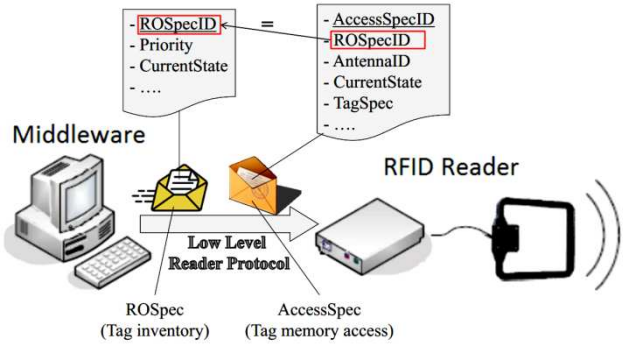

Figure 4. LLRP Runtime Operation

LLRP operates as follows. The middleware can define and send several ROSpec and AccessSpec (always in disabled state) to the reader. Each AccessSpec is affected to one ROSpec that will be responsible of triggering it. At a given time, only one ROSpec is under execution. Figure 5 shows the behavior of a reader "with one ROSpec" in a three-state FSM. The transitions in this FSM are labeled "*X/Y*", where *X* represents the input of the reader (usually a request coming from the middleware) and *Y* is the output of the reader (usually a response to the middleware request). An ROSpec moves to "Inactive" state when the reader receives an "Enable(ROSpecID)" request, then waits for a start condition such as an event from a motion sensor or simply a request from the middleware, to start the inventory process (the ROSpec is then in "Active" state). When some tags are identified, the reader checks if these tags match any TagSpec of an "Active" AccessSpec belonging to the running ROSpec (an AccessSpec moves to "Active" state by "Enable(AccessSpecID)" request from the middleware). If so, the reader performs the specified operations in the AccessSpec on the selected tags (tag memory access).

LLRP is very flexible. It gives the user the full power in the definition of the Reader Operation and Tag Memory Access specifications. But this usually leads to mis-configuration errors such as "the user has activated the wrong ROSpec", "the AccessSpec is not linked with the right ROSpec", *etc.* Effects of this kind of errors can be "the reader doesn't identify all tags located in its field of view", "the reader doesn't retrieve the needed information from the tag memories", *etc*.

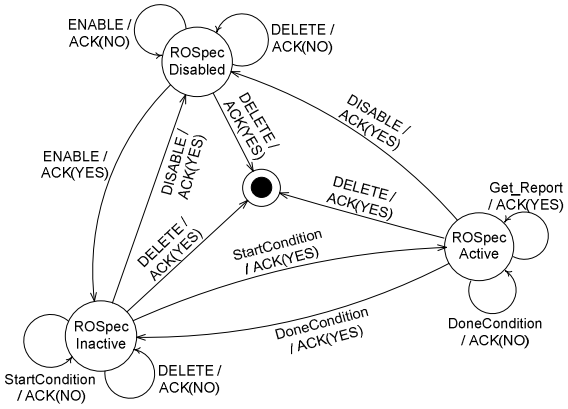

Figure 5. A three state FSM of a reader

2.2.1 LLRP failures

The set of failures that we are interested in, are classified according to their causes into two categories.

a) Failures due to inconsistent design or misconfiguration

- Masking of useful data or even blocking all data by the middleware.
- Mismatch between the received and the expected data by the middleware or the reader.

b) Failures due to runtime conditions

- Slow execution / component overload.
- No data capture.
- Erroneous received data.

2.2.2 Failure Processing

As said before, RFID readers are inherently inaccurate. This usually leads to the transmission of erroneous data to the RFID middleware. The approach we propose to deal with the

aforementioned failures consists of two mechanisms. The first one uses a state verification process using the existing LLRP model to deal with the failures due to inconsistent design and misconfiguration. The second one extends the LLRP model to deal with the failures not covered by the first mechanism (*i.e.*, failures due to runtime conditions).

a) Using the existing LLRP model

Failures due to inconsistent design or misconfiguration are caused by the fact that one of the participants (middleware or reader) in the communication is in the wrong state; *i.e.*, the middleware (or the reader) is in a state where it does not expect the received commands or data. Then, as a consequence, although they are useful data, they will be ignored and an error message may be returned. For example, let's assume that a reader is in a state where the ROSpec is disabled instead of being inactive (the user omits to enable the ROSpec or the reader does not interpret correctly the "Enable" request of the middleware). So, when some tags enter in the field of view of the reader, the motion sensor sends a signal to the reader in order to start the tag inventory. But, this signal is just ignored and the tags are not read.

This kind of failures is handled without any need to extend the existing LLRP model. All we have done is translating the textual specification of LLRP protocol into a finite state machine (FSM). This FSM is used with usual test sequence generation techniques ("Distinguishing Sequence") to deal with these failures. A distinguishing sequence is a set of transitions that allow identifying the initial and current state of the reader or the middleware [8]. This is useful to deduce the exact configuration of the reader (*i.e.*, number of ROSpec and AccessSpec, the state of each one of them, which AccessSpec belongs to which ROSpec, *etc.*) when the failure occurs. This FSM consists in 18 states and more than 200 transitions. For sake of simplicity, we will only show how to exploit the Distinguishing Sequence technique in an example to identify the failure causes.

To show how the Distinguishing Sequence technique works, let's take a more simplified FSM such as in Figure 6. To identify the state of the reader or the middleware when the failure occurs, we construct the successor tree of this FSM. A successor tree shows the behavior of an FSM starting from all possible states under all possible input sequences.

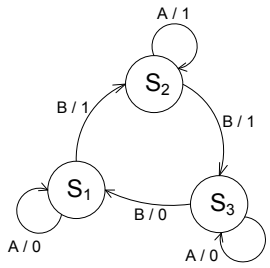

Figure 6. Arbitrary three-state FSM

Figure 7 shows an example of a successor tree built up from the FSM of Figure 6. It shows the transition sequences that distinguish between the different states of the FSM. When a failure occurs, the FSM can be in any state. So, the objective is to identify this state.

In this successor tree, the input *B* separates the state S_1 from S_2 and S_3; *i.e.*, if we have an output "0" (*resp.*, "1") after applying the input *B*, we are sure that the current state of the FSM is S_1 (*resp.*, S_2 or S_3). Then, the input *A* separates the state S_2 from S_3. At the end of this state verification process, if we have a sequence such as {*B/1, A/0*}, then, we are sure that the current state of the FSM is S_3 and the **initial state** at the launch of this verification process is S_2, *i.e.*, the unique state from which the sequence {*B/1, A/0*} is possible. So, the sequence {*B/1, A/0*} is a distinguishing sequence of the state S_2. Other examples of distinguishing sequences are: {*A/1*} for S_2, {*A/0, B/0*} for S_3, {*A/0, B/1*} or {*B/1, B/1*} for S_1, *etc.*

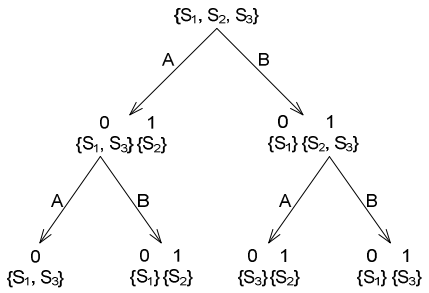

Figure 7. The successor tree of the arbitrary FSM

Now, let's see how to use this technique in a real case. As an example, to detect the cause of the failure "Masking of useful data by the reader" such as "no tag memory content" (*i.e.*, the middleware receives only the Tag Identifiers but not their memory contents), the middleware sends to the reader specific commands using the technique of distinguishing sequence to check if this error is due to a wrong internal state of the reader; *i.e.*, the AccessSpec is not executed to access the tag memories. At the end of the diagnosis, we deduce that the reader's initial state was one of the following, which is the cause of the failure.

- *Case 1:* ROSpec is "Active" but not the AccessSpec.
- *Case 2:* ROSpec and AccessSpec are "Active" but the AccessSpec is not in the list of AccessSpec to be triggered by the running ROSpec.
- *Case 3:* ROSpec and AccessSpec are "Active" but the identified tags do not satisfy the triggering conditions of the AccessSpec.

b) Extending the LLRP model

In this approach, we propose to extend the LLRP model to handle the failures caused by the running conditions.

1) Slow execution / component overload

These failures may be caused by the amount of raw data to process that leads the middleware to slow down. We should know that the middleware associates to each new reader an instance of the LLRP FSM to permanently maintain consistent communications with it. So, to check that the source of this problem comes from the huge amount of data that are fed back to the middleware, the middleware just needs to read all the instances of LLRP FSM looking for all the readers that are in "communicating state". Then, it only remains to verify that the number of the current communicating readers does not exceed the capacity of the middleware. If it does, as a possible solution, the middleware will ask the readers which have internal memories to pause the communication and to temporary store the data in their memories.

2) No data capture / erroneous received data

These failures are also caused by inconsistent design and misconfiguration, but they may also be caused by:

- The reader antenna is damaged or broken.
- The presence of external disturbances (radio waves of other devices, water, metal objects, etc.).
- The tags are out of the reader field of view or the signal power level of the reader is too weak.
- The tags move too fast. So, the reader hasn't enough time to identify all tags.
- The type of tags is not supported by the reader or on the contrary, the selected air protocol is not supported by the tags.

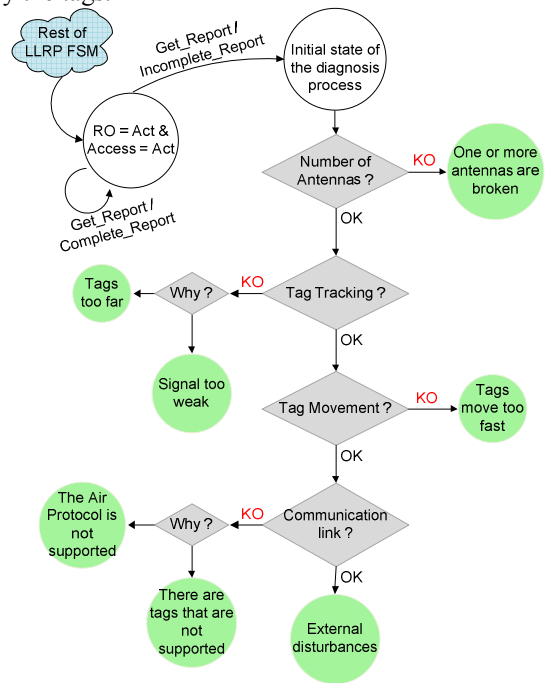

Figure 8. LLRP failure diagnosis diagram

Figure 8 emphasizes the LLRP extension. The lozenges in the diagram represent the inputs in the FSM, and the leaves (green circular shapes) are the causes of the failure.

As we can see in Figure 8, when the reader is executing an ROSpec with an AccessSpec (ROSpec and AccessSpec are active), the middleware may ask for the result of these specifications. If the report is incomplete (*e.g.*, there are tags that are not inventoried correctly, impossible read / write of tag memories, *etc.*), the middleware triggers the diagnosis process. The failure causes are hierarchically sorted according to their diagnosis difficulties. So a "broken antenna" is easier to diagnose than a "faulty communication link" as we will see below:

- The middleware can verify if an antenna is damaged or broken by asking the reader to give it back the number of "operational" antennas. If the reader is not able to perform this action and knowing that most readers have at least two antennas (one for transmission and one for reception), the middleware can transmit a signal by an antenna to receive it by the other one. If this operation succeeds, we assume that the antennas are not broken. To verify if one of them is damaged, the middleware will vary the receive sensitivity[3] or the signal power level of the antenna to determine the communication scope of the antenna as a damaged antenna will cover a smaller area. So, it is possible to deduce if the antenna is damaged or not (the distance between the antennas must be known).
- For the second cause of failure (*i.e.*, "the tags are out of the scope of the antenna"), the middleware will locate the tags by varying the power level or the receive sensitivity of the antenna as explained above. Figure 9 shows an area containing four readers, and how to exactly locate a tag to verify if it is out of the scope of a reader. For example, when the tag T_2 is not identified by the reader R_1 while it should, the middleware will exploit the neighbors of R_1 to locate the tag T_2 by reducing the scope of these readers enough to deduce that T_2 is out of R_1 field of view.

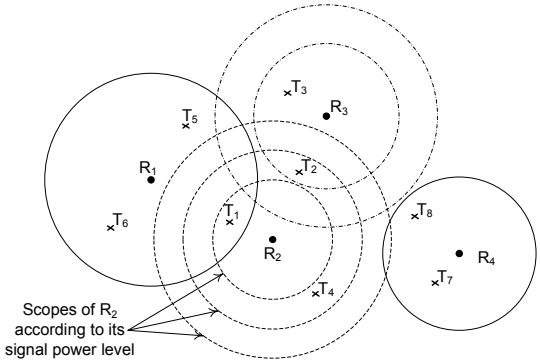

Figure 9. Tag localization technique

[3] Receive sensitivity indicates how faint a radio frequency signal can be successfully received by a given receiver. The lower the power level that the receiver can successfully process, the better the receive sensitivity [9].

- To deduce that the failure is due to tags that move too fast, the middleware can proceed as follows:
  - If the tags are carried by a conveyor belt, the middleware has just to know the speed of the conveyor belt and to compare it with the maximum speed supported by the reader.
  - The second possibility is to calculate the speed of the tagged item using its two previous positions (or three, to calculate the tag speeding), also by using the neighbors of the reader under analysis.
    For example, if reader R_1 in Figure 10 reads the tagged item at a given time, and R_2 reads it 2 seconds later, the middleware can process the item speed (2.5 *m/s*), so it knows when this item will probably appear in the scope of R_3 and with what speed will pass it. The middleware can also deduce the tag acceleration (if the tag speed is changing) by knowing the spent times between at least 3 positions.

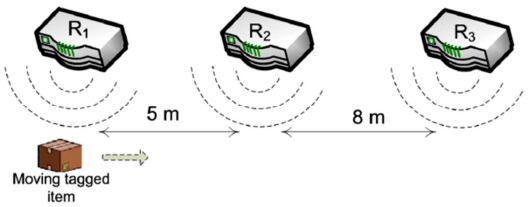

Figure 10. Tag speed calculation

- To detect the failure of the communication link, we proceed as follows:
  - If erroneous data are received by the middleware, then we are sure that the problem comes from the communication link between the reader and the middleware otherwise the erroneous data will be filtered by the reader.
  - To deduce the causes of the problem when it comes from the communication link between the reader and the tags, the middleware will ask the neighbors of the concerned reader which air protocol they use to successfully identify the tags. So, the middleware can deduce that the reader has used a non-supported protocol. Otherwise, the middleware will check if the reader does not support the concerned tags (*e.g.*, UHF reader that try to read HF tags) always by asking its neighbors.

If none of these causes are found, we assume that the failure is due to external disturbances such as the presence of metal objects between the reader and the tags.

3. IMPLEMENTATION AND EVALUATION

We have analyzed the behavior of the *Identifiability* by varying each parameter to show the impact of this latter on the *Identifiability* (we recall that the *Identifiability* is the ability of correctly diagnosing the readers). Figure 11 shows an example of the impact of the parameters *t* (number of analyzed tags) and *r* (probability that the failures of a reader have visible effects on the system). Figure 12 shows another example where the increase of the number of readers decreases the false negatives (faulty readers that are not detected). So, the *Identifiability* increases.

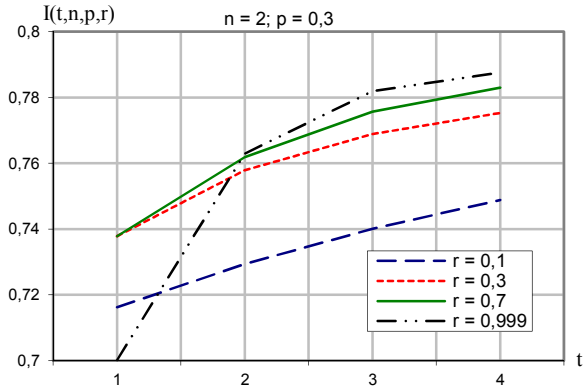

Figure 11. Identifiability variation with the number of tags

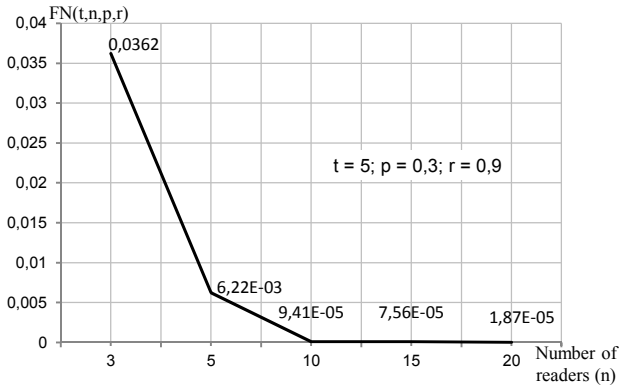

Figure 12. False Negative variation according to the number of readers

Without showing all results of this simulation, we can say that the *Identifiability* increases with the increase of the parameters *n, t, r* and the decrease of the parameter *p*. This is consistent because when *p* is high, most readers are faulty. So, they will be considered as fault-free and the fault-free ones as faulty.

We have implemented the probabilistic diagnosis approach in three ways according to the moment where the reader results comparison is triggered.

3.1 "Calculation done at the end" implementation

In this implementation, the middleware starts the process of analyzing once it has all the reader results. This is

interesting regarding the precision of the diagnosis but it takes much time (indeed the middleware waits until all tags are read by all readers in the path of these tags). This time is in the range of seconds or even minutes while the execution time of the diagnosis process when the reader results are available is in the order of milliseconds. For this reason, we propose two alternative implementations that are much faster.

3.2 Alternative implementations

3.2.1 "Calculation done five by five"

In this implementation, the middleware starts the diagnosis process every time it has the results of 5 readers. When the five readers are analyzed by the middleware, it moves to the five next readers, and so on. We have fixed the size of the reader group under analysis to five, because, with five readers, the middleware has enough data to perform mostly a correct diagnosis and does not take much time such as in the first implementation.

3.2.2 "Moving window"

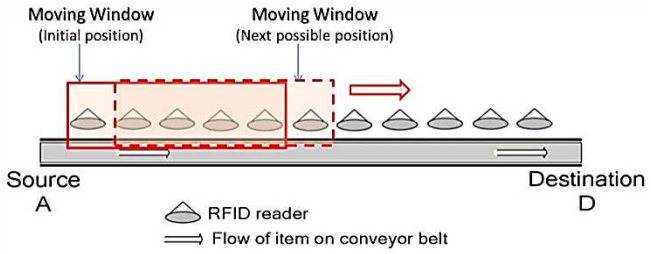

Figure 13. Moving Window approach

In this implementation, the middleware analyzes 5 readers at a time as in the second implementation. The difference between these implementations lies in the selection of the five readers to be analyzed. Indeed, in this approach, the progress speed of the Moving Window varies according to the analyzed readers at each step. When the readers are analyzed by the middleware, the faulty ones are ejected from the window, and the same number of readers is added (from the remaining reader in the path) to the moving window. If all readers of the window are fault-free, only the first reader in the window is ejected as shown in Figure 13.

Table 2. Sample test case

	T_0	T_1	T_2	T_3
R_0	1	1	0	1
R_1	1	1	0	0
R_2	1	1	0	0
R_3	1	0	1	0
R_4	1	0	1	0
R_5	1	0	1	0
R_6	1	0	1	0
R_7	1	0	1	0
R_8	1	0	1	0
R_9	1	1	1	0

```
  Moving window:        Calc. at the end:     Calc. 5 by 5:
   | R0:  0,9993592773  | R0:  0,9846765334  | R0:  0,9993592773
   | R1:  1,0000000000  | R1:  0,9846765334  | R1:  OK
   | R2:  1,0000000000  | R2:  0,9846765334  | R2:  OK
   | R3:  0,9999199097  | R3:  OK            | R3:  0,9999199097
   | R4:  0,9999199097  | R4:  OK            | R4:  0,9999199097
   | R5:  OK            | R5:  OK            | R5:  OK
   | R6:  OK            | R6:  OK            | R6:  OK
   | R7:  OK            | R7:  OK            | R7:  OK
   | R8:  OK            | R8:  OK            | R8:  OK
   | R9:  1,0000000000  | R9:  0,9846765334  | R9:  1,0000000000
Time needed (all methods): 0.088745357 seconds.
Time needed by:
   Moving window method:      0.001501587 seconds.
   Calc at the end method:    0.086095859 seconds.
   Calc 5 by 5 method:        8.0541E-4 seconds.
-------------------------------------------------------------
```

Figure 14. Sample diagnosis result

To evaluate these alternative implementations, we have compared their accuracies and their detection rates with those of the original implementation (Calculation done at the end implementation) in three different scenarios. The first one contains more fault-free readers than faulty ones. The second scenario contains the same number of faulty readers and fault-free ones. The last scenario contains more faulty readers than fault-free ones. Each scenario has 44 test cases for a total of 923 readers to diagnose. The number of the participating readers in each test case varies from 3 to 60 readers and the number of tags varies from 2 to 6 tags.

Table 2 shows a test case that is implemented as a matrix (readers, tags) such as the one of the results comparison step (Section 2.1.2). Each line of the test matrix represents the results of one reader. When a reader exceeds its predetermined performance limit ($R_i(T_j) = 0$) while the other ones don't, it is marked as temporary faulty according to the analyzed tag. At the end of the comparison process, a reader is considered faulty if it exceeds the limit of the performance parameter at least one time. The process of verifying if a reader exceeds or not the performance parameter limit is performed by another program that is not presented here.

Figure 14 shows the diagnosis results of the previous test case by all three implementations on a three-column table and the processing time of each approach. This table contains the failure probabilities of the faulty readers.

The evaluation of the alternative implementations with the aforesaid scenarios identifies the same trends. So, we show only the result of one scenario. This scenario consists of 44 test matrices totaling 665 fault-free readers and 258 faulty ones. The evaluation of the alternative implementations is summarized in Figure 15 and Figure 16.

The terminology used in these figures is the following:

- *Correct Negative*: Fault-free reader is considered as such.
- *Correct Positive*: Faulty reader is detected.
- *False Negative*: Faulty reader is not detected.
- *False Positive*: Fault-free reader is wrongly considered as faulty one.

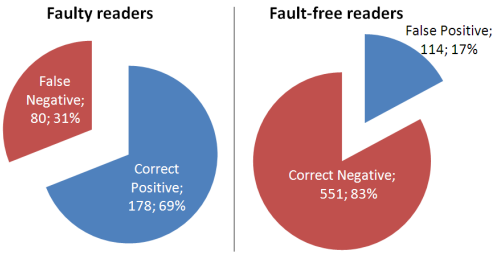

Figure 15. Moving Window Diagnosis

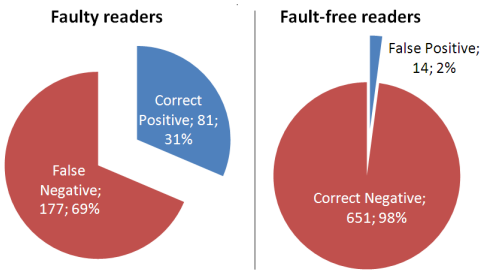

Figure 16. Calculation Done 5 by 5 Diagnosis

The "Moving Window" approach detects more failures than the "Calculation Done 5 by 5" approach regardless the execution environment. But it also generates much more false alarms (*i.e.*, false positives). In addition, the "Moving Window" approach is faster than the "Calculation Done Five by Five" approach because after the first position of the window, the middleware has just to wait for the results of usually one next reader to include it in the window rather than five next readers for the other implementation. The only cases where the "Moving Window" approach is inefficient are when there are at least three faulty readers in the same window at the beginning of the diagnosis process. This will distort all the diagnosis. Indeed, the faulty readers constitute the majority. So, the real fault-free readers will be considered faulty at each time and ejected from the window.

So, the choice between these implementations depends on the dependability policy adopted by the end-user. If he wants to detect the RFID component failures as soon as possible, with a maximum failure detection rate, the Window Moving implementation is the best. If he wants a diagnosis mechanism that gives fewer false alarms and that works in a reasonable time, the "Calculation done 5 by 5" approach is better. Finally, if he wants to have the most reliable diagnosis even if it takes much more time, then "Calculation done at the end" approach is the best.

4. CONCLUSION

We have presented in this paper a probabilistic diagnosis approach based on a statistical analysis to monitor the readers and tags in an RFID system. It consists of three steps. The first one is the calculation of the neighbors of each reader according to the analyzed tags; *i.e.*, the neighbors of a reader for a given tags will not be necessary the same for the following groups. The second step is the comparison of all results of all readers, and then in the third step, once a faulty reader or tag is found, the proposed approach will associate to this result a failure probability according to the running conditions. After that, we have introduced a state verification mechanism to refine the diagnosis. Finally, we have presented three different implementations of the probabilistic diagnosis, and we have shown which one is better according to the running environment and the objectives of the end user.

In future work, we will design an RFID middleware that will accommodate both diagnosis mechanisms to evaluate them in real world deployments with fault injection (environmental disturbances).

REFERENCES

[1] D. J. Glasser, K. W. Goodman and N. G. Einspruch, "Chips, tags and scanners: Ethical challenges for radio frequency identification," *Ethics and Information Technology,* vol. 9, no. 2, pp. 101-109, 2007.

[2] P. Sanghera, F. Thornton, B. Haines, F. Kung Man Fung, J. Kleinschmidt, A. M. Das, H. Bhargava and A. Campbell, How to Cheat at Deploying and Securing RFID, Massachusetts, USA: Syngress Publishing, Inc., Elsevier, Inc., 2007.

[3] G. Fritz, V. Beroulle, O.-E.-K. Aktouf, M. D. Nguyen and D. Hély, "RFID System On-line Testing Based on the Evaluation of the Tags Read-Error-Rate," *Journal of Electronic Testing, SpringerLink,* June 2011.

[4] C. Floerkemeier and M. Lampe, "Issues with RFID Usage in Ubiquitous Computing Applications," *Lecture notes in computer science - Pervasive computing,* vol. 3001, p. 188–193, 2004.

[5] S. R. Jeffery, M. Garofalakis and M. J. Franklin, "Adaptive Cleaning for RFID Data Streams," *Proceedings of the 32nd international conference on Very large data bases,* pp. 163-174, September 2006.

[6] G. Fritz, B. Maaloul, V. Beroulle, O.-E.-K. Aktouf and D. Hély, "Read rate profile monitoring for defect detection in RFID Systems," in *RFID-Technologies and Applications (RFID-TA), IEEE*, 2011.

[7] S. Rangarajan and D. Fussell, "A Probabilistic Method for Fault Diagnosis of Multiprocessor Systems," *Eighteenth International Symposium on Fault-Tolerant Computing,* pp. 278-283, 1988.

[8] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines – A Survey," *Proceedings of the IEEE,* vol. 84, no. 8, pp. 1090-1123, August 1996.

[9] D. A. Westott, D. D. Coleman, P. Mackenzie and B. Miller, CWAP - Certified Wireless Analysis Professional Official Study Guide: Exam PW0-270, Indianapolis, Indiana: Wiley publishing Inc., 2011.