Real-time Inventory Management with RFIDash

Dept. of CIS - Senior Design 2012-2013

Preetam D'Souza, Max Guo, David Wang, Insup Lee
pdsouza@wharton.upenn.edu, maxmguo@seas.upenn.edu, wada@seas.upenn.edu, lee@cis.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

ABSTRACT

Overstocks, understocks, theft (both internal and external), incorrect manual inventory adjustments, long checkout lines– this is just a sampling of the many challenges facing retail stores today. These symptoms all stem from the current state of retail inventory management (IM) systems. Existing methods involve manual input or barcode scanning for individual items, an error-prone and time consuming method.

RFIDash, our proposed system, makes this process more efficient by leveraging the wireless identification permitted by Radio Frequency Identification (RFID). Although there are prototype RFID systems described in academia, the majority of these approaches are infeasible due to hardware costs and the inability to scale well to a real world store environment. Thus, we aim to develop a practical system that minimizes cost, enhances inventory accuracy, encourages real-time data analysis, and seamlessly scales to any retail environment. With this infrastructure in place, the benefits are clear–retail owners can accurately see and understand, in real-time, changes in their inventory to ensure fully-stocked shelves and happy shoppers.

1. INTRODUCTION

A retail store's ability to translate customer visits to purchases is largely dependent on a well-designed inventory management system. It is critical that a shopper finds the right item at the right time. However, many retailers are currently paying the price of aging IM systems and suffer from overstocks, understocks, shrinkage, incorrect manual inventory adjustments, and long checkout lines.

1.1 Motivation

Current commercial IM systems suffer from two main weaknesses: they are (1) slow and (2) inaccurate. They are slow because they rely on manual cycle counting, an approach typically involving store employees manually counting every single physical item in a store. Since this process frequently takes many hours to complete, stores can only get an accurate picture of their stock levels every few weeks. In a move to optimize this process, stores have recently moved to barcode-based IM systems to mitigate human errors. However, this system still suffers from having to physically scan every individual item in a store. The long lag times between updating physical inventory directly contributes to the inaccuracy of current IM systems. In fact, industry studies show that IM systems only have accuracies of 65-75% on average [15]. The large amount of time spent performing manual inventory counts, coupled with the inherent inaccuracies of this approach, result in a large variance between perceived and actual inventory levels. This gap directly translates into lost selling opportunities, frustrated customers, and disappointing revenues for the affected retailer. Fortunately, RFID offers a host of benefits that can be adapted to build a superior system that is both scalable and accurate.

1.2 Background

Radio Frequency Identification, or RFID, refers to a group of technologies that can be used for automatic, wireless identification of arbitrary objects. Over the years, RFID has improved in both reliability and cost and is now frequently used in contactless smart cards, supply chain management, and other areas of asset tracking [24]. There has recently been an increased interest in RFID tracking systems due to now prevalent global standards for RFID frequency spectrums, communication, and electronic product codes (EPC). The adoption of these standards allow tag schemes to be consistent throughout the world, opening up new doors for RFID on a global scale [2].

The fundamental unit of an RFID system is an RFID tag. A tag consists of a low-cost microchip that can store information in non-volatile memory, along with a radio antenna. Moreover, these tags come in two distinct flavors: active and passive. Active tags have an internal battery that is used to power the underlying circuit and antenna. These tags are capable of broadcasting their data without an external power source and generally have a longer read range. Passive tags, on the other hand, do not have an internal battery and are incapable of broadcasting their data without being activated by an RFID reader. Due to the lack of an onboard battery, these tags usually have shorter ranges but are attractive for their very low cost [31].

In order to store information on a tag, an RFID printer flashes a 96 bit EPC that can uniquely identify an object [33]. This EPC is essentially a more sophisticated Universal Product Code (UPC). These tags are then affixed to objects that need to be tracked. When an RFID reader sends out a radio signal in the vicinity of these tags, the tags respond to the signal by emitting their EPC back to the reader. The end result is a wireless system that can concurrently identify multiple tagged objects over large distances.

1.3 Goal

Current IM systems have failed to answer the demands of modern day retail businesses. A few RFID IM systems have been developed recently in order to fix this problem, but

suffer from being far too expensive [23]. We hope to design a new end-to-end approach for precise and scalable inventory management that leverages the distinct advantages offered by RFID, yet still remains cost-effective.

Our RFIDash system consists of (1) hardware including item-level RFID tags coupled with strategically positioned RFID antennae and readers, and (2) software that analyzes the reader's data to provide a real-time snapshot of a store's current inventory. In order to minimize cost, reader units will be kept to a minimum as these are typically the most expensive components in an RFID system. Antennae will be carefully positioned to maximize read range instead. Additionally, one of the largest costs (that is often overlooked) in existing RFID systems is the price of the software middleware. A large portion of our cost-savings is achieved by building an in-house cloud-based software-as-a-service (SaaS) middleware layer.

A real-time inventory system allows retail stores increased control over their business to respond to changing floor inventory on the fly. Specifically, our proposed system can be used to mitigate over and under stock situations, enhance store security, and increase merchandise visibility, all leading to decreased time and labor costs, increased revenue, and most importantly, happy shoppers.

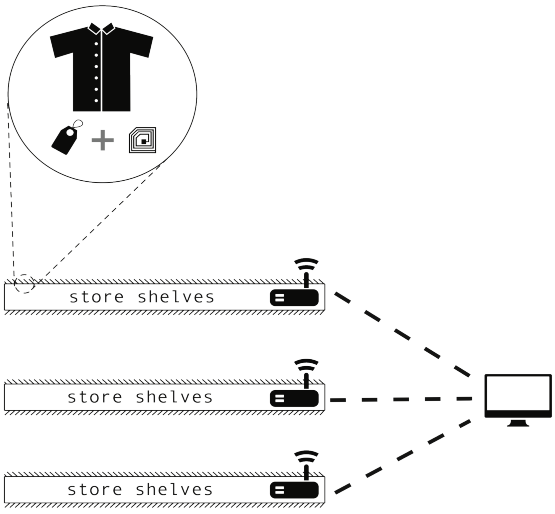

Figure 1: Overview of RFIDash

The overarching vision for RFIDash is to provide an extensible platform for building "smart" shopping services capable of interpreting real-time shopping data. This opens up a wide range of rich applications, such as real-time location based services and targeted advertising, that can pave the road for retail stores of the future.

2. RELATED WORK

RFID has just recently started to be accepted as a reliable technology for asset tracking. It has been used for "smart" transportation passes, theft control in retail stores, toll collection, and even has been embedded in travel documents to identify illegal counterfeiting. The current trend in the RFID industry, however, is increasing interest in RFID systems for supply chain management.

2.1 Industry Initiatives

In fact, multinational retailers like WalMart have declared RFID initiatives that encourage suppliers to tag pallets and cases so they can be tracked from point of manufacture to WalMart's distribution centers [13]. RFID tracking along the supply chain has been embraced by organizations as diverse as the Department of Defense and Airbus for increasing visibility [18]. But all of these applications have mostly been reserved for batch goods tracking from manufacturers to distribution centers.

In the past few years there has been increased interest in RFID solutions at the retail level of the supply chain, including a few pilot programs to implement item-level RFID solutions in retail stores. In 2010 WalMart began the process of tagging specific items like jeans and men's underwear from point of manufacture to the sales floor in order to improve inventory accuracy, estimating that nearly 250 million items could be RFID tagged across the United States [25]. Apparel retailers have displayed the most interest in retail RFID systems, including American Apparel and Falabella. Pilot programs initiated by these two firms have shown dramatic increases in inventory accuracy over barcode-based techniques using mobile handheld readers [17].

2.2 Current Research on RFID Systems

Although there is a lack of academic work on commercially deployable real-time IM systems, there has been some research on location accuracy for RFID tracking. According to the Square grid RFID reader antenna network system, accuracies of up to 0.3 meters can be achieved through a carefully arranged n by n ft grid with n^2 readers. In this approach, different overlapping coverage areas of each RFID reader are used (as shown in Figure 2 and Figure 3) to pinpoint the exact region the RFID signal originated from [14]. This square grid approach has extremely high location accuracy and requires a large number of readers (n^2 to be exact). Since each reader is a few thousand dollars and the average retailer has 5,000-10,000 ft^2 of space, it may require hundreds if not thousands of readers for each store [29]. This approach is simply not cost-effective; by emphasizing location accuracy, it becomes widely impractical in industry.

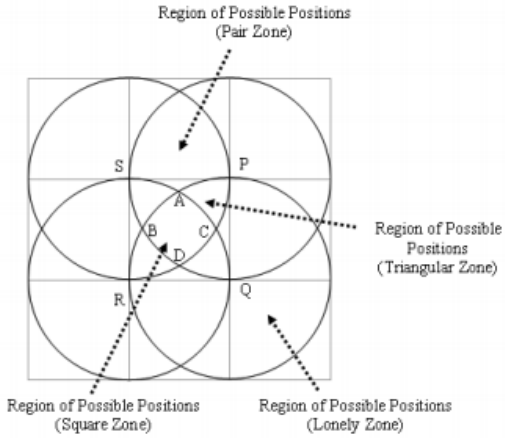

Figure 2: Single Block

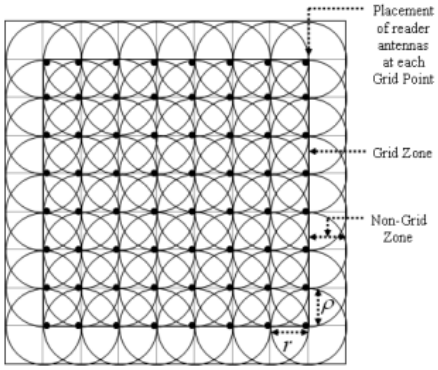

Figure 3: Grid Consisting of Overlapping Blocks

2.3 Existing Data Cleaning Algorithms

There are a few existing algorithms when it comes to cleaning and dealing with RFID data. The purpose of these algorithms is to provide data that is both accurate and easily understood. Common challenges are multiple reads and inconsistent reads.

In Khoussainova et al.'s probabilistic algorithm, they impose integrity constraints on the data. As the data streams in, it is checked against these constraints and then assigned a certain probability of correctness. If the data is determined to be incorrect, the missing data can be filled in based on past data in order to further clean the data [16].

The filtering algorithm proposed by Jeffrey et al. involves overlaying a time frame over the data stream to smooth out the data across a certain amount of time. It implements methods such as binomial sampling and π-estimators to make the reading window as accurate as possible. This way, the RFID data is cleaned before it is interpreted [11].

Masciari also relates the data stream to time in his data cleaning algorithm. However, his approach is different. In this algorithm, the data stream is associated with a time stream that represents the data's structure. The data's correctness is determined by comparing the Fourier transformation of the data's time structure to a correct structure [19].

Xiao et al.'s algorithm combines spatial analysis along with temporal analysis in what is called Complex Event Processing. They impose constraints similar to the probabilistic algorithm and they also filter the stream temporally. The algorithm is more complicated but results in more accurate data [12].

3. SYSTEM MODEL

Our system architecture consists of a three layer technology stack as shown in Figure 4. The first layer consists of the RFID hardware and the network that links the readers together, the second is the middleware, and the third is the actual application layer.

3.1 Hardware Layer

The first layer consists of hardware needed to support an in-store RFID infrastructure. This includes RFID tags, readers, and antennae. This layer is responsible for the actual reading of physical tags located on the sales floor.

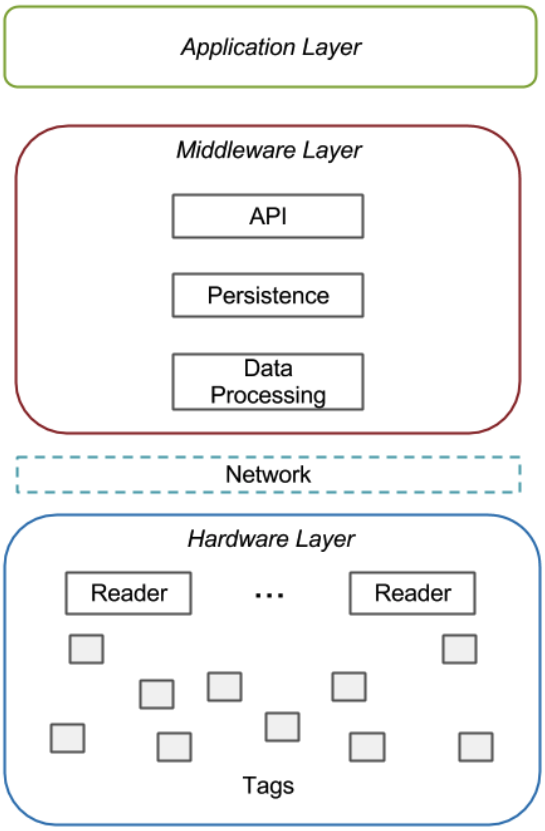

Figure 4: System Model

3.2 Middleware Layer

Just above the hardware lies the first software layer to deal with raw data capture from the grid of RFID reader nodes. Reader data can be notoriously difficult to deal with due to false positive and negative reads, redundant tag captures, and latencies from individual readers over time [5]. Additionally, different reader types may require distinct communication protocols. In order to cope with dirty reads and heterogeneous hardware, a middleware layer was built that will be responsible for processing reader data, filtering it to remove anomalous reads, and persisting clean data to a database for later analysis and querying. In essence, the middleware is responsible for providing a clean abstraction of our RFID hardware to the frontend application layer.

3.3 Application Layer

Finally, the frontend application layer links the backend and hardware infrastructure into a single unified display, allowing the customer to easily visualize any changes at the inventory level and to use other applications employing this real-time infrastructure. As shown in Figure 9 and Figure 11, this application layer will focus on two distinct views: dashboard and checkout mode. The dashboard mode is the inventory part of RFIDash, whereas the checkout mode is a new feature (see Section 6: Application Case Study for

further details).

The dashboard consists of a real-time updating inventory list of all the items along with their price, EPC, count, size, and other tags. As a tagged item of clothing is removed from the racks, the frontend web application will show the quantity changes in real-time, automatically sorted by lowest in stock to highest in stock. With this, a manager can quickly pinpoint exactly how many items are left for each item and size by searching within this inventory view, either by EPC or item name.

Notifications can also be set, allowing users to set "alarms" if inventory levels are depleted below a certain threshold. Currently, this is indicated by a brief yellow flash, indicating the item has crossed over to a different threshold level. This would simulate an on-call schedule in a real store, where certain employees are responsible for restocking the store during different shifts of the day.

3.4 Approach

RFID readers and tags were set up in a very basic configuration within a square grid. Specifically, our experimental setup consists of a single static reader and 4 antennae in a 9m x 8m room. The antennae were set up at the corners of the room in order to maximize the total area covered. Holding the reader configuration constant, focus was placed on writing middleware software to ensure consistent and reliable real-time readings of all tags within the grid. A simple inventory list view was used on the frontend to actually make sense of the read data.

Our evaluation criteria was chosen to align with our goals: to enhance inventory accuracy while keeping cost to a minimum. The chosen accuracy measure is simply the inventory accuracy of RFIDash. This can easily be determined since the ground truth of the true inventory can simply be counted up. This number is then compared to current store inventory system accuracies of 65-75% [15]. We hope to achieve a significant improvement over traditional rates. To measure the cost component of our system, the net cost of tracking a square meter will be calculated.

4. SYSTEM IMPLEMENTATION

Specific implementation details for each of the three layers in RFIDash's software stack (as shown in Figure 5) follow.

4.1 Hardware Layer

Individual, passive EPC Gen 2 RFID tags were selected to be placed on store items. These specific tags were chosen since EPC Gen 2 is a well-supported industry standard with features well suited for our application. These include interrogation operation in the 860 - 960 MHz UHF frequency band, a 96 bit EPC, customizable user memory, and a security mechanism for locking read/write access [21, 28]. These tags will be attached to merchandise just like a traditional price tag.

An Impinj Speedway Revolution R420 UHF RFID reader was chosen for its portable form-factor, power-over-ethernet (PoE) capability, and flexible antenna options (including four antenna ports) [10]. Four Lair/Cushcraft 902-928 MHz far-field, circularly polarized antennas were chosen to be paired with the R420 to adequately cover a 9m x 8m room. These antennae are strategically positioned to cover the entire floor of RFID-tagged items. Finally, the R420 is connected to an in-store network via Ethernet to communicate with the rest of the stack.

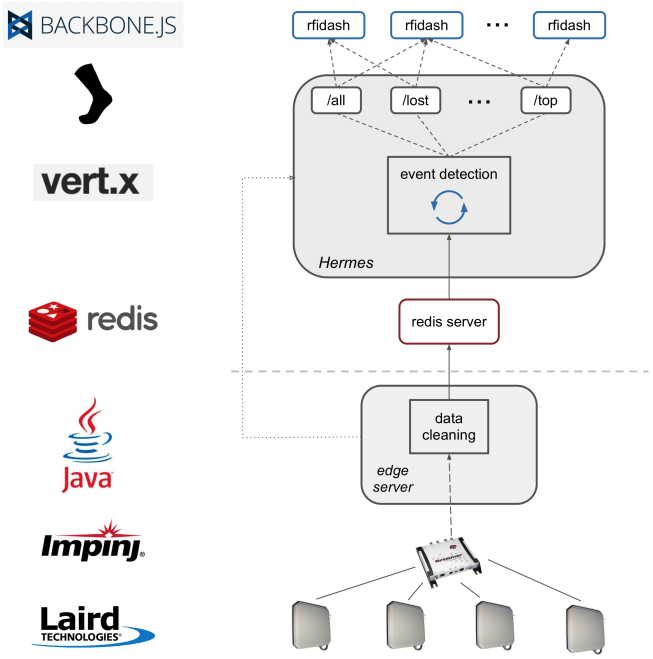

Figure 5: Technology Stack

4.2 Middleware Layer

4.2.1 Requirements

The middleware layer encompasses capturing data from the RFID readers, cleaning the data, storing the data, and then presenting it back to the front end on request. In effect, the middleware can be thought of as an application-agnostic abstraction of our hardware layer. RFID middleware design is a complex undertaking, although there are a few common requirements that are crucial to its success: flexible hardware interaction and control, efficient and accurate data cleaning and processing, and flexible data broadcasting to business-level applications [5].

Direct interaction with the hardware is necessary for reading tags. The R420 reader provides an industry standard interface for RFID readers known as the Low Level Reader Protocol (LLRP). Thus, the first level of our middleware will be responsible for directly controlling the R420 reader via LLRP. Once the readers have been triggered to read, they will return the tags in the form of read events.

The next step is to store the data into a database for future use. However, in order to avoid noisy data, an extra layer of data cleaning is needed before any data can be stored. Bad data could be the result of inconsistent reads, multiple reads, or interference from the surrounding environment. Inconsistent reads and interference will lead to an under-count of items whereas multiple reads will result in an over-count. Therefore, a data cleaning algorithm will be implemented to remove any anomalies before the data is stored to the database.

The data cleaning algorithm also has to interpret the incoming data. The raw data is a simple byte stream and

is essentially meaningless to a store manager. They want to see inventory quantities and not hexadecimal numbers. Therefore, the software has to make sense of the data as well before storing the results.

After the data is cleaned, it will be stored in a central database so that the frontend can access it later. The backend database has to be able to handle high volumes of traffic in a short period of time because once inventory needs to be taken, all the items in the store will be read at once and potentially written into the database within seconds. On top of this, the database will be constantly accessed by both the backend (to store the data) and the frontend (to read the data). As a result, the database has to have reliable uptime and cannot suffer from outages.

Once the data has been stored, it can be used by the frontend application. This application is a tool for managers to conveniently grasp an overall picture of their store's inventory. Developing the application requires access to the database, but dealing with actual database queries is not an ideal solution. Instead, the backend will also provide an Application Programming Interface (API) for the frontend to use so that it can communicate with the database without any hassle.

4.2.2 Implementation

The language family of choice for our middleware includes Java Virtual Machine (JVM) based languages such as Java and Groovy. The JVM has proven its reliability and performance over the years, and in addition to these advantages offers unique benefits for dealing with RFID hardware. For example, to deal with the data capture, the Java LTK library provides an interface that enables simple interaction with the Impinj Speedway Revolution Reader [7, 8]. Writing the data capture in Java allows for portability and an easy installation process when dealing with new stores. Java also makes data cleaning and storing easier due to well-documented official and third-party language bindings for databases.

In terms of overall system architecture, the Amazon Web Services (AWS) cloud platform was the preferred choice for distributed components, such as central servers. This design choice was made in order to improve our system's scalability and fault tolerance by leveraging AWS's enterprise-level datacenters and expertise in providing managed cloud services. A key advantage with using AWS is the automatic scaling flexibility it offers. Whether reading ten or a million tags into a database, AWS can seamlessly scale to meet these needs. Application servers are hosted on Amazon EC2 virtual instances on the West and East coasts to limit latencies and ensure reliability for our system. Using EC2 also offers the unique capability for dynamically resizing compute capacity, which may be necessary for stores with high tag movement. Redis, a NoSQL in-memory key-value store is used for persistent storage of read events. Redis automatically dumps data to disk periodically to ensure data protection, while executing in-memory to optimize performance.

The actual algorithm for data cleaning involves examining the RFID reads to determine whether or not the data stream is accurate. To take care of inconsistent reads, a time frame of reads is taken into account. For example, if an item appears on the first read, disappears on the second read, and reappears on the third read, it can be marked as a possible inconsistency and the second read can be ignored as seen in Figure 6. Taking care of multiple reads is slightly less complicated. If the same ID number appears twice in the same read, then it is a multiple read and therefore should only be counted once [11, 19].

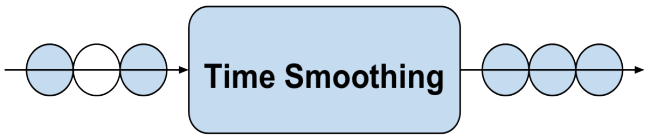

Figure 6: Time Smoothing Algorithm for Data Cleaning

Data cleaning also means data interpretation. The raw data stream consists of packets of data that follow the EPC convention. After the data is smoothed out, the software parses out the specific headers and product information in each data packet. This way the product can be correctly identified and counted before being stored in the database.

In order to provide a flexible structure for data consumption by the application layer, our middleware supports the publish/subscribe (PubSub) messaging pattern in the spirit of event-driven programming. This API design allows loose-coupling between the physical RFID hardware and the frontend application layer. "Topics" were created for relevant events (for example, a specific reader or item category) that can then be selectively subscribed to by the application layer. This allows the frontend to only receive information it is interested in using, rather than being flooded by updates from every single read event. For instance, if the frontend is currently only interested in examining "top-sellers" tag updates, it can subscribe to a "top-sellers" topic so it will only be notified of events related to the most popular items in a store. This helps make the frontend lightweight by avoiding filtration at the application level.

Concretely, the asynchronous web microframework Vert.x was used for implementing the pub-sub system, with fast, full-duplex communication over the WebSocket protocol [30, 3].

4.3 Application Layer

4.3.1 Requirements

In a busy retail store, a manager must be able to quickly access the application and view important inventory management vitals at any given moment, so this application will have different design requirements from most frontend application layers. Usability, interaction, and speed will be key areas of focus. The frontend layer should be device agnostic to allow a manager to access the frontend application from any tablet, laptop, or computing device, so the implementation will target the web browser. Since this works on any device, the application will follow responsive design in CSS/HTML, where the cascading style sheets automatically adapt to changing screen sizes and optimally display the data regardless of the minimum or maximum screen size. Additionally, network outages are a possibility, so offline functionality will have to be supported. The application also needs to have asynchronous UI design - any interactions

that trigger remote GET/POST requests should not block any user interaction (for example, saving notification settings that require confirmation from the server won't popup a "saving alert" that prevents the user from continually using the application).

4.3.2 Implementation

For web applications, JavaScript is the main and only language of choice. However, most JavaScript applications become big and hard to manage, especially since file loading is blocking and application logic is not easily separable from model and controller logic in the web browser. With this in mind, the application layer is done in a Model View Controller (MVC) framework to enable high reusability/updatability of the models and clear separation between logic and models. There are many MVC frameworks, but Backbone.js is used for its simplicity and easy ability to override default functionality by using JavaScript's prototypical inheritance features. An added benefit is its easy data synchronization with our RESTful middleware API.

To facilitate multi-file development (the web unfortunately doesn't have simple "include header.js" without blocking the browser calls, which are extremely slow), a plugin called Require.js is used. Require.js allows for easy additions to a dependencies list for each JavaScript file to promote asynchronous file loading. As Backbone.js and other JQuery libraries start to follow the Asynchronous Module Definition API, Require.js can be removed.

HTML5's local storage is used to implement a simple data storage model for offline content. As much as data storage is a huge emphasis, proper data validation and cleaning will be implemented to prevent unauthorized access based on fake data in local storage, which will be done through Backbone.Model's default validation methods.

SockJS, a library to abstract WebSockets, is used to allow for push notifications to and from the frontend layer to the backend layer without having to constantly poll for updates.

For deployment, Grunt.js is used to concatenate and compress all the JavaScript files into a single file to reduce load times. Finally, since everything is implemented in JavaScript, CSS, and HTML (static files), much of the caching is taken care of on the frontend application layer by distributing it across multiple content distribution networks such as Cloudfare and AWS.

The frontend receives real-time updates through the global RFIDash PubSub system as seen in Figure 7, which is implemented on top of Backbone.Events under the namespace "app.vent." As SockJS from the middleware sends out "event:add", "event:change", and "event:delete" events, RFIDash PubSub takes the events and directs them to the appropriate "dashboard" namespace. Then, the frontend portion will update the appropriate collections which results in a re-rendering of the frontend in real-time.

The frontend portion for the dashboard mode will be made up of several views and subviews, as shown in Figure 8. The *ItemListView* will contain a list of all the inventory items sorted by quantity in ascending order. Each *ItemView* listens to their respective *ItemModel* for changes. If any of these attributes (especially quantity) changes, it triggers a re-render in the corresponding *ItemView*. If the quantity crosses into a different notification level, which is set in the ItemModel, the ItemView will temporarily flash yellow before fading.

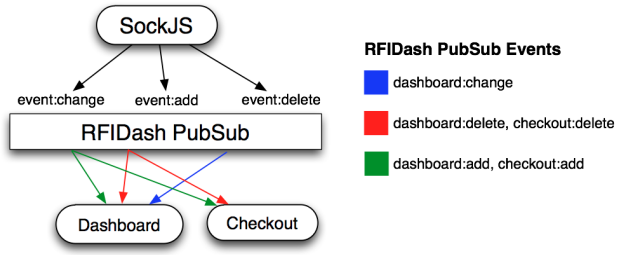

Figure 7: RFID PubSub System

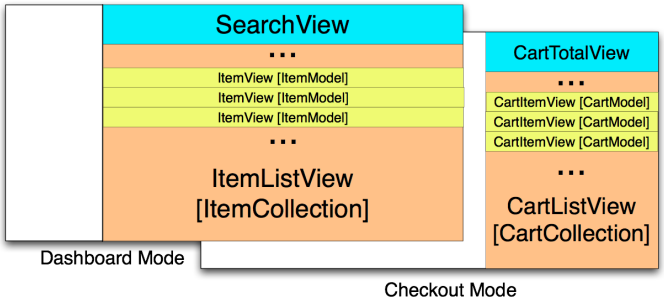

Figure 8: Different Views for RFIDash's Frontend

The search filtering works by spawning a web worker (i.e. background task in the web browser) to do a simple regex filter task. As the results are sent back, the *ItemCollection* is updated via an event (again, using the RFIDash PubSub) and only those items matching the search criteria are returned and rendered.

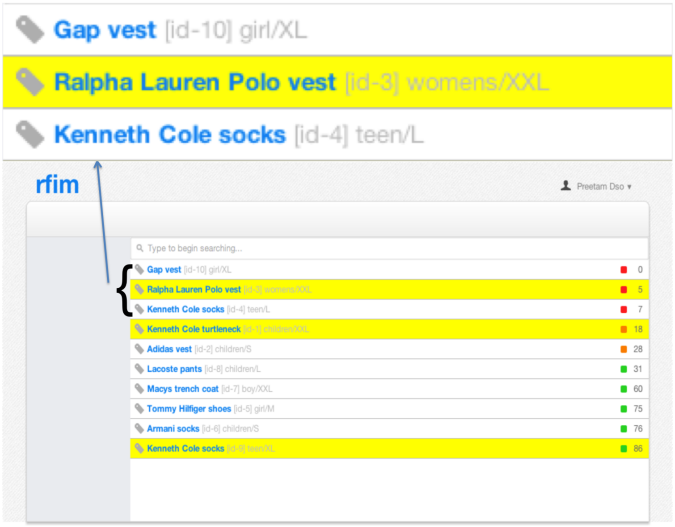

Figure 9: Frontend List View

5. SYSTEM RESULTS

In line with the goals and experimental approach outlined above, inventory accuracy and average system costs were calculated from test data to understand RFIDash's performance and practicality.

5.1 Inventory Accuracy

After RFIDash was built, its inventory performance was assessed by conducting thorough testing within a 9m x 8m room. The reader was placed near the center of the room, with the four antennae being placed as far as possible diagonally outwards from the central reader in a rectangular configuration. 43 passive RFID tags were positioned at random around the room at varying distances and heights from the antennae. The test tags used were an assorted random sample of common EPC UHF Class 1 Generation 2 (Gen2) tags. The test tags differ by antenna shape, size, and length. This ensures that the testing, and more importantly, the results gathered are as general as possible.

RFIDash was set to read tags for a duration of 0.9 seconds within every 1 second while only pushing data on every tenth read and the final accuracy was reported as the number of tags seen during the read duration divided by the total number of tags (43). A tag was marked as read if it registered at least once during the read window.

Initial accuracy results were in the 75-80% range, which were surprisingly low. After further experimentation, it became apparent that several of the test tags were faulty and were displaying intermittent signal strengths. This was determined by repeating the experiment with one tag at a time. At the end of the procedure, 8 of the 43 tags were isolated, deemed weak, and removed from the test tag set for the next test iteration. It is widely known among the RFID community that several tags in a set are usually defective and these findings only support that fact [26].

Testing with the new set of 35 properly functioning tags (as seen in Figure 12) immediately showed increased accuracies in the 90% range. However, this was with poor antennae placement. With improved antennae locations, accuracies up to 100% were seen.

In further testing, obstructions were placed near tags in order to test different material effects on accuracy. Cardboard boxes, fabric, glass, and plastic materials did not interfere in any way with accuracy. However, metals and liquids very close to tags can completely eliminate their signal.

RFIDash clearly improves upon the 65-75% accuracies that are typical in existing IM systems by bringing rates up to the 90% range [15]. This is a significant baseline improvement of 20-40%, not to mention the significant advantages of having an IM system that is fully automated and instantly responsive.

5.2 System Costs

The system costs mainly come from the hardware. The Speedway Revolution R420 reader itself costs $1500. Each antenna costs $120, for a total of $480 with 4 antennae. With this setup, a 9m x 8m room can be covered accurately. The tags, if bought in bulk, end up being 7 to 11 cents per tag [27]. This amounts to a total of $27.50 per square meter.

However, this can be interpreted as an upper bound on the cost of the system since each reader can support up to a total of 32 antennae. The area covered can then be expanded to approximately eight times the original coverage. The additional hardware required for this would be 4 antenna hubs and the extra antennae. Each hub costs $265, for a total of $1060. With this new setup, the cost of the system drops to $11.11 per square meter. This is a lower bound on the system cost because it optimistically assumes all 32 antennae cover non-overlapping 9m x 8m spaces [1].

A good approximation for RFIDash's true cost would be somewhere between these two bounds. A reasonable estimate would be the average of about $20 per square meter, or $1.86 per square foot. Given that average sales per square foot in a typical American mall is roughly $400, this is a reasonable sum [6]. To put this figure into perspective, it would cost only $20,000 to completely cover an average 10,000 ft^2 retail store like Gap.

6. APPLICATION CASE STUDY

One of the key reasons why retailers fail to convert customer traffic into sales is long checkout lines [4]. Modern checkout systems need to be fast–a perfect use case for RFID's multi-item reading capabilities.

To better understand the power of RFIDash as an extensible real-time inventory platform, consider the design of a practical RFID-based fast checkout system that relies on RFIDash's hardware and software stack. This application case study will hopefully showcase the ease of extending RFIDash to create other useful real-time based applications.

6.1 Requirements

In order to implement fast checkout, there must first be a way to confine the read area of the RFID antenna to a certain region. This prevents items from being accidentally checked out and bought. For example, if there is a line for the checkout lane, only one person's items should be scanned at a time. The system should not be picking up the items being bought by the next person waiting in line.

Once the read zone can be confined, a simple and intuitive frontend checkout view is needed to allow shoppers to see the individual items they are buying and a tally of the total cost. While the dashboard mode benefits the store manager, a new frontend checkout mode is needed that is geared towards helping customers during the checkout process. To do this, the checkout mode introduces a real-time system where each item does not have to be manually scanned. As the RFID tagged items pass the antennas, the web application will show a checkout cart being updated in real-time with each item name, quantity, and price of that item(s). The total price in the checkout cart is updated at all times. Finally, although the checkout mode shares similar requirements to the dashboard mode, checkout mode adds on an extra layer of requirements. From a checkout point of view, item accuracy is extremely important (i.e. it is important not to miscalculate prices or drop certain items), so additional attention will be made to ensure price accuracy is maintained.

6.2 Implementation

The algorithm used to confine the tags that are picked up is based on Received Signal Strength Indication (RSSI). RSSI is the measurement of the power of the signal being read and its units are in decibels. As the tags are scanned by the antennae, the strength of the signals coming from each tag are known. Using this information, a threshold value is set in the code that confines the read area to only the area around the checkout antennae. This threshold value can be adjusted as needed since different readers allow for a different range of decibel values.

Once the RSSI algorithm was implemented at the hardware layer, the rest of RFIDash's infrastructure could be used, unmodified, to create a working prototype of fast check-

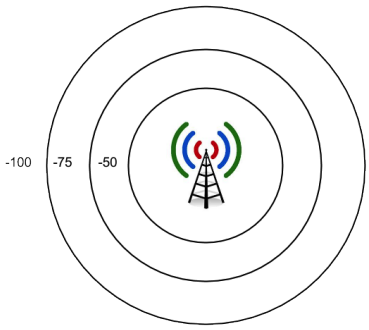

Figure 10: Received Signal Strength Indication

out. The middleware layer adds a new "checkout" topic that isolates checkout events from inventory events consumed by RFIDash's dashboard mode. After this minor addition, the only significant change that had to be made was to the frontend in order to create a checkout-style interface.

The checkout frontend portion is made up of several views and subviews, similar to the $ItemListView$ for dashboard mode. A new view called the $CartListView$ contains a series of $CartItemViews$, each of which is backed by a $CartModel$ (see Figure 8). Like $ItemView$, $CartItemView$ listens for changes in the $CartModel$. If anything changes (e.g. the quantity of the specific item), the price for that $CartModel$ is recalculated. $CartTotalView$ keeps a running total of the total cost and is re-rendered when any of the $CartModels$ change.

As seen in Figure 7, RFIDash PubSub routes events to the appropriate "dashboard" or "checkout" namespace. The frontend portion of dashboard or checkout will then update their appropriate collections which results in a re-rendering of the frontend in real-time. It is worth noting the power of RFIDash's PubSub architecture, since new topics for new applications can easily be added.

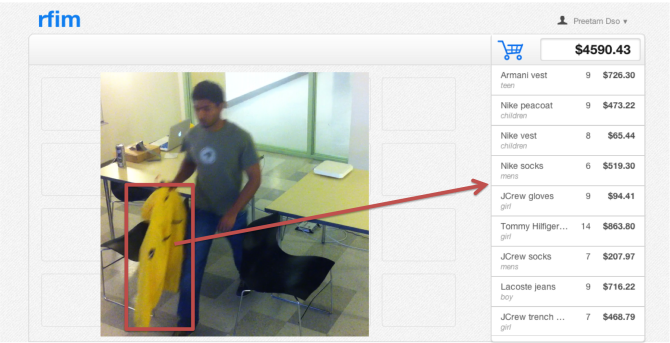

Figure 11: Zero Wait Time with Transparent Checkout

6.3 Results

RFIDash's checkout mode was tested after the inventory mode. The setup for this mode is different than the setup for the inventory mode. Instead of having the four antennae being placed at the corners of a rectangle, only one antenna was used. It was placed perpendicular to the ground at a doorway in order to simulate a checkout aisle. For our reader, the received signal strengths varied between -40 dB and -70 dB. The threshold was set to -55 dB in order to create a box of about one meter in each dimension. Preliminary testing consisted of a shopper walking by the doorway at a normal walking pace, with no pausing – this resulted in about 80% accuracy. However, a more realistic checkout scenario would involve the shopper pausing in front of a checkout screen to check the total price of his/her goods and make the final payment. Accuracy rates quickly exceed 90% in this scenario.

With the dashboard and checkout mode, RFIDash offers examples of two practical applications that can be built on top of the middleware layer's API. However, that doesn't limit other other applications (e.g. analytics, email notifications) to be built - these two modes are just a sample of what is possible. Although further calibration is needed to ensure the highest level of accuracy needed for a commercially deployable checkout system, the ease and speed of development with RFIDash's existing infrastructure showcases the system's potential as a powerful backbone for building real-time applications.

7. FUTURE WORK

The basic functionality of RFIDash is in place, but there are still further improvements that could be made. RFIDash accurately tracks inventory and speeds up checkout in real time, but there are still huge technical hurdles in getting RFIDash to production at a real store, which is the ultimate goal. To do so, more testing, automation, and integration from all three tiers of the system (i.e. hardware, backend, and frontend) will be required.

7.1 Large Scale Testing

Much testing has already been done with a variety of different obstacles in a 9m x 8m room to show the potential of RFIDash, but most stores are not confined to a 9m x 8m room. In order to cope with larger store layouts, testing needs to be performed with many more antennae and possibly even multiple readers. RFIDash was designed in a loosely-coupled fashion, so the addition of readers and antennae will not affect any part of the system except the lowest, embedded hardware layer. Nevertheless, logistical issues like reader and antenna placement and wiring need to be addressed for large scale deployments.

7.2 Full Retail Integration

Full integration with existing IM and checkout systems is necessary in order to commercially deploy RFIDash in a retail store. This entails building support for writing custom store EPCs to blank tags, incorporating an RFID printer into the overall system so that stores can print new labels for items, offering a manual correction application in case of power/system outages, and hooking into current point of sale software systems. Additionally, RFIDash's inventory and checkout modes exist separately from each other; they will need to be integrated in order for them to be used concurrently.

7.3 Automation

Accuracy testing was conducted with 43 individually labeled RFID tags, but testing automation will become increasingly important in order to test RFIDash at scale. This

will involve connecting a RFID writer to RFIDash to automate the printing/entering of new items into our system.

7.4 Transparent Data Collection and Sharing

One of the greatest possibilities with an RFID IM system is the ability to quickly and easily share information along the entire supply chain from manufacturer to warehouse to retailer. Ideally, items are tagged with RFID chips as soon as they are manufactured. An integrated RFID system would allow these tagged items to be tracked by all interested parties as it advances along the supply chain. In order to make this a reality, RFIDash would need to be adapted so that it could be applied at earlier stages of the supply chain. Although this is a longer term vision, it is a powerful motivation for the widespread use of RFID.

8. ETHICS

Since RFIDash employs RFID technology ethical issues arise from the trackability of RFID chips and customer privacy. As a customer leaves a store, the store item will still be tagged with a RFID chip until he/she decides to remove the chip embedded within the price tag. Up until the chip is destroyed, anyone with a RFID reader could potentially read the data on these tags. For example, large retailers can drive around with cars equipped with long range RFID readers and collect geolocation and data on RFID chips. This rich data set would contain a map of products and exactly where they were found, which could be used for carefully targeted advertising. Although clearly an extreme scenario, this detailed profile of each household could pose a huge privacy threat. To protect against this, RFID chips could be encrypted [32] or "killed" after the checkout process. Although the RFID chips could be encrypted, it raises the prices of RFID tags from ten cents to five dollars [22] and RFIDash would be hard pressed to remain cost-effective. Instead, as the items go through checkout, the RFID chip can be killed [20] through a variety of means, rendering it unable to send back signals to the long range RFID readers. As another layer of security, all EPC Gen 2 compliant tags (which RFIDash uses) support write locking tags with a password to prevent tag overwriting [9].

With the long range RFID readers in a typical store setup, it's also possible that RFIDash will pick up RFID tags from other systems. However, RFIDash strictly filters out any tags not belonging to the store system to avoid collecting private information. One area of concern within the stores, however, is forging RFID signals. People with RFID signal cloners could potentially clone the signals from RFID tags and confuse the inventory management system in place. Fortunately, all tags contain an unique ID so RFIDash filters out duplicate EPC data. Physical tag exchanging (i.e. replacing the tag of one item with that of a cheaper item) can still happen as they do in most stores, but that is outside the scope of RFIDash. Ultimately, if tampering with the RFIDash read system does happen, it will not actually affect the products sold at the point of sale, but will only slightly skew real time analytics, which will return back to normal after the cloned signals leave the store.

9. CONCLUSION

Inventory management (IM) is an important tool for retailers to not only predict sales trends but to also keep tabs on stock levels. The current state of IM systems involves updating through manual counting, barcode scanning, and (rarely) integration with point-of-sale software. RFIDash fixes these problems by providing a real-time solution that automatically updates the count of each item through an all-in-one, RFID-based hardware and software system. With this real-time system, there is no need to go around each week tediously hand counting in-store stock, allowing store employees to spend more time on restocking and helping customers.

From a hardware aspect, RFIDash not only sets up an optimal coverage of RFID tags, but also provides a smart data filtering level that abstracts away the complicated data processing step as described in Section 2.3. This data filtering removes duplicate reads and adjusts the coverage range, allowing it to be adaptable for any type of room configuration and extendable to any system requiring a real-time, accurate tracking system.

There are many IM systems out there, but RFIDash offers a fully event-driven platform on top of which real time applications can selectively subscribe to different channels of actions (e.g. new shipment, changes in quantity, item deletion, checked out). The level of detail offered by these different channels results in endless possibilities: the user can tell exactly which items are selling well, and when an item is out of stock, it is highlighted in red to alert the user. The dashboard mode in RFIDash provides not only this, but an easy to search interface, allowing store owners to pinpoint the exact item they are searching for with great flexibility and precision.

In order to demonstrate the powerful infrastructure that RFIDash offers as an extensible platform for building real-time applications, fast checkout is also supported within RFIDash. There is a pressing need for efficient checkout mechanisms since checkout lines can be extremely long, especially in congested areas or during times of natural disaster. RFIDash's checkout feature repurposes standard RFID hardware to automatically scan items as they pass through a checkout aisle. Aside from encouraging customer visits and streamlining sales throughput by saving checkout time, RFIDash reduces costs for stores since employees can now focus on more pressing tasks.

In the end, RFIDash provides a framework for retailers to deploy real-time applications; in this case, tracking inventory and helping speed up checkout. As manufacturing efforts grow and retailers face an increasing demand for products on the shelves, gathering data on inventory and sales trends will be crucial to maintaining a competitive advantage. Although more testing and future work needs to be done, RFIDash is capable of solving many of these problems with it's real-time tracking and analysis, simple hardware setup, and intuitive dashboard UI.

10. REFERENCES

[1] BuySpeedwayRevolution. `http://www.buyspeedwayrevolution.com/`.

[2] EPCglobal. Electronic product code (epc): An overview. `http://www.gs1.org/docs/epcglobal/an_overview_of_EPC.pdf`.

[3] I. Fette and A. Melnikov. The websocket protocol. `http://tools.ietf.org/html/rfc6455`.

[4] Marshall L. Fisher, Jayanth Krishnan, and Serguei Netessine. Retail store execution: An empirical study.

[5] Christian Floerkemeier and Matthias Lampe. Rfid middleware design - addressing application requirements and rfid constraints.
[6] Elizabeth Holmes. Sizing up property, mall stores try to shrink. http://online.wsj.com/article/SB10001424052748704059004575127711780438090.html.
[7] Impinj. Creating rfid applications with java. http://learn.impinj.com/articles/en_US/RFID/Creating-RFID-Applications-with-Java/.
[8] Impinj. Impinj ltk programmers' guide.
[9] Impinj. Locking memory on epc rfid tags. http://learn.impinj.com/articles/en_US/RFID/Locking-Memory-on-EPC-RFID-Tags/.
[10] Impinj. Speedway revolution uhf rfid reader. http://www.impinj.com/Speedway_Revolution_UHF_RFID_Reader.aspx.
[11] Shawn R. Jeffery, M. Garofalakis, and Michael J. Franklin. Adaptive cleaning for rfid data streams.
[12] Xiao Jia, Yuan Wenming, and Wang Dong. Complex event processing model for distributed rfid network.
[13] RFID Journal. Wal-mart details rfid requirement. http://www.rfidjournal.com/article/view/642.
[14] A. R. Wasif K. G. Tan and C. P. Tan. Objects tracking utilizing square grid rfid reader antenna network.
[15] Y. Kang and S.B. Gershwin. Information inaccuracy in inventory systems: Stock loss and stockout.
[16] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints.
[17] RFID Sherpas LLC. Know what's in store with rfid.
[18] Matthew Malone. Did wal-mart love rfid to death? http://www.smartplanet.com/blog/pure-genius/did-wal-mart-love-rfid-to-death/7459.
[19] Elio Masciari. Rfid data management for effective objects tracking.
[20] Mike. How to block/kill rfid chips. http://www.instructables.com/id/How-to-blockkill-RFID-chips/step4/How-to-kill-your-RFID-chip/.
[21] Motorola. Understanding gen 2: What it is, how you will benefit and criteria for vendor assessment.
[22] Annalee Newitz. The rfid hacking underground. http://www.wired.com/wired/archive/14.05/rfid.html.
[23] PennState. The impact of rfid on supply chains. http://research.smeal.psu.edu/news/the-impact-of-rfid-on-supply-chains.
[24] Mark Roberti. The history of rfid. http://www.rfidjournal.com/article/view/1338/2.
[25] Mark Roberti. Wal-mart relaunches epc rfid effort, starting with men's jeans and basics. http://www.rfidjournal.com/article/view/7753.
[26] SecurityInfoWatch. Vendor says new rfid labeling technology cuts out defective chips. http://www.securityinfowatch.com/press_release/10592179/vendor-says-new-rfid-labeling-technology-cuts-out-defective-chips.
[27] Louis Sirico and Mark Davenport. Passive uhf rfid tags and smart labels buyer's guide. http://rfid.net/best-practices/43-best-practices/135-passive-rfid-smart-label-buyers-guide.
[28] SkyRFID. Rfid gen 2 - what is it? - smart rfid! http://www.skyrfid.com/RFID_Gen_2_What_is_it.php.
[29] Trey Thoelcke and Michael B. Sauter. The 9 most successful retail stores in the usa. http://www.usatoday.com/story/money/business/2012/11/18/most-successful-retail-stores/1710571/.
[30] Vert.x. Vert.x - effortless asynchronous application development for the modern web and enterprise. http://vertx.io/.
[31] Samuel F. Wamba, Louis A. Lefebvre, and Elisabeth Lefebvre. Enabling intelligent b-to-b ecommerce supply chain, management using rfid and the epc network: A case study in the retail industry.
[32] Junyu Wang and Xi Tan. Encrypt instead of 'kill'. http://www.rfidjournal.com/articles/view?9700.
[33] Roy Want. The magic of rfid.

APPENDIX

Figure 12: Final 35 RFID Test Tags

Figure 13: RFID Hardware