RF^{2}ID: A Reliable Middleware Framework for RFID Deployment

Nova Ahmed, Rajnish Kumar, Robert Steven French, Umakishore Ramachandran

College of Computing
Georgia Institute of Technology, Atlanta, GA 30332
{nova, rajnish, robert.steven, rama}@cc.gatech.edu

Abstract

The reliability of RFID systems depends on a number of factors including: RF interference, deployment environment, configuration of the readers, and placement of readers and tags. While RFID technology is improving rapidly, a reliable deployment of this technology is still a significant challenge impeding wide-spread adoption. This paper investigates system software solutions for achieving a highly reliable deployment that mitigates inherent unreliability in RFID technology. We have developed (1) a virtual reader abstraction to improve the potentially error-prone nature of reader generated data (2) a novel path abstraction to capture the logical flow of information among virtual readers. We have designed and implemented an RFID middleware: RF^{2}ID (Reliable Framework for Radio Frequency Identification) to organize and support queries over data streams in an efficient manner. Prototype implementation using both RFID readers and simulated readers using an empirical model of RFID readers show that RF^{2}ID is able to provide high reliability and support path-based object detection.

1. Introduction

RFID technology is rapidly becoming part of today's life through a variety of applications in smart homes, airports, and supply chain management. With the ubiquity of RFID deployments, there exists a growing amount of data being generated from multiple sources that needs to be processed in order to support user queries. However, handling large volumes of sensor generated data streams stresses even high performance computing resources and broadband network bandwidth. It imposes several challenges, especially when these streams are distributed across multiple geographic locations. First, the amount of data from different sources that needs to be organized properly is very large; second, because of the nature of RFID applications, the streamed data and possibly legacy data must be fused together in a time-sensitive manner (e.g. using timestamps); and finally, the overall system has to be able to manage complexities in a scalable manner to answer queries efficiently. These problems become more severe in RFID systems where data is generated from error prone devices like RFID readers [12, 13].

RFID readers use radio waves to communicate with electronic tags which vary in type, capability and size. Passive tags, in particular, are gaining considerable amount of attention as a low cost solution for automating a wide range of applications. However, one of the major drawbacks of the readers using passive tags is their unreliable behavior. Typical readers accurately detect tags 80-90% of the time, whereas readers with improved performance have a detection rate of 95-99%. Unfortunately, this rate is adversely affected by different environmental factors such as the presence of metal objects, objects containing liquids, interference from multiple readers, or the presence of multiple tags within close proximity of one another. For example, the detection rate drops to 70% when a reader attempts to detect more than five tags [11, 12, 13].

A middleware for RFID deployment requires careful design considerations taking into account the inherent vulnerabilities of these devices. We have designed a novel system architecture that addresses different aspects of such a middleware design. In designing this architecture, we had the following goals:

- **Reliability:** Our primary challenge is to provide system level reliability for inherently unreliable components provide a certain Quality of Service (QoS) as required by application queries.
- **Load Balancing:** System load should be balanced across the computational elements when handling a large amount of data.
- **High Throughput:** The system is expected to provide extremely high update rates from the readers and therefore requires high throughput to handle the sheer numbers of items.
- **Scalability:** The system should be capable of handling the addition of new readers or to

1-4244-0910-1/07/$20.00 ©2007 IEEE.

dynamically ignore malfunctioning readers without significantly impacting performance.

- **Data Organization:** For large amounts of data there is a need to properly sort and organize data for efficient query responses.

There have been several recent proposals involving middleware systems for RFID deployment. Savant [9] is designed to handle large amounts of data in a hierarchical manner by constructing a tree of subsystems. Similarly, the architecture defined in High Fan-in System [6] uses a tree like structure for data and query propagation. The problem is handled using a publish-subscribe model in the work of RFIDStack [3]. WinRFID [11] is another architecture that uses web services to ensure data availability. All of these systems aim to provide a scalable solution in distributed environments for filtering, organizing, and providing query results to the end user.

The focus of our work is complementary to these systems. In our work, we focus on the inherent unreliability of RFID systems, and ask whether the reliability can be improved using a middleware system. We use the nature of the dataflow as our ally to improve reliability. Further, the data flow can help in data organization as well, which has been focus of earlier systems. For example, consider a typical RFID deployment scenario in a warehouse. RFID readers are placed at appropriate locations, and they continuously read tags associated with the moving items, e.g. palettes. Often, palettes of a particular item type follow the same physical path from an entry point to the designated destination in the warehouse. Therefore, for many applications, generated tag data can be categorized based on their physical flow.

Most RFID middlewares support two major application categories: *object tracking* and *object location*. Both applications have an internal data flow that follows a *path*. In a typical supply chain scenario, as an example of an object tracking application [20], readers are placed along conveyor belts for item detection. Airport baggage claim systems are another example of object tracking applications [21]. In both these scenarios, readers are statically placed, and the tagged items detected by the readers construct a path. Conversely, in the case of object location applications, tagged items are placed randomly and readers move along a path locating items in their wake. Robots equipped with RFID readers looking for lost objects in an environment is an example of object location. Another scenario is using robots in a disaster situation: tags deployed in the environment may guide the robot in its recovery operation. As should be evident, a path naturally gets created following the sequence of tags acknowledged by readers. We have focused on the object tracking scenario for the work presented in this paper[2]. A middleware that has inherent support for this flow of data considered as a path can be very useful for organizing, filtering and thus responding faster to queries. Using the intuition of data flow, we have designed a path based distributed system architecture for RFID middleware called *RF^2ID (*Reliable Framework for Radio Frequency Identification).

Our proposed scheme consists of (1) a *virtual reader* abstraction to capture the static and potentially error-prone nature of the physical readers and antennas in a scalable manner; and (2) a novel path abstraction, called *Vpath (Virtual Path)* to capture the logical flow of information among the virtual readers as RFID-tagged objects move throughout the environment. Using a notion of path at the system level gives us several advantages. First, the system load can be distributed among multiple virtual readers that constitute a specific virtual path. Second, different QoS attributes can be defined for a path, such as accuracy and priority levels that the virtual readers use to operate on data flowing through the corresponding path. Finally, as there is an internal representation of data based on path attributes, it becomes trivial to support path-related operations on the data, e.g., searching for query results, or making a future projection of data behavior based on history.

The main contributions of this paper are as follows:

- **Study of the unreliable behavior of RFID devices:** A middleware for RFID devices needs to take into account the inherent unreliable nature of these devices. We have done an extensive study of these devices to identify the variety of parameters that affect RFID reader performance.
- **Design of a path based system architecture:** RF^2ID uses Virtual Path (Vpath) to logically distinguish the flow of data streams. Using Vpaths, the system is able to provide higher system reliability, provide a capability to organize the data streams for efficient query management, and offer a vehicle for load balancing among the readers.
- **Implementation of RF^2ID system and its evaluation:** We have developed a prototype implementation of the RF^2ID architecture. The implementation incorporates physical readers as well as simulated physical readers to enable controlled scalability experiments. Our evaluations demonstrate the improved system reliability and item detection capabilities of a path based architecture.

The remainder of this paper is organized in the following way. The system architecture is described in

[2] However, it should be noted that the architecture is general and applies to object location scenario as well

Section 2. Section 3 provides the implementation of the proposed scheme. The system evaluation is presented in Section 4, followed by related work (Section 5) and conclusions (Section 6).

2. System Architecture

Figure 1 shows a schematic diagram of the system architecture. It is motivated by the design goals mentioned in Section 1. While a centralized system may provide reliability, it is not scalable, especially when the deployment involves a large geographical area. Hence the proposed system architecture is a distributed one, and consists of *virtual reader, name server* and *path server* components.

2.1 Virtual Readers

To achieve reliability in a scalable and distributed manner, we have designed a novel abstraction called *virtual reader (VR). Virtual Paths (Vpaths)* are logical channels across a set of VRs. A Vpath is created dynamically taking into account various VR-specific parameters described in later subsections. Each VR is responsible for a set of *physical RFID readers (PR)* within its vicinity.

For applications such as a warehouse wherein the topology does not change very often, the PR to VR mapping takes place during *initialization* (described in next subsection). A VR is responsible for *data management, path management,* and *query management*. VRs use local variables and data structures to make individual local decisions that affect global system behavior. We use the following notations for describing the functions carried out by a VR:

- $\{VR_1, VR_2, .., VR_n\}$ denotes the set of VRs corresponding to a Vpath P
- $\{PR_{(i,1)}, PR_{(i,2)}, .., PR_{(i,m)}\}$ denotes the set of PRs corresponding to a VR_i
- Each VR maintains five lists and a variable for each path P*:*
  - o *observedTagList*: list of tags detected by the associated PRs
  - o *receivedTagList*: list of filtered tags from the PRs eliminating duplicates
  - o *expectedTagList*: list of expected tags based on information from neighbor VR_{i-1} for a given path
  - o *missingTagList*: list of tags expected from VR_{i-1} but not received from the associated PRs
  - o *spuriousTagList*: list of tags received from associated PRs but not expected (based on information from VR_{i-1})

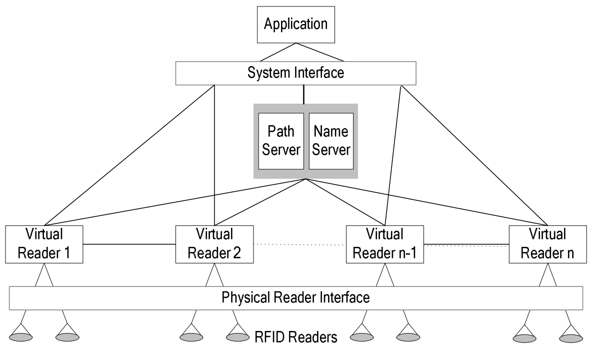


Figure 1: RF^{2}ID architecture with Virtual Readers, Vpaths connecting VRs, Name Server and Path server for available information on Virtual Readers and RFID readers (PR).

- A set of control parameters in a VR to make path creation decisions:
  - o *conn$_{in}$*: maximum number of incoming messages handled per unit time
  - o *conn$_{out}$* : maximum number of outgoing messages handled per unit time.

2.1.1 Initialization

The VR to PR assignment is done during the initialization phase. A PR is inherently unreliable; hence, multiple PRs are associated with a given VR to reduce the probability of errors. A VR is associated with a set of PRs in its geographical region. Figure 2 shows an RFID deployment in a warehouse. The figure shows a 2-dimensional (plan) view of the physical space where the system is deployed. The physical space is divided into regions (dashed rectangles) and a VR is assigned to each geographical region. The tracks show the belts along which items will flow physically from source to destination. Each VR is initially assigned to a disjoint set of PRs that are in the geographical region covered by that VR. The set of PRs used by a VR during system operation is path specific. In Figure 2, VR1 uses upper set of PRs for the path created between A and D; whereas, it uses PRs from the bottom for the path from B to E.

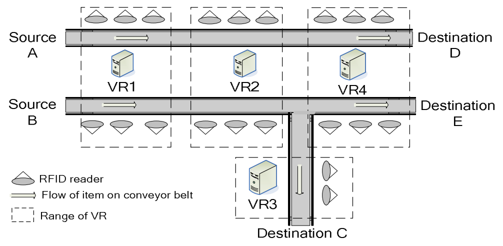


Figure 2: A warehouse scenario showing RFID reader (PR) and VR deployment.

2.1.2 Data Management Data management consists of two major tasks: filtering the data (from the PRs associated with this VR, and from neighboring VRs), and timestamping the data in a consistent manner. We assume that VR_i is able to send its information (*expectedTagList* message) to VR_{i+1} in path P, before the items physically reach the neighbor.

The *data acquisition and processing phase* by VR_i is the process of accumulating scanned data from all PRs$\{PR_{(i,1)}, PR_{(i,2)}, .., PR_{(i,m)}\}$ contained in path P. A corresponding *observedTagList* is generated for each PR in order to compare them with one another. The result of these comparisons is a concrete list of tags (eliminating duplicates) which is recorded in *receivedTagList*, timestamped with the current system time. In the *data comparison phase,* every VR_i (except the source VR), receives a list of expected items from VR_{i-1} along path P and stores it in *expectedTagList.* The timestamps in *expectedTagList* and *receivedTagList* are then compared to guarantee a consistent temporal ordering amongst these VR-generated lists. The items of *expectedTagList* and *receivedTagList* are also compared to create a list of items expected but not acquired in *missingTagList* and received items that are not expected in *spuriousTagList.*.

2.1.3 Path Management Path management includes a *load estimation and path creation phase* and an *overload management* phase.

As shown in Listing 1, the load estimation and path creation phase works as follows. Let us denote the VR receiving the path creation request as VR_{req}, and the requested path as P. VR_{req} first contacts the path server to check if any existing path matches P. If a match is found, the matching path is returned. If a match is not found, VR_{req} contacts the name server for possible set of VRs that can potentially belong to the path P. Each VR_i in path P estimates its current load w_{curr} from the number of existing paths (total_Path) it is currently serving. The load estimation takes into account the current load w_{cpu}; incoming and outgoing messages from VR_i to VR_j; and messages from associated PRs. The numbers of incoming and outgoing messages to VR_i are bound, respectively, to $conn_{in}$ and $conn_{out}$ to restrict the total number of messages handled by a VR in a given time period. After estimating w_{curr}, each VR_i estimates the additional load w_{est} for the new path using history data from previous paths. If the combined workload of w_{curr} and w_{est} does not exceed a predefined threshold w_{tr}, then VR_i sends a *participateNewPath* message to VR_{i+1}; otherwise it sends a *notparticipateNewPath* message to VR_{i+1}. Similarly, VR_{i+1} forwards the received messages and its own participation intent to the next VR along P, and this step is repeated until the destination VR_n is reached. If the total number of participating VRs is above a predefined system threshold VR_{tr}, VR_n sends a *pathCreation* to its neighbor VR_{n-1} which in turn sends it to its previous neighbor, and so on. Otherwise a *pathNotCreated* exception is sent out to the application.

Listing 1: Path creation algorithm

```
total_Path: Total number of paths associated with a given VR
total_PR: Total number of PRs associated with a given VR
total_VR: Total number of VRs in the system
$Message_{in}$ (source): Incoming messages from source
$Message_{out}$ (destination): Outgoing messages to destination
α: Per message CPU cost (%)
$w_{cpu}$: Average per path CPU load on a VR (%)
$w_{curr}$: Current CPU load on a VR (%)
$w_{est}$: Estimated CPU load on a VR due to a new path P (%)
$w_{tr}$: Threshold of maximum permissible CPU load on a VR (%)
$VR_{tr}$: Minimum number of VRs needed for a new path P
$VR\_gCount_{PathID}$: Final count of number of VRs that are part of the
new path
$VR\_localCount_{PathID}$: Local copy of the final count in each
participating VR

//The following steps are executed at $VR_i$, for i=1 to total_VR
//Computing current load on each $VR_i$,
 $w_{curr}$ = for each path j =1 to total_Path in $VR_i$
      SUM($w_{cpu}$ + α*( $Message_{in}$ ($VR_{i-1}$)+ $Message_{out}$ ($VR_{i+1}$))+
      for k=1 to total_PR SUM(α*$Message_{in}$ ($PR_j$));

//Computing estimated load on $VR_i$ due to new path creation
$w_{est}$ = $w_{cpu}$ + α*(estimated $Message_{in}$ ($VR_{i-1}$)+
                    estimated $Message_{out}$ ($VR_{i+1}$ )) ;

if($w_{curr}$ + $w_{est}$ ≤ $w_{tr}$ ) // load is below threshold
   Send participatePathCreation (PathID,VRID) to $VR_{dest}$;
else
   Send notparticipatePathCreation(PathID,VRID) to $VR_{dest}$;
if(VRID == $VR_{dest}$) {    // it is the destination VR
  $VR\_gCount_{PathID}$=
  Total number of participatePathCreation messages received;

  if($VR\_gCount_{PathID}$ ≥ $VR_{tr}$ ){
    Register (PathID, $VR\_gCount_{PathID}$) in Path Server;
    VR_ $localCount_{PathID}$= $VR\_gCount_{PathID}$;
    Send pathCreation (PathID, $VR\_gCount_{PathID}$) to $VR_{i-1}$;
  }
  else
    throw pathNotCreated exception to the Application;
}
else { // it is an intermediate VR
     Receive pathCreation (PathID, $VR\_gCount_{PathID}$) from $VR_{i+1}$;
     $VR\_localCount_{PathID}$= $VR\_gCount_{PathID}$ ;
     Send pathCreation (PathID, $VR\_gCount_{PathID}$) to $VR_{i-1}$;
}
```

The overload management phase works as follows: although each VR makes an assumption of its future load, at any point of time, one or more VRs in a given path may become overloaded. An overload is detected through comparing current system load w_{curr} with the threshold w_{tr}. The overloaded VR_i will update *VR_ globalCount*$_{PathID}$ which contains number of participating VRs and send messages to the other VRs (in the path) to reduce their local value *VR_ localCount*$_{PathID}$ for this path. Both the global and local count of VRs keep total number of participating VRs but the local values in different VRs may be different at any execution point and any VR changing its local value is responsible to update its global value and pass that information along its neighbors. Any VR_i receiving the overloaded message updates its local variable VR_ *localCount*$_{PathID}$ and checks if the updated value for this path is below $VR_{tr.}$ The application is notified about the path overload whenever the number of participating VRs falls below the threshold VR_{tr}.

2.1.4 Query Management. Any VR can respond to queries from an application. Vpaths allow a VR either to answer the query itself or to route the query to a VR that is best able to answer the query. The set of queries include aggregation operations such as:

- Information on all items
- Information on items along a specific path
- Information on items of a specific type or number of items over a period of time

The real power of the path abstraction lies in its ability to handle item location queries and troubleshooting queries such as:

- Locate a missing or misplaced item
- Locate an item last recorded by a specific VR or a specific PR in a VR
- Information about a malfunctioning reader (e.g., a reader that is consistently missing items)
- Information about a specific conveyor belt (e.g., a Vpath that is missing items below a threshold)

It is well known that the time to respond to a real time query has to be very precise [25, 26]. The system is well positioned to handle time specific queries since the readers timestamp the information they gather as well as disseminate. Thus for example, if a VR receives a query that is in its immediate past, it will forward it to the appropriate VR that is ahead of it in the Vpath.

2.2 Name Server and Path Server

The name server is responsible for keeping the mapping between the topology of the warehouse and the VRs assigned to different regions. It knows the physical location of the PRs in the warehouse as well as the physical routes via conveyor belts that exist in the warehouse. In a scenario where the deployment is reasonably stable such as a warehouse, all the information in the name server are defined at initialization[3]. A lookup request to the nameserver for a physical route between a source, destination pair *(Source(x, y), Destination(x, y))*, yields the set of VRs $\{VR_{source}, VR_1, VR_2, \ldots VR_n, VR_{dest}\}$ that are along that route.

The path server carries dynamic information about the paths that exist in the system at any point of time. A path is registered with the path server upon creation by the destination $VR_{dest,}$ and contains the path id, participating VRs in that path, and a lifetime associated with the path. The path entry is deleted when the lifetime expires allowing an automated garbage collection scheme. A new request to use an existing path or a subpath increments the path's lifetime.

3. Implementation

We have implemented a proof of concept prototype of the proposed system architecture. The prototype allows us to perform controlled experiments to quantify the reliability properties of the proposed architecture. It embodies all the same distributed computing elements as described in Section 2, albeit in a stripped down form. The main difference is in the decision making components. The implementation uses a *centralized path controller* which works with a *static route map* instead of the path server and name server, described in the previous section. This structural difference does not affect our study on system performance or reliability. On the other hand, our prototype includes physical readers as well as simulated physical readers. This feature of the implementation allows us to perform controlled experiments far beyond the limited scale of the real hardware at our disposal using the same distributed architecture. In the next few subsections, we describe the features of the prototype system.

3.1 Virtual Readers

In a real deployment, a VR may be mapped to any PC class machine (or even to one of the PRs in a given region). Our implementation is in C, and fully implements the functionalities outlined in Section 2.1. MPI is used for inter-VR communication.

In the VR implementation, the path management and creation mechanisms are considerably simplified to get a prototype system up and running for experimentation. Using a static route map, each VR knows its neighbors for any desired source-destination pair. So, upon receiving a path request, a source VR knows the neighbor to contact to

[3] A dynamic scenario requires a PR discovery phase followed by a PR to VR mapping process. Such dynamic configurations are outside the scope of our paper.

create a path. The path creation request is transmitted in a chain-like fashion along the static route by the VRs until it reaches the destination VR. When the message reaches the destination VR, it broadcasts the successful path creation message to all the VRs. This is clearly inefficient when there is a large number of VRs in a Vpath. Further, this can be a source of bottleneck if a specific destination VR is "hot". Since the primary purpose of the prototype implementation is to study reliability, we decided to use this simple minded path creation and management. However, a full-fledged deployment will use the detailed algorithms in Section 2.1 using parameters such as computational and communication load on each VR.

3.2 Physical Readers

The physical reader used in our implementation is ALR-9800 [22] from Alien Technology. We have done extensive study of this reader with two antennas supporting point to multipoint and multi static architectures, operating at frequency range of 902.75 MHz to 927.25 MHz, and including 50 hopping channels with channel spacing of 500 KHz. We studied tag behaviors using 6 passive RFID tags with reader power set to its maximum level (31.5 dB).

Alien reader provides a rich set of APIs to access a variety of reader and tag parameters. Reader discovery methods such as setDiscoveryListener() and startService() are used for network component discovery within the same subnet. For a large system consisting of thousands of readers, this auto discovery is very efficient. Various reader methods are used to get and set different reader parameters and observed tag information such as getReaderType(), getAcquireMode(), getTagList(), setRFAttenuation(), and setRFLevel().

A PR talks to its associated VR using Unix sockets. The tags are placed by a PR in a queue abstraction implemented using sockets. Thus the PR-VR communication follows a producer-consumer model.

3.3 Simulated Physical Readers

We have implemented a simulated physical reader that is designed to have the same interface and behavior as the Alien readers. Thus a VR treats a simulated PR no different from a true PR. These simulated physical readers offer a powerful and transparent mechanism to study system scalability. Different parameters that affect the reader accuracy such as reader to tag distance, reader to tag angular position, reader power level, etc., are defined as input parameters to the simulated PRs. It also provides provisions to set predefined reader accuracy level. They can be set to be different for different simulated PRs to study a heterogeneous deployment, or all the same for a homogeneous deployment. In the experiments conducted we have considered various reader accuracy levels. A uniform distribution is used to define a set of items and the simulated reader randomly misses items depending on the predefined accuracy. The item detection behavior is random that reflects the PR's physical property studied in section 4.2.2. It differs from other simulators concerned with tuning device level components such as the work in [28]; instead, our focus is on efficient and distributed computations on reader generated data in order to support scalability studies of the system.

3.4 Path Controller and Static Path Map

As we mentioned earlier, the path server and name server are non-existent in our prototype implementation. Instead, the system maintains a *static route map* (in lieu of a name server) that contains for each potential source-destination pairs the list of VRs. Upon a request to create a new path, this static route map is consulted to determine the set of VRs that could be participants for the new Vpath. The VRs initiate the path creation as we detailed in Section 3.1. The VRs that are potential participants may elect to be part of the Vpath or not depending on their current CPU load and memory utilization. Once a path is created, the centralized path controller creates an entry in its table with a unique ID for the new Vpath and the set of participating VRs for that path.

4. Evaluation

Two different aspects of the deployment are studied in this section. First, we conduct a study of the behavior pattern of RFID devices: this study focuses on various parameters that affect the reading of the tags such as reader to tag distance, angular position of tag to reader, number of readers on a palette, and RF attenuation. We present a quantitative study of these aspects of physical readers. Then, we present an evaluation of our prototype system to demonstrate its effect on improving the reliability. The study on RF^{2}ID includes experiments with real deployments of RFID devices as well as simulated readers to expand the scope of our study.

4.1 Reliability Studies of RFID Devices

We have done an extensive study on the ALR-9800 [22] RFID readers. We studied tag behaviors using 6 passive RFID tags with reader power set to its maximum level (31.5 dB). In the experiments, we vary the key factors that affect the number of detected tags by a reader: the reader to tag distance, reader to tag angle and RF attenuation level. Lastly, we study how the tag reading varies over time when the above parameters remain unchanged.

Figure 3(a) shows the impact of reader to tag distance when varying the quantity and proximity of tags from the antenna. For example, using five tags, 3 tags are detected at a distance of 40 inches; whereas for 2 tags all the tags are detected up to 40 inches away. Similarly, Figure 3(b) exhibits how the angle of the antenna affects accuracy while varying the total number of tags. For a fixed reader to tag distance of 15 inches– it is observed that, when the palette is placed $\pm$30° from perpendicular to the antenna, most of the tags are detected; otherwise, the number of tags detected begins to degrade.

Reader attenuation level also has an impact on the number of detected items, which is shown in Figure 4(a). Considering a group of 6 tags placed 15 inches from the reader at an angle of 90°, only 3 items can be detected with a reader attenuation of 9dB. However, if the attenuation is decreased to 0dB, the number of tags identified by the reader is increased to 6. In Figure 4(b) RFID reader behavior is observed over a period of 100 seconds when a palette of six tags is placed 15 inches from the antenna. Our study shows that tags are often missed by the reader, with the specific missing tags varying randomly over time. This behavior demonstrates the inherent unreliability of these readers.

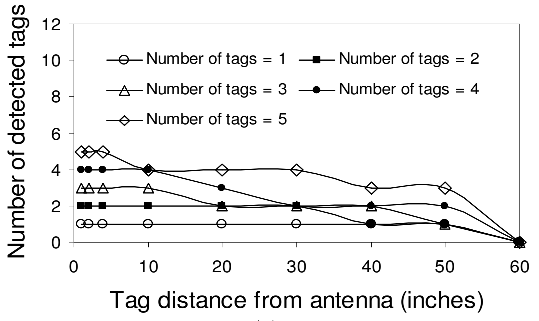
Our experiment with the readers gives insight into their unreliable nature. We have highlighted the inherent unreliability of the readers as well as the influence of environmental conditions on their performance.

4.2 Evaluating RF^{2}ID

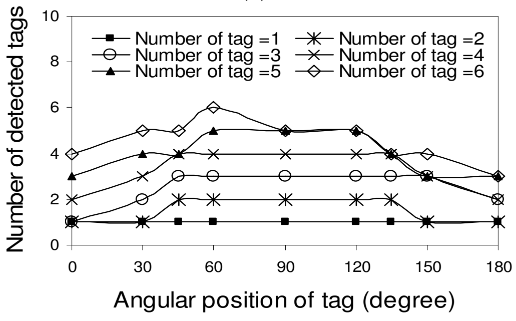
This experiment demonstrates our proposed architecture of RF^{2}ID with a static path topology. The experiment is done on a 100-node Linux cluster with dual Pentium-4 Xeon processors and gigabit Ethernet using MPI for communication among the processors. Each VR is mapped to a processor. The study incorporates the actual physical readers and tags seen in section 4.1.

4.2.1 Experimental Setup. We have conducted two set of experiments to evaluate RF^{2}ID. The first set of experiments evaluate the improved system performance in terms of reliability while the RFID resources are inherently unreliable. Then we explore the strength of the path based system to locate missing or misplaced items along its traversal. The experimental setups are:

- ***Item Tracking:*** Here we have placed two separate antennas 120 inches apart, considering each receiving antennas as an individual reader. The current setting emulates a supply chain scenario with two antennas and a (manually) moving cart carrying the tags. The physical route of the tags brings them within 5 inches of each antenna. We overcome the limited number of RFID tags by reusing the same set of tags with different timestamps.

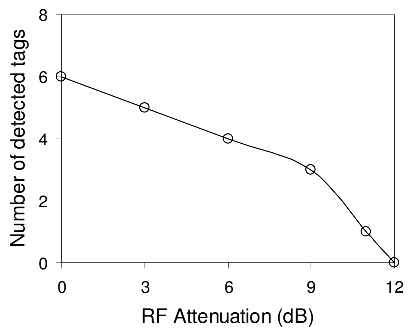


(a)

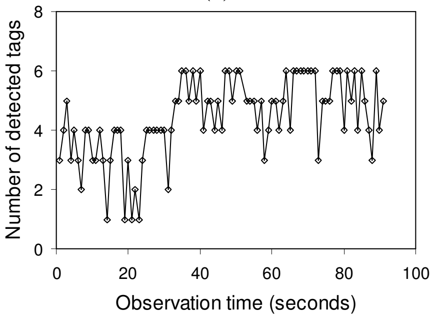


(b)

Figure 3: (a) Varying the number of detected tags with different tag to antenna distance (b) Varying the antenna to tag angle with different number of tags to determine number of detected tags.



(a)



(b)

Figure 4: (a) Number of detected items with varying attenuation of RFID readers (b) RFID reader behavior when palette of 6 tags placed in a fixed distance and detected tags are observed over a period of 3 seconds.

We have used a delay in tag traversal to make sure the system is able to distinguish the difference between a set of tags in consecutive iterations. As RFID readers show reduced performance in the presence of multiple items, we have considered a maximum of six items per palette throughout our experiment. For the scalability study of the system, we have used simulated physical readers discussed in Section 3.3.

- ***Item Location:*** The system initiates a search mechanism to *locate misplaced or missing items*. From the route of the item, the corresponding path is detected in the system. Then the search starts at the destination-VR to the source VRs to figure out the highest indexed VR in the path having the item specific information. When the search narrows down to a particular VR, it looks for information on the item in its associated PRs, and picks the one with the latest information on the item. The VR then controls the antenna power of that particular PR to locate the possible item position within a radius. The mechanism used here is to start the search with the maximum PR power level, if the item is not detected the system assumes the item has been physically misplaced. Otherwise, the VR gradually decreases the PR power level to detect the minimum power level required to detect the item. The system is able to define an approximate radius of the item position using empirical studies of antenna power to item position relationship discussed in section 4.2.2. Further investigation using techniques such as the ones discussed in [23, 24, 10] are part of our future investigation.

4.2.2 Study on Reliability. We investigated improvements in reliability where reliability is defined as the number of detected items in the presence of false negative readings. *False negative* readings indicate a reader is not able to detect an existing item.

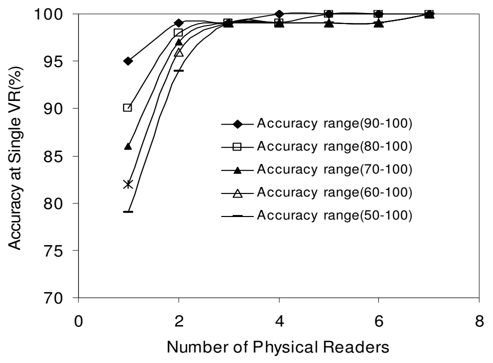
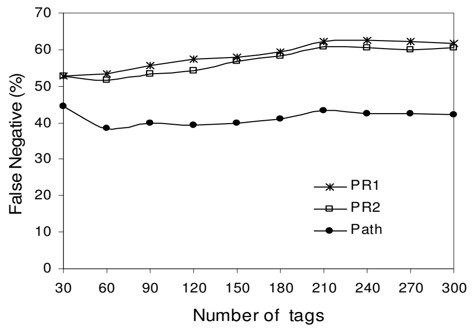


Figure 5: Accuracy as a percentage of found items at a virtual reader level as the number of physical readers is varied.

Figure 5 shows the impact of increasing accuracy level in a single VR on the system reliability. We have simulated multiple PRs attached to a single VR. For example, when the VR consists of four PRs, with accuracy varying from 50% to 100%, the aggregated (VR) accuracy level of the system is obtained as 97%.

Figure 6(a) demonstrates improved performance when two physical RFID readers are used with varying number of tags. Individual PRs show false negative readings from 55% to 60% of the time. But using our notion of Vpath false negatives are reduced significantly. A reading over 120 tags show 52% and 55% false negative readings in PR1 and PR2 where a Path reduces this to 39%. Figure 6(b) specifies the list of tags detected by PR1, PR2 and the combined tag information obtained in a VR consulting these PRs.



(a)

Items Detected by PR1

A02A 0508 11A1 8480 8A01 0101
A02A 0508 11A1 8483 2A01 2001

Items Detected by PR2

A02A 0508 11A1 8482 2A01 0101
A02A 0508 11A1 8480 8A01 0101
A02A 0508 11A1 8483 1A01 1901

Items Detected by VR using PR1 and PR2 in VPath

A02A 0508 11A1 8482 2A01 0101
A02A 0508 11A1 8480 8A01 0101
A02A 0508 11A1 8483 1A01 1901
A02A 0508 11A1 8483 2A01 2001

(b)

Figure 6: (a) False Negative Reading of the System using single PR (PR1, PR2) and Path among PRs. (b) Detected items using two different physical readers PR1, PR2 and VPath.

Due to the limited number of physical RFID readers and tags, we have used simulated physical readers to examine reliability in the presence of larger numbers of VRs. A one to one mapping of PR to VR effectively simulates the unreliability of PRs. In Figure 7(a), it is observed that the increasing number of VRs reduces the number false negatives. For example, 10 VRs with 50%

reader accuracy achieves less than 10% false negatives. Similarly, improved accuracy is observed in Figure 7(b) in terms of identified tags from source to destination. Items can be missed at a particular VR but as we have a notion of Vpath, the information propagates properly through the VRs. A reader with accuracy level as low as 40% is improved to 97% by using 15 VRs.

4.2.3 Missing and Misplaced Item Location. Our Vpath abstraction plays an important role in item location. We have examined the maximum tag detection range of readers in perpendicular position by gradually decreasing antenna attenuation level from 31.5 dB. This study allows us to define a radius around the reader to find possible tag locations.

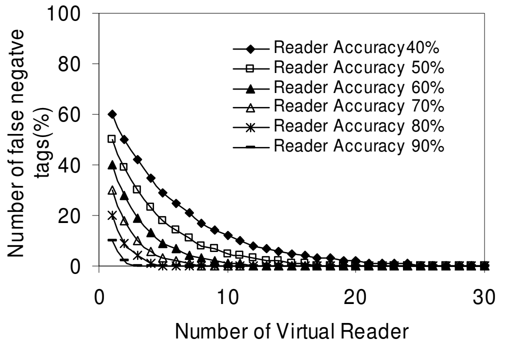
Figure 8 shows the relationship between reader attenuation and maximum item detection range. In an item tracking scenario, a particular item or group of items may be misplaced along its physical route (e.g., conveyor belt). After deciding that an item is lost, the system identifies which PR last observed this item and initiates item location operations. Antenna attenuation can be used to locate, within a certain radius, the last physical location of a lost item. To this end, we have placed a tag at a 28 inches distance from the reader at an angle of 90 degrees relative to the antenna. Here the item is detected by the reader at attenuation set to 7dB indicating a 30 inches radius.

From our study on RF^{2}ID, it is evident that the system reliability is improved by abstractions of Vpath and VR.

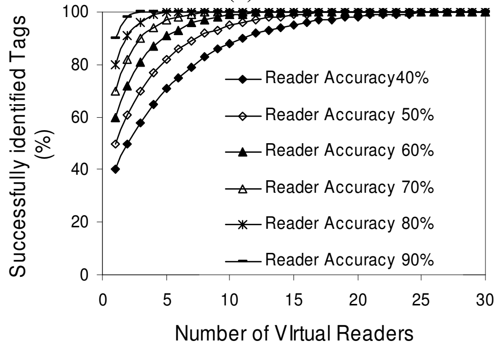
5. Related Works

The *Savant* architecture involves a hierarchy of software components called Savants [9] which are distributed throughout the corporate infrastructure. The RFID middleware design of RFIDStack [3] focuses on reducing data flooding with built-in aggregation and filter types. Full content-based routing is used to deliver captured data only to interested parties, and a feedback mechanism allows data consumers to notify producers of data properties. The architecture of High Fan-in Systems [6] utilizes a tree structure or "bowtie" topology with large numbers of sensors at the edges and a hierarchy of progressively wider-scoped computational nodes. De et al. [10] propose a system for object tracking that builds on the Savant architecture. Similarly, MAX [23] uses a tree-like structure using a static base station with several substations for object location applications. WinRFID [11] uses a hierarchical tree-like architecture that uses web services for deployment of information. SCOUT [24] uses two different approaches depending on an application to ensure the scalability of object tracking for mobile devices. None of these systems exploit the data flow inherent in the movement of items typical of RFID deployments.

The concept of path is used in many different contexts including fault tolerance [14], compiler optimization techniques [15], profiling distributed systems [16, 17], and resource allocation [18, 19]. Scout OS [8] defines a path abstraction to navigate through the layers of the network stack and the Ninja project [7] utilizes a path abstraction as a way to compose multiple services distributed on the Internet into a single logical unit. Our work is inspired by the use of paths in these various contexts.



(a)



(b)

Figure 7: (a) False negative readings by varying number of VRs (b) Tags identified along the paths as a percentage of all tags as the number of VRs is varied.

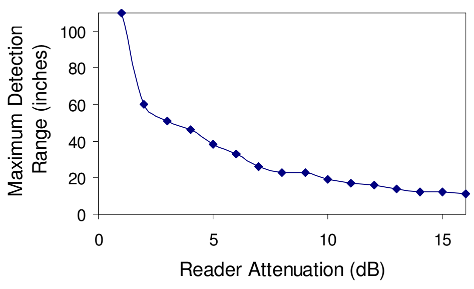


Figure 8: Maximum detection range of a Reader with varying attenuation level.

6. Conclusion and Future Work

We have presented the design and implementation of *RF^{2}ID,* a reliable middleware framework for RFID deployment. We have proposed two novel abstractions: virtual reader and virtual path, and we have presented experimental results to validate the usefulness of these abstractions. We have shown that a path-based architecture improves system accuracy and enables the support of queries over partially located items, i.e., items whose tags are lost at some intermediate location along the physical route from source to destination. Our future work includes incorporating load balancing in path creation and management. We also plan to experimentally compare our framework to other ways of organizing VRs, e.g., using a tree-like hierarchical structure. We plan to develop a complete system using RFID devices and evaluate it extensively with different hardware configurations and physical mappings of RFID deployments.

7. Acknowledgements

The work has been funded in part by an NSF ITR grant CCR-01-21638, NSF NMI grant CCR-03-30639, NSF CPA grant CCR-05-41079, and the Georgia Tech Broadband Institute. The equipment used in the experimental studies is funded in part by an NSF Research Infrastructure award EIA-99-72872, and Intel Corp. We would like to thank Paolo Mentonelli, Echezona Ukah and Vladimir Urazov for their contributions to the RF^{2}ID project. Finally, we would like to thank members of the Embedded Pervasive Lab (EPL), College of Computing, Georgia Tech.

References

[1]. Intermec, *"Why Gen 2 is the RFID for CPG Manufacturers and how to make sure gen 2 performance will meet their needs*".

[2]. ALIEN, *"EPCGlobal Class 1 Gen 2 RFID Specification"*.

[3]. C. Floerkemeier et al., "*RFID Middleware Design - Addressing Application Requirements and RFID*," In Proceedings of sOc-EUSAI 2005.

[4]. M. C. O'Connor, *"RFID users want Clean Data,"* http://www.rfidjournal.com/article/articleview/1232/1/14/

[5]. F. Wang et al., *"Temporal Management of RFID Data*," Proceedings of the 31st international conference on Very large data bases VLDB, 2005.

[6]. M. J. Franklin et al, *"Design considerations for High Fan-in Systems: The HiFi Approach*," Proceedings of the 2nd CIDR Conference, 2005.

[7]. S.D. Gribble et al, *"The Ninja Architecture for Robust Internet-scale Systems and Services,"* Computer Networks 35, 473-497, 2001.

[8]. D. Mosberger et al., *"Making Paths explicit in the Scout Operating System,"* OSDI, 1996.

[9]. Oat Systems and MIT Auto-ID Center, *"The Savant,"* Technical Manual. February, 2002.

[10]. P. De et al., *"An Ubiquitous Architectural Framework and Protocol for Object Tracking Using RFID Tags,"* In Proceedings of the MobiQuitous 2004.

[11]. B. S. Prabhu et al., "*WinRFID – A Middleware for the enablement of Radio Frequency Identification. (RFID) based Applications,*" UCLA – Wireless Internet for the Mobile Enterprise Consortium.

[12]. *"The Basics of RFID Technology,"* http://www.rfidjournal.com/article/articleview/1337/1/129/

[13]. T. Hassan et al.,*"A Taxonomy for RFID,"* Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06), 2006.

[14]. M. Chen et al., "*Path-based Failure and Evolution Management*," In Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI), 2004

[15]. G. Ammons et al., *"Improving data-flow analysis with path profiles,"* In Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation, 1998

[16]. P. Barham et al., *"Magpie: real-time modelling and performance-aware systems,"* In Proceedings of the 9th Workshop on Hot Topics in Operating Systems, 2003.

[17]. T. Gschwind et al., *"WebMon: A Performance Profiler for Web Transactions,"* WECWIS : 171-176, 2002.

[18]. M. Welsh et al., "*SEDA: An Architecture for Well-Conditioned, Scalable Internet Services,*" In Proceedings of the Symposium on Operating Systems Principles, 2001.

[19]. R. J. Mehra et al., "*Virtual Services: A New Abstraction for Server Consolidation,"* In Proceedings of the 2000 USENIX Annual Technical Conference, 2000.

[20]. R. Weinstein, *"RFID: A Technical Overview and its Application to the Enterprise,*" IT Professional, vol 7, 2005.

[21]. M. C. O'Connor, *"McCarran Airport RFID System Takes Off,*" RFID Journal, http://www.rfidjournal.com/article/articleview/1949/1/1.

[22]. *"Alien Technology RFID Readers*", http://www.alientechnology.com/products/alr9800.php

[23]. K. Yap et al., *"MAX: human-centric search of the physical world,"* SenSys, pp. 166-179, 2005

[24]. S. Kumar et al., *"SCalable Objecttracking through Unattended Techniques (SCOUT),"* Technical Report USC CS TR00-738, University of Southern California, 2000.

[25]. Bruckner et al., *"Modeling Temporal Consistency in Data Warehouse*," DEXA Workshop: pp. 901-905, 2001.

[26]. R. Armstrong,*"Seven Steps to Optimizing Data Warehouse Performance*," *Computer*, vol. 34, no. 12, pp. 76-79, 2001.

[27]. L. Ho et al., *"A prototype on RFID and sensor networks for elder healthcare: progress repor,"* ACM SIGCOMM Workshop on Exper. Appro. To Wireless Network Design and Analysis, 2005.

[28]. Y. Han et al., "*System Modeling and Simulation of RFID,*" Auto_ID Labs Research Workshop, September 23rd, 2004.