DHT-based distributed ALE engine in RFID Middleware

Loïc Schmidt, Nathalie Mitton, David Simplot-Ryl, Roudy Dagher, Roberto Quilez

DHT-based distributed ALE engine in RFID Middleware

Loïc Schmidt, Nathalie Mitton, David Simplot-Ryl, Roudy Dagher, Roberto Quilez
INRIA Lille - Nord Europe, CNRS, University of Lille 1 - FRANCE
Email: {firstname.surname}@inria.fr

***Abstract*—Following the "Internet of Things" concept, each object is associated with a unique identifier which will allow to retrieve information about it in large databases.**

In the process of managing a large amount of objects, and consequently a large amount of events from readers, without overloading the network, these events have to be filtered and aggregated. This is the aim of the Application Level Events (ALE) standard from EPCGlobal, which receives events from readers and sends a useful and well constructed report to the business application. The ALE may be connected to several hundreds of readers. As the number of readers may increase with the increase of the company, a bottleneck may appear with all readers events sent to the ALE. A solution for scalability is to distribute the ALE.

In this paper, we propose an efficient way to solve this problem based on a Distributed Hash table (DHT). One role of the ALE is to insulate business application from technical concern so in our solution, we present a mechanism to distribute the ALE using Chord, a well-known peer-to-peer lookup system, and being transparent for business applications. This solution is compliant with the EPCglobal existing standard, scalable, robust and transparent for other layers of the middleware. We show that our solution generates only 10% overhead than in a nominal case while offering a better robustness and scalability when numbers of tags and readers increase significantly.

I. Introduction

The "Internet of Things" aims at creating a large wireless network in which all objects would have a unique identifier. This concept is attributed to the MIT Auto-ID Center, founded in 1999 [1].

This idea goes along with Radio Frequency IDentification technology (RFID). An RFID tag can be placed on all objects, offering a way to question them and know their identity. Using this ID, together with an efficient object name service (ONS) [10] and shared databases, we retrieve information at anytime. Internet of things standards can be found under the name of EPCglobal Network [2]. The Auto-ID Center is now known under the name of Auto-ID Labs. Once each item has its unique ID, called Electronic Product Code (EPC) in the EPCGlobal network, several operations can be performed (*i.e.* traceability, inventory, *etc.*).

The development of these standards provides a middleware offering a unified way to connect business applications to it and to perform the operations abovementioned regardless of the technical concern (*i.e.* readers management, filtering and aggregation of reader events, *etc.*). In the case of an inventory operation, the middleware has to deal with several reader events to filter and aggregate in a single report for the business application that needs this inventory. One key aspect is to be compliant with these existing EPCglobal standards [5], [8] because they are used in various companies all around the world. As EPCglobal is also a part of GS1 [3], which has standardized bar-codes such as EAN/UPC, these standards are already widespread in Europe and North America.

In this paper, our concern is a company with several warehouses equipped with RFID readers. The current EPCGlobal ALE standard offers to business applications a way to make a stocklist of all products of all its warehouses by sending a specification to each ALE in each warehouse. All ALE engines will then send a report and the business application can eventually draw its inventory. The business application thus still needs to perform filtering and aggregation over these multiple reports. If the number of readers and/or the number of ALE is too important, the application will eventually saturate and crash under the load.

Note that one may think that this problem does not hold in an EPCGlobal network because companies hold EPC-IS [6] and query this latter one rather than directly the readers. However, the EPC-IS can thus be seen as a business application and thus could be saturated depending of the company needs. In addition, EPC-IS is not mandatory.

In this paper, we propose a DHT-based distributed ALE based on distributed hash tables and peer-to-peer (P2P) mechanisms to prevent from this bottleneck. In our system, any ALE is translated into a node of the P2P system. When receiving specifications, it has to split and distribute it to other involved ALEs and merge all reports locally before sending the final report to the business application. Our proposition provides a way to distribute the ALE together with the rules to split and merge specifications and reports. To the best of our knowledge, it is the first one to provide the following features:

- **standard compliant:** every ALE procedure described in this paper matches the current ALE standards [8] and recommendations from EPC Global,
- **scalable:** the system supports a higher load than the traditional EPC Global ALE system as experimentations show,
- **transparent:** for other layers of the middleware: business applications send EPCSpec and receive EPCReport exactly as when there is only one ALE without performing any additional action and without being aware of the number of ALE engines.

Through analysis and experimentations, we will show the rise

of the bottleneck in a classic architecture and how our solution solves this problem.

This paper is organized as follows. In Section II we give an overview of an RFID middleware and some EPC Global standards. We also present our motivation, we define the problem we are interested in and we review different solutions found in the literature on the distribution of the ALE. Section III presents our contribution before showing some results in Section IV-A. We show the limitation of one ALE and compare it to our solution. We finally conclude in Section V.

II. Context

A. EPCGlobal Network

An Electronic Product Code (EPC) is an identifier used in the EPCglobal architecture [7]. Defined by several partners, such as the Auto-ID Labs, GS1, ETH Zurich, *etc.*, it provides a way to uniquely identify items and is based on the GS1 identification system used in EAN/UPC bar codes [3].

In addition of the Tag Data Standard [9] and Tag Data Translation [9][19], which define the structure of an EPC, EPCGlobal has developed several standards in order to provide a general network architecture for retrieving ID from EPC tags, managing, storing and sharing events along the process. The two main components of the middleware are the Reader Protocol (RP) [5] and the Application Level Events (ALE) [8] (Figure 1). The former provides an abstraction layer insulating the reader hardware specifications to upper layers. The latter filters and aggregates events received from readers (via RP) in order to send only one report to business application containing only needed information which parameters have been previously defined. By doing so, the ALE prevents the overload of the network and of applications.

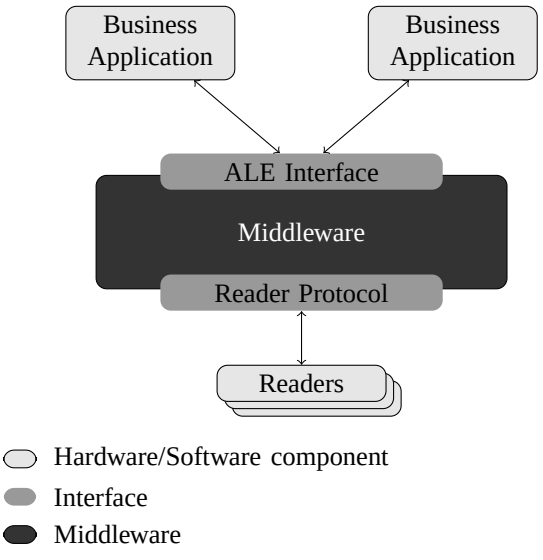


Fig. 1. RFID middleware

1) EPC ALE ECSpecs and ECReports: In order to use the middleware (and furthermore the connected readers), an application has to send a specification (ECSpecs) to the ALE engine that explains what kind of operation the application needs to perform. This specification contains information such as the kind of EPC to report, when to start and stop the capture, where to send the report, *etc.*Figure 2 is an XML file describing an ECSpecs.

```
1. <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2. <ns2:ECSpec xmlns:ns2="urn:epcglobal:ale:xsd:1">
3.   <logicalReaders>
4.     <logicalReader>LogicalReader1</logicalReader>
5.     <logicalReader>LogicalReader2</logicalReader>
6.   </logicalReaders>
7.   <boundarySpec>
8.     <repeatPeriod unit="MS">10000</repeatPeriod>
9.     <duration unit="MS">9500</duration>
10.    <stableSetInterval unit="MS">0</stableSetInterval>
11.  </boundarySpec>
12.  <reportSpecs>
13.    <reportSpec>
14.      <reportSet set="CURRENT"/>
15.      <output includeTag="true"/>
16.    </reportSpec>
17.  </reportSpecs>
18. </ns2:ECSpec>
```

Fig. 2. ECSpecs file

This file defines what the application requires from ALE to report. First, logical readers to be involved in the process have to declared (Lines 3-6). In our example, the application needs report on tags read by logical readers 1 and 2. Then, the boundarySpec section (Lines 7-11) defines when to start and stop the reading. Here, application needs readers to be activated every 10s for $9,5$s. AT last, the reportSpecs section (Lines 12-17) states what to report on tags. In our example, the application asks for the list of tags read but it could also be the number of tags, the list of tags removed or added since last reading, the list of tags which ID matches a given pattern, a combination of all these variants, etc.

Once the specification is sent to the ALE and validated, the middleware performs the operation and then sends the report (ECReport) to the application. Figure 3 is an XML file describing an ECReport that may be reported for the above-mentioned ECSpecs. The report indicates the specifications it answers (Line 3) and gives the list of tags read. In our example, two tags have been detected in the fields of involved logical readers (Lines 12-14 and 17-19): urn:epc:tag:sgtin-96:1.211298.0070875.0 and urn:epc:tag:sgtin-96:1.211298.0070875.1.

2) EPC ALE reader management: In the ALE, readers are configured as logical readers via LRSpecs files. Figure II-A2 shows the definition of a logical reader. A logical reader can be composite or not. If the reader is not composite (the logical reader is the physical reader itself), it defines one physical reader with informations needed to contact it: name, connector (LLRP, RP, or proprietary adapter), IP address and port (in case of network reader), *etc.* If a logical reader is composite, it is defined as a set of other logical readers (composite or not).

```
 1. <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 2. <ns2:ECReports totalMilliseconds="32024"
 3.                specName="specCURRENT"
 4.                date="2010-05-28T11:22:10.721+02:00"
 5.                ALEID="ETHZ-ALE630889259"
 6.                xmlns:ns2="urn:epcglobal:ale:xsd:1">
 7.   <reports>
 8.     <report>
 9.       <group>
10.         <groupList>
11.           <member>
12.             <tag>
13.               urn:epc:tag:sgtin-96:1.211298.0070875.0
14.             </tag>
15.           </member>
16.           <member>
17.             <tag>
18.               urn:epc:tag:sgtin-96:1.211298.0070875.1
19.             </tag>
20.           </member>
21.         </groupList>
22.       </group>
23.     </report>
24.   </reports>
25. </ns2:ECReports>
```

Fig. 3. ECReports file

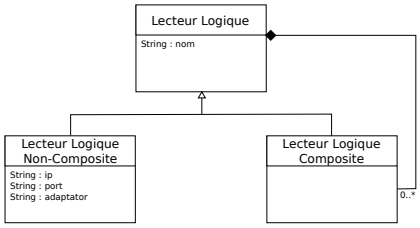


Fig. 4. Logical reader

3) EPCGlobal architecture: A typical architecture of an EPCGlobal network is shown in Figure 5. A business application is connected to the ALE server. Some physical readers are connected to this server and configured as logical readers. Tags are read by readers and events are sent to the ALE server. The server filters and aggregates events from readers according to specifications received from the business application, and finally sends reports to this latter one. With the ECSpecs file above-mentioned, only LogicalReader1 and LogicalReader2 events (*i.e.* tags read) will be used to build the report.

B. Problem statement and related work

A problem arises with the typical architecture shown in Figure 5 when too many tags are read or too many readers are connected to a single ALE. In such cases, a bottleneck appears between readers and the ALE. Indeed, in our typical architecture, if a great amount of readers are connected and each reader detects a huge set of tags simultaneously, the network may not be able to manage so many readers events and the ALE may not be able to process so much data. So one ALE for managing all company warehouses readers is not scalable enough. A solution would be to duplicate ALE engines.

Figure 6(a) presents the same case as above but with two ALE engines. By using a load balancing approach [17], this solves the problem of overload of work for ALEs. This solution offers mechanisms to know how loaded are the different ALEs and to migrate ECSpecs from an over-loaded one to some under-loaded other. It provides also a way to migrate readers concerned by the newly migrated ECSpec in order to allow the under-loaded ALE to perform the operation described in the above-mentioned ECSpec.

But solving the problem of overload of work for ALEs arises three new problems: *(i)* conflicts: if an ALE is over-loaded by several ECSpecs that all use a same reader, EC-Specs are not movable, and this ALE remains overloaded; *(ii)* network overload: if the overloaded ALE and its readers are located in Japan and the under-loaded ALE in Europe, after the migration of one ECSpecs from Japan (and readers reconnection), all reader events will have to cross the world, which may not be a scalable and economic solution in term of bandwidth occupancy, energy saving and latency; *(iii)* transparency for business application: if the business application goal is to perform an inventory of all items, *1)* it has to send a specification to each ALE involved in the inventory process, and *2)* it receives one report per ALE that may have redundant data and *3)* needs to filter and aggregate once again (*e.g.* tag 5 in Figure 6(a) may be read by both Reader 3 and Reader 4 if they are close), what is not the role of the business application. This only puts off the bottleneck problem between the ALE and the business application. Indeed, this architecture does not offer scalability when the number of ALEs increases consequently.

In order to solve the problem of transparency, Liu *et al.* [15] propose a Global ALE that splits ECSpecs before sending them to right Sub-ALEs and merges ECReports received from above-mentioned Sub-ALEs. A Connection Pool manages the data received from the reader layer (*i.e.* the EPC Pool) and the Sub-ALEs pool (with CPU usage, IP address, *etc.*). This is the component that scheduling procedures.

By doing so, they provide a solution to the ALE overloading problem (*i.e.* by reducing the number of readers managed by the ALE) and the transparency to higher levels of the network (*i.e.* business applications). But the system is centralized like the typical architecture (Figure 5) and a bottleneck may appear between Sub-ALEs and the Global ALE.

The ALE overloading problem is beyond the scope of this paper. Our proposition is to **distribute** ALEs and to give them mechanisms to process **specifications** and **reports** (split ECSpecs and send part to concerned ALEs, receive and merge reports) providing **transparency** to business application. Each ALE will be distributed like in peer-to-peer system using a dynamic hash table (DHT). Here, it focuses on scalability, transparency for upper layers and network load, providing a solution for above-mentioned problems, and compatibility with existing EPC ALE standard, providing reusability for already developed or future business applications.

III. DHT-BASED DISTRIBUTED ALE

A. Preliminaries

Providing a scalable and efficient location service in the context of self-organizing systems is a non–trivial problem,

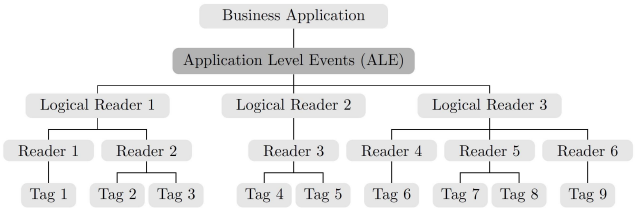
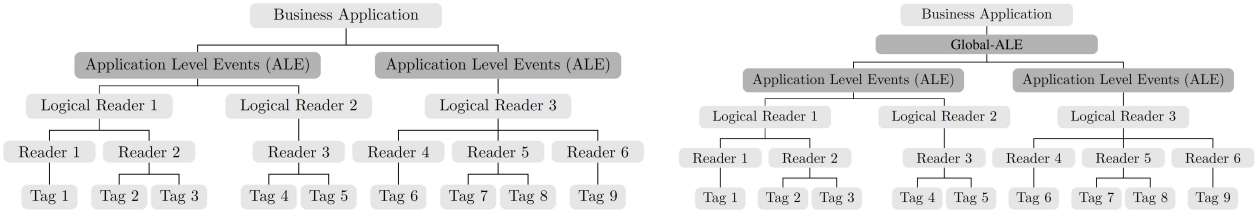


Fig. 5. Typical architecture



(a) General architecture (b) Global ALE

Fig. 6. Two ALEs in two warehouses

due to the spontaneity of networks. This requires a dynamic association between identification and location of a node, and the specification of a mechanism to manage this association. Furthermore, there is the need for minimizing the control message overhead for routing or location discovery. An efficient solution is to perform an *indirect routing* [12], [14], [22].

An indirect routing operation is performed in two steps: *(i)* first locate the target and then *(ii)* communicate with the target. The main difference with classic routing is how the target is located. Instead of using a big routing table containing all information and addresses of targets, in indirect routing, this table is distributed among the nodes and is accessible via a hash function. Distributed Hash Tables (DHT) represent the basis of indirect routing. Basically, they provide a general mapping between any information and a location establishing then a location-independent routing layer. This allows the network to decouple the location of a node from the location itself. With this approach, the information can be totally distributed, which is important for achieving scalability in large scale networks.

In this paper, we use Chord [20] as P2P system. We choose Chord for its simplicity and it answers all our needs, but we can easily switch to another system if needed. [20] uses a circled virtual space. Each node is responsible for a partition of this circled virtual space. The main advantage of Chord is its robustness. Indeed, Chord allows a node to join and leave, reflecting changes to the rest of the network. Moreover, its lookup operation always results in success or definitive failure in predictable time. Each node maintains about $O(\log N)$ and a lookup operation is performed in $O(\log N)$ (with N the number of nodes in the system).

B. ALE P2P system

1) Registration of a non-composite reader: By uniquely assigning the value of the property *PhysicalReaderName*, this value is used as the key for the DHT. The information needed in order to interrogate the ALE connected to the reader is depending on its implementation. In our case, it is the url of the webservice it provides. For the rest of this section, terms *ALE* or *node* are both used to define nodes of the P2P network, which are also ALE. Terms *reader* or *object* are used for non-composite logical readers and the term *record* stands for a pair *(readerName,aleUrl)*.

The P2P system provides several operations in order to add or delete records, retrieve information stored for a key, join or leave the network, and so on. In our case, the distributed ALE is built upon the Chord protocol.

Once the P2P network structure is established for all connected ALEs, let us see how to register a new reader. To better understand, let us have a look on Figure 7. Let Reader 1 be a new reader connected to ALE 4, configured as in a regular ALE 4 (arrow ❶). ALE 4 uses the hash function to know where it has to declare this new reader. The hash function returns ALE 5. Thus, ALE 4 sends the pair *(readerName, aleUrl)* to the contact node ALE 5 (arrow ❷). Once this is done, every node can retrieve the url of ALE 4, managing Reader 1 by performing the same operation on the property *PhysicalReaderName* of Reader 1.

2) Queries mechanisms on a non-composite reader: Once readers are recorded, the distributed ALE system is operational. As explained in Section II, when a business application wants to perform an operation, it first defines specifications of the operation (*i.e.* the ECSpecs) in accordance with EPCglobal standard. In the case of the distributed ALE, if the specifications concern readers located on different readers, they have to be split and send to the different involved ALE. We suppose that the ECSpec used in the one on Figure 2 for illustrating this splitting operation. ECSpec defines two logical readers to be involved in the process (LogicalReader1 and LogicalReader2, Lines 5 and 6).

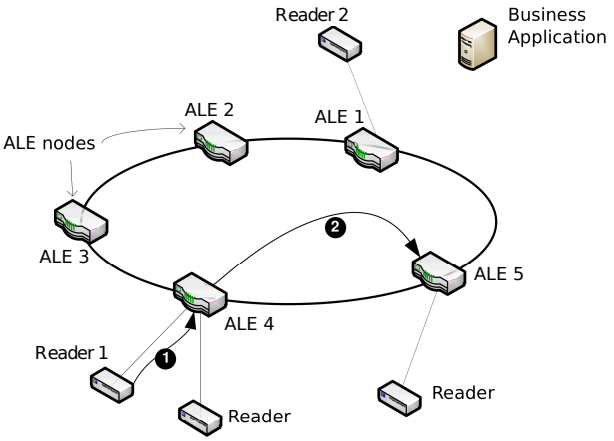


Fig. 7. Recording a new reader in distributed ALE system

Figure 8(a) presents a distributed ALE system with the different steps performed at runtime in order to use the system with the above-mentioned ECSpecs (*i.e.* spread specifications sent by business applications and merge reports). The LogicalReader1 and LogicalReader2 in the ECSpecs are configured as non-composite and respectively parameter Reader 1 (connected to ALE 4) and Reader 2 (connected to ALE 1). When a business application needs to perform the inventory operation on Reader 1 and Reader 2, it first has to send the ECSpecs XML file to an ALE belonging to the network (arrow ❸). It may be the closest one, the less loaded one or whatever, it depends on the implementation and/or configuration choices. In our case, let us suppose that the application connects itself to ALE 1 which thus receives the specification. ALE 1 knows its readers and that Reader 1 is not connected to it, but Reader 2 is. Here comes the hash function, which allows ALE 1 questioning the P2P structure. By hashing the logical reader 1 name $LogicalReader1$, ALE 1 gets the ALE responsible to store information about this EPC (here ALE 5 where Reader 1 has been registered previously), and can query it (arrow ❹). ALE 5 then answers with the url of ALE 4 hosting Reader 1 (arrow ❺). Then ALE 1 splits the ECSpecs files based on different involved readers. In our case, the splitting operation consists in splitting in two ECSpecs files, one for ALE 1 with only $LogicalReader2$ (Figure 9) and one for ALE 4 with only $LogicalReader1$ (Figure 10).

The first ECSPecs (related to Reader 2) is kept and processed by ALE 1 while the second (related to Reader 1) is sent to ALE 4 (arrow ❻) via the webservice url received from the lookup operation. The two involved ALEs can now process locally and independly with their ECSpecs as in a regular scenario. It means configuring local reader, launching the inventory, filtering and aggregating reader events, building the report. Once reports are ready, ALE 1 knows that a second report is needed, and waits for it. ALE 4 sends its reports to ALE 1 (arrow ❼) which can then merge reports. Finally, ALE 1, the contacted node, sends the consolidated report to the business application (arrow ❽).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECSpec xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <logicalReaders>
    <logicalReader>LogicalReader2</logicalReader>
  </logicalReaders>
  <boundarySpec>
    <repeatPeriod unit="MS">10000</repeatPeriod>
    <duration unit="MS">9500</duration>
    <stableSetInterval unit="MS">0</stableSetInterval>
  </boundarySpec>
  <reportSpecs>
    <reportSpec>
      <reportSet set="CURRENT"/>
     <output includeTag="true"/>
    </reportSpec>
  </reportSpecs>
</ns2:ECSpec>
```

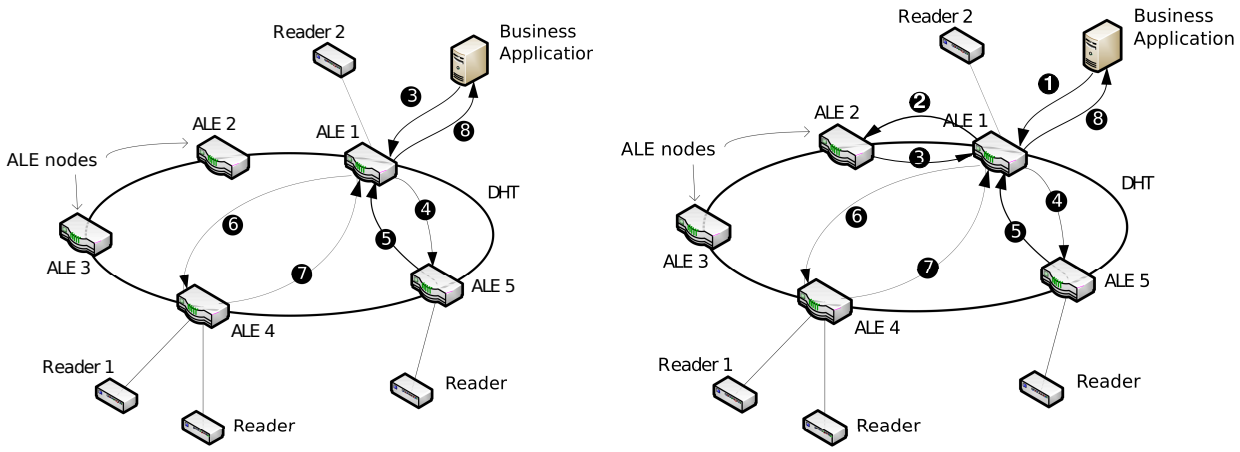
Fig. 9. ECSpecs part for ALE 1

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECSpec xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <logicalReaders>
    <logicalReader>LogicalReader1</logicalReader>
  </logicalReaders>
  <boundarySpec>
    <repeatPeriod unit="MS">10000</repeatPeriod>
    <duration unit="MS">9500</duration>
    <stableSetInterval unit="MS">0</stableSetInterval>
  </boundarySpec>
  <reportSpecs>
    <reportSpec>
      <reportSet set="CURRENT"/>
     <output includeTag="true"/>
    </reportSpec>
  </reportSpecs>
</ns2:ECSpec>
```

Fig. 10. ECSpecs part for ALE 4

This is a simple example that shows the case where multiple distant readers (and so on multiple ALEs) are involved in the ECSpecs. The merge step is important in order to remove double records, manage groups and keeping the transparency towards applications. Indeed, an ECSpecs can specify some grouping patterns. Grouping patterns allow to group identifiers in the report (*e.g.* group by manufacturer, group by item type, *etc.*). Before sending the final report to the business application, ALE 1 has to check and rebuild groups. This is the main characteristic that provides a complete **transparency** for business applications. Applications don't have to split and merge specs and reports themselves. The second characteristic for transparency is the use of the ALE Reading API and ALE Logical Reader API defined in ALE standard as interface between nodes or for upper layer of the middleware. This **EPCglobal standards compliance** provides an easy way to replace existing any EPCglobal ALE by a distributed one without re-writing code of any business application. Finally, building the ALE component of a middleware upon a P2P protocol provides the needed **scalability**, **dynamics** and **robustness**.

3) Distributed Composite Logical Readers: In our DHT, records are pairs *(readerName, aleUrl)*. We can thus retrieve the webservice URL of the ALE connected to the reader just by knowing the name of this reader. In order to solve the distributed composite logical reader (DCLR) problem, we have



(a) Non composite Reader

(b) Composite Reader

Fig. 8. Report generations in distributed ALE system

add a new type of record in our DHT. The key remains the name of the reader, but the value is a list of the reader names that compose the DLCR. By knowing the name of a DCLR, we can thus retrieve a list of reader names and perform the operation as usual. Let's go back to our network described in Figure 7. After the arrival and recording of the reader 1, our DHT contains the four following records:

- Reader 1, URL of the node 4
- Reader 2, URL of the node 1
- Reader 3, URL of the node 4
- Reader 4, URL of the node 5

By adding a new record *(readerName, ReaderNamesList)*, we can define a logical reader called "Distributed Reader 1", gathering reader 1 and 2 for example. Our DHT would look like this:

- Reader 1, URL of the node 4
- Reader 2, URL of the node 1
- Reader 3, URL of the node 4
- Reader 4, URL of the node 5
- Distributed Reader 1, {Reader 1, Reader 2}

Figure 8(b) presents the runtime mechanisms when receiving the ECSpecs of Fig. 11, which only differs from the one from Fig. 2 in the involved reader. This time, we use the DLCR previously defined. When ALE 1 receives the specification from the application (arrow ❶), it first analyses involved logical readers, as usual. This ALE does not know this "Distributed Reader 1" (*i.e.* this reader is not configured locally on this ALE), so it performs a lookup in the DHT. By hashing the reader's name, it can now know the ALE responsible for the record, ALE 2, and can query it (arrow ❷). The answer (arrow ❸) shows that this reader is composed by Reader 1 and Reader 2. We are now in the case describe in the previous section III-B2 with Figure 8(a). ALE 1 knows the reader 2, lookup in the DHT for the reader 1, finds ALE 5, query it (arrow ❹) and retrieves ALE 4 URL (arrow ❺). It splits the specification file, send one part to ALE 4 (arrow ❻), keeps its part, waits for reports, merges them and finally sends the final report to the business application (arrow ❽).

```
 1. <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 2. <ns2:ECSpec xmlns:ns2="urn:epcglobal:ale:xsd:1">
 3.   <logicalReaders>
 4.     <logicalReader>Distributed Reader 1</logicalReader>
 5.   </logicalReaders>
 6.   <boundarySpec>
 7.     <repeatPeriod unit="MS">10000</repeatPeriod>
 8.     <duration unit="MS">9500</duration>
 9.     <stableSetInterval unit="MS">0</stableSetInterval>
10.   </boundarySpec>
11.   <reportSpecs>
12.     <reportSpec>
13.       <reportSet set="CURRENT"/>
14.       <output includeTag="true"/>
15.     </reportSpec>
16.   </reportSpecs>
17. </ns2:ECSpec>
```

Fig. 11. ECSpecs using our distributed composite logical reader

Thus, with a business application developed for using a distributed reader, we just have to modify the record in the DHT in order to include new readers to this distributed one. No more changes are needed on the business application. Transparency is kept.

4) Transparency when readers are added or removed: One main advantage of our solution is the transparency for upper layers of the middleware. Indeed, by using the EPCGlobal standard interface, a business application configured to use readers connected to only one ALE does not need to change its configuration. On the other hand, an application that performs an inventory of all warehouses of a company has a specification file that does not include new readers if added. While new readers are configured on the ALE of this new warehouse, and so in the DHT, this application will send the old specification and performs an inventory of all old readers without taking into account this new warehouse. This application must be changed in order to add the warehouse in the inventory process. Between adding a new reader in an XML file and building the split of ECSpecs and merge of

ECReports in business application, we can easily guess which task is the simplest. But we have lost the transparency. One way to by-pass this is to offer mechanisms for managing distributed logical readers. Thereby, by defining a global logical reader for this application, we just have to add new readers in the definition of this global logical reader. The business application does not need to change anymore, everything is performed in the new warehouse configuration process.

The following section presents some results of comparison between a normal ALE and the distributed ALE.

IV. Evaluation

A. Implementation details

It is worth noting that experimental results will greatly rely on implementation. In our case, starting from the Fosstrak [11] implementation of the ALE, which has for contributors some people from Auto-ID Lab and ETH Zurich, we have plugged it to a java implementation of the Chord protocol called openchord [16]. The structure of one node in our solution is shown in Figure 12.

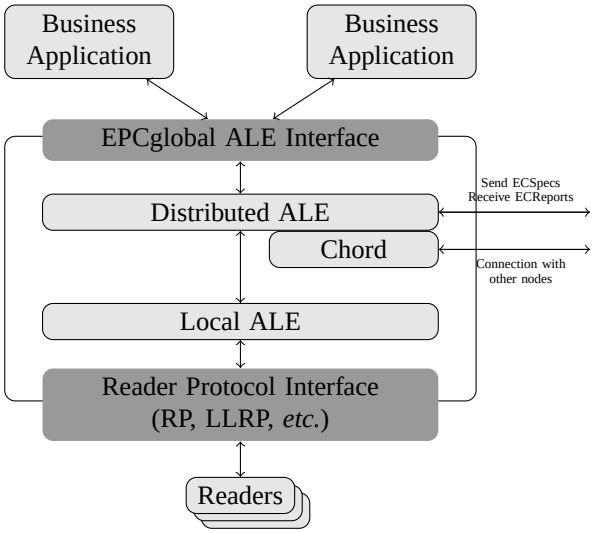


Fig. 12. Software architecture of one distributed ALE node

The distributed ALE component receives specifications from business applications through the standard compliant ALE interface. By using its own readers database and the lookup operation of the Chord component (arrow "Connection with other nodes" in Figure 12), it is able to know which readers are local and which are not. It can now split the specification (as explained in section III-B2) and send them to the local ALE (if local readers are involved) or to distant ALEs through the internal ALE interface (arrow "Send ECSpecs - Receive ECReports" in Figure 12). We chose to implement these latter functions with protocol buffers to minimize the overhead due to the message deserialization (see details in [18]). To the best of our knowledge, we are the first ones to use protocol buffers in such implementations.

B. Results

In order to have enough readers and tags to evaluate our solution, we chose to develop our own LLRP reader simulator rather than using RIFIDI. We made that choice to simplify the test process. Our simulator is lighter than RIFIDI and can be launched in command lines. It takes in input a list of tag identifiers to notify in a setting file, which allows to validate the ECReports content at the end. This simulator allows to ALE to connect to it and simulates RFID tags reading events. For the experiment purpose, we notify the reading of 500 tags every 5000 ms and *Keep-Alive* events are sent every 10000ms in order to keep the connection with the ALE. The used specification asks for a report every 12000 ms of received events for 6000 ms.

By introducing split and merge mechanisms along with threads waiting for reports, we add some work for the ALE component. Before evaluating the solution, we have to remember the aims: a distributed events engine, standard compliant, scalable and which use is transparent for upper layers of the middleware. The standard compliance is verified by the use of the EPCGlobal interfaces and the transparency is guaranteed by split and merge mechanisms of respectively specifications and reports. For the scalability, we have to know if these two mechanisms and the lookup in DHT process are not unreasonably more expensive than the solution of simple duplication of ALE, removing transparency.

The average execution time with the fosstrak ALE is about 4328ms against 4746ms with our solution. When we distribute the four readers among two ALEs, this average comes to 5915ms. Yet, the latency introduced by the additional functionalities is less than 10% of the nominal case by providing scalability and reliability and preventing ALE to crash.

So we fix to 500 the number of tags presents in the field of one reader, and we vary the number of readers connected to one ALE. We can thus know the threshold from which the report received by business application is incomplete. As shown in figure 13 (in red), a single ALE reaches this threshold of 9 readers before sending incomplete reports. This ALE component is strictly the same used in the other test except that it never connects to other nodes or uses the DHT.

We then add a new node in the ALE Chord structure and we vary again the number of connected readers in order to find the new threshold. This threshold is reached for 14 readers (in blue in figure 13).

These results shows that in order to offer transparency for upper layers, the threshold is reached with less readers than in the basic solution. But not only has our solution transparency, but it is also simpler to configure. Indeed, with a simple duplication of ALE, when we configure specifications for business applications, we have to know *(i)* names of involved logical readers, mandatory information even in our solution, but also *(ii)* service URLs of ALE components connected to these involved readers, what is not needed for us.

By lack of time, we only evaluate the cost brought by the transparency. Further testing on scalability, with more readers, more ALEs, more queries would be interesting in order to

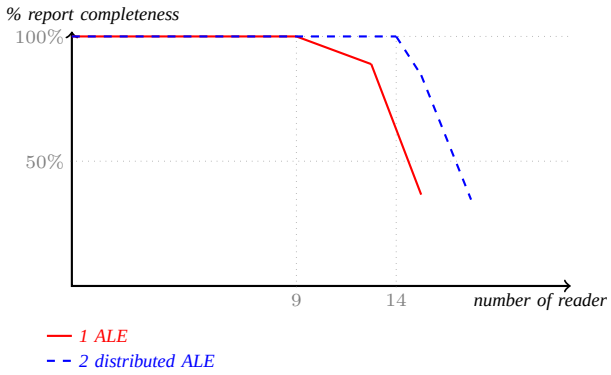


Fig. 13. Threshold of loosing information in the report depending on connected readers number

verify that this cost does not grow exponentially, even if our feeling on this cost is that it will become negligible against the development and configuration complexity of business applications plugged to a middleware without transparency. And as it has been said before, experimental results will greatly rely on implementation. More tests or optimization techniques in the code could help to decrease this cost.

V. Conclusion

In the scope of the Internet of Things, where all objects is carrying an unique identifier, standards have to be defined in order to retrieve informations about objects all over the world. Auto-ID Labs and EPCglobal Inc. define standards for such kind of infrastructures. They define a way to encode ID in RFID, to retrieve data from RFID, to filter and aggregate those IDs into well constructed reports for business applications, *etc*.building an RFID middleware. By doing so, business applications can query this middleware without knowledge about reader protocols, network communications, and so on. A problem rises when too many readers are connected to the middleware. In one hand this overload the work of the ALE, or in the other hand, business applications have to query all ALEs involved.

Using the P2P and DHT principles, we have provided mechanisms that solve the problem of ALEs overload (*e.g.* by distributing readers among them) while keeping transparency for business applications. This distributed application level event engine is EPCglobal standards compliant, scalable, robust and transparent.

Future works will first consist in extending experimentation to the implementations of more than 2 distributed ALE and to a comparison with a global ALE. We will then focus on studying the possibility of the CAN protocol instead of Chord in order to use the benefits of a 2-dimensional virtual space. Linked with the distributed logical readers, and geographical coordinates, it may offer the possibility to define geographical logical readers: querying the big French logical reader will query all physical readers in France, or to automatically define logical reader depending on geographical position or any other information.

Acknowledgments

This work is partly supported by FP7 Aspire European Project [4], the French project ICOM from the PICOM [13] (Pôle de compétitivité des Industries du COMmerce) and the French ANR project WINGS [21] .

References

[1] Auto-id labs. http://www.autoidlabs.org.
[2] Epc global. http://www.epcglobalinc.org.
[3] Gs1, *GS1 General Specifications v10*. www.gs1.org.
[4] FP7 Aspire. Fp7 ict ip project advanced sensors and lightweight programmable middleware for innovative rfid enterprise applications (aspire). http://www.fp7-aspire.com, 2008.
[5] EPCglobal. The Reader Protocol (RP) standard. http://www.gs1.org/gsmp/kc/epcglobal/rp/, 2006.
[6] EPCglobal. Epc information systems. http://www.gs1.org/gsmp/kc/epcglobal/epcis/, 2007.
[7] EPCglobal. Epcglobal architectural framework, version 1.2. http://www.epcglobalinc.org/standards/architecture/architecture_1_2-framework-20070910.pdf, 2007.
[8] EPCglobal. The Application Level Events (ALE) specification. http://www.gs1.org/gsmp/kc/epcglobal/ale/, 2008.
[9] EPCglobal. Epcglobal Tag Data Standard (TDS). http://www.gs1.org/gsmp/kc/epcglobal/tds/, 2008.
[10] EPCglobal. Object naming service (ONS) standard, version 1.0.1. http://www.epcglobalinc.org/standards/ons, 2008.
[11] Fosstrak. Open source rfid software platform. http://www.fosstrak.org/.
[12] J. P. Hubaux, Th. Gross, J. Y. Le Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: the terminodes project. *IEEE Communications Magazine*, 39(1):118–124, January 2001.
[13] FUI ICOM. Infrastructure pour le commerce du futur, 2008.
[14] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of ACM Mobicom*, Boston, MA, August 2000.
[15] F. Liu, Y. Jie, and W. Hu. Distributed ale in rfid middleware. In *Proc. of 4th the International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, pages 1–5, 2008.
[16] open Chord. Open chord. http://open-chord.sourceforge.net/.
[17] J.G. Park, H.S Chae, and E.S. So. A dynamic load balancing approach based on the standard rfid middleware architecture. In *Proc. of the IEEE International Conference on e-Business Engineering (ICEBE '07)*, pages 337–340, Washington, DC, USA, 2007.
[18] Loïc Schmidt, Roudy Dagher, Roberto Quilez, Nathalie Mitton, and David Simplot-Ryl. DHT-based distributed ALE engine in RFID middleware. Research Report 7316, INRIA, 06 2010.
[19] Loïc Schmidt, Nathalie Mitton, and David Simplot-Ryl. Towards unified tag data translation for the internet of things. In *Proc. Wireless Communication Society, Vehicular Technology, Information Theory and Aerospace & Electronics Systems Technology (VITAE'09)*, Aalborg, Denmark, 2009.
[20] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.
[21] ANR VERSO WINGS. Widening interoperability for networking global supply chains. http://www.wings-project.fr/, 2009.
[22] Y. Xue, B. Li, and K. Nahrstedt. A scalable location management scheme in mobile ad-hoc networks. In *Proceedings of IEEE Conference on Local Computer Networks (LCN)*, Tampa, FL, USA, 2001.