Efficient Data Interpretation and Compression over RFID Streams

Richard Cocci, **Thanh Tran**, **Yanlei Diao** and **Prashant Shenoy**
Department of Computer Science, University of Massachusetts Amherst

Abstract

Despite its promise, RFID technology presents numerous challenges, including incomplete data, lack of location and containment information and very high volumes. In this work, we present a novel data interpretation and compression substrate over RFID streams to address these challenges in enterprise supply-chain environments. Our substrate employs a time-varying graph model to efficiently capture inter-object relationships such as co-location and containment. It then employs a probabilistic inference algorithm to determine the most likely location and containment for each object and an efficient stream compression algorithm to remove redundant information from the output stream. We have implemented a prototype of our interpretation and compression substrate and have evaluated it using synthetic RFID streams that emulate a warehouse and supply-chain environment. Our results show that our inference techniques provide good accuracy while retaining efficiency, and our compression algorithm yields significant reduction in data volume.

1 Introduction

RFID is a promising electronic identification technology that enables a real-time information infrastructure to provide timely, high-value content to monitoring and tracking applications. An RFID-enabled information infrastructure is likely to revolutionize areas such as supply chain management [14], healthcare [14], pharmaceuticals [14], postal services [17], and surveillance [18] in the coming decade.

Data stream management is central to the realization of such a monitoring and tracking infrastructure. While data stream management has been extensively studied for environments such as sensor networks [31, 24, 7, 8, 27, 28], existing research has mostly focused on sensor data that captures continuous environmental phenomena. RFID data—a triplet <tag id, reader id, timestamp> in it's most basic form—raises new challenges since it may be insufficient, incomplete, and voluminous.

Insufficient information: Since RFID is inherently an identification technology designed to identify individual objects, a stream of RFID readings does not capture inter-object relationships such as co-location and containment. For instance, a RFID stream does not directly reveal what items are packaged in which cases or which pallets are adjacent to one another on a warehouse shelf.

Incomplete data: Despite technological advances, RFID readings are inherently noisy with observed read rates significantly below 100% in actual deployments [10, 19]. This is largely due to the intrinsic sensitivity of radio frequencies (RFs) to environmental factors such as occluding metal objects [11] and contention among tags [12]. Missed readings result in lack of information about an object's location, making the tasks of determining location, containment and anomaly detection significantly more complex.

High volume streams: Perhaps the key distinguishing characteristic of RFID streams are their high volumes—large deployments are expected to generate unprecedented volumes of data that can overwhelm the data stream system. For instance, the largest retailers are expected to generate millions of terabytes of data in a single day when tagging each item [29]. Hence, it is imperative that data be filtered and compressed close to the hardware while preserving all useful information.

Recent research on RFID data cleaning [13, 19, 20] has employed temporal and spatial smoothing to alleviate the problem of missed readings. Although restricted to per-tag cleaning or multi-tag aggregate cleaning in a given location, these techniques do not capture relationships between objects or estimate object locations. Probabilistic RFID processing has been recently proposed as a more general solution [15] but this research is still in its infancy. Compression techniques for RFID warehouses use expensive disk-based operations such as sorting and summarization [16] or employ application-specific logic [30]. Hence, they are unsuitable for fast low-level compression of RFID streams.

In this paper, we present SPIRE, a system that addresses the above challenges. SPIRE departs from the prior work by building an *interpretation and compression* substrate over RFID data streams. Such a substrate enables accurate interpretation of observed data, even though the raw data is insufficient and incomplete. Further, it infers inter-object relationships such as containment and location. Finally, it enables online compression by discarding redundant data such as an unchanged object location or an unchanged containment between objects. Online compression significantly reduces data volume, thereby expediting processing and reducing data transfer costs.

Our interpretation and compression substrate employs three key techniques, which are also main contributions of this paper: 1) a time-varying graph model that captures possible object locations and containment relationships with its stream-driven construction, (2) a probabilistic algorithm that infers the most-likely locations and containment relationships of objects, and (3) an output algorithm that transforms an input stream to a compressed, yet informative output stream. We have implemented our interpretation and compression substrate and have evaluated it using synthetic RFID streams emulating an enterprise supply-chain environ-

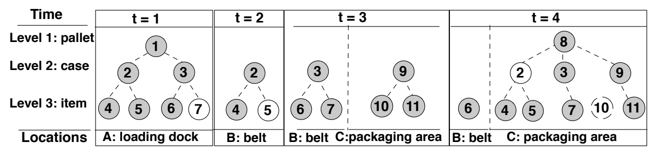


Figure 1: A sequence of observations in a warehouse.

ment. Our results show that our inference techniques provide good accuracy while retaining efficiency, and our compression algorithm yields significant reduction in data volume.

The rest of the paper is organized as follows. Section 2 formulates the problem. Sections 3, 4, and 5 describe our interpretation and compression substrate. Section 6 presents our experimental results. Finally, Section 7 presents related work, and Section 8 our conclusions.

2 Problem Statement

Before defining the problem, we present the notion of the physical world. A *physical world* covers a specific geographical area comprising a set of objects O, a set of pre-defined, fixed locations L, and an ordered discrete time domain T. The set of locations can be either pre-defined logical areas such as aisle 1 in warehouse A, or (x, y, z) coordinates generated by a positioning system.

At time instant t, the *state of the world* includes:

1. the set of objects present in each *location,* encoded by the boolean function $\_resides$($o_i \in O$, $l_k \in L$, t) which is true iff object o_i is present at location l_k; and

2. the *containment* relationship between objects, encoded by the boolean function $\_contained$($o_i \in O$, $o_j \in O$, $l_k \in L$, t), which is true iff both objects o_i and o_j are in location l_k and o_i is contained in o_j.

In this work, we refer to the functions $\_resides$ and $\_contained$ as the *ground truth.* The state of the world changes whenever an object enters the world, an object exits the world through a designated channel, or an existing object changes its location or containment relationship with other objects. The set of locations L also contains a special location called *unknown*. In particular, an object can be in the "unknown" location if it is not present in any pre-defined location (e.g., if it is in transit between two locations) or if it exited the physical world improperly (e.g. was stolen).

RFID readers provide a means to observe the physical world. The readings produced at time t are collectively called an *observation of the world.* In this work, we focus on readers mounted at fixed locations—a common configuration in today's RFID deployments. For such fixed readers, a reading captures the location of the object (which is the same as the location of the reader). [1] Such readings, however, are inadequate for capturing the containment between objects. The observation of the world may be incomplete since some objects may not respond to reader queries (due to technological limitations).

[1]The assumption of precise location does not hold when mobile readers are used. Interpretation and compression of mobile reader data is a topic of our future work.

The **data interpretation** problem is to construct an approximate yet accurate estimate of the state of the world based on the observations thus far. We define an approximation using functions $resides$ and $contained$ that given specific arguments, return probabilistic values representing the likelihood of the function being true. Then the data interpretation problem can be formulated as: given the current time now and an object o_i, report

1. the most likely location of the object, denoted by $argmax_k\ resides(o_i, l_k, now)$, and

2. the most likely container of the object, denoted by $argmax_{j,k}\ contained(o_i, o_j, l_k, now)$.

Note that in our definition, data interpretation over streams is only concerned about the present state of the physical world and not the past or the future.

The **data compression** problem is to transform the input stream into an output stream with a reduced data volume but with no loss of information. Such compression requires the knowledge of what data is redundant and thus can be safely discarded; in this work, we use interpretation to obtain such knowledge. The combination of interpretation and compression yields an output stream that (i) augments the input stream with additional, likely information about objects, and (ii) has a significantly reduced volume of data.

A running example. A warehouse scenario is depicted in Figure 1, where RFID readers are installed above the loading dock, the conveyor belt, and the packaging area. At time t=1, the reader at the loading dock reports objects 1 to 6, denoted by the shaded nodes. These nodes are arranged according to the packaging levels that the reported tag ids indicate [9]. Object 7 is also present but was missed by the reader, denoted by an unshaded node, i.e. a *missed reading*. Containment between objects, depicted by the dashed edges, is not reported by the readings and often uncertain. Examples of *ambiguous containment* are the containment between items 4, 5, 6 and cases 2, 3 at this time.

At time t=2, case 2 is scanned individually on the conveyor belt. It is possible to confirm the containment between the case and its item(s) now if the domain knowledge of the deployment reveals such *special readers* that scan containers of a particular type one at time. At t=3, case 3 is scanned on the belt and a new case 9 is read in the packaging area.

At time t=4, item 6 is read at the belt again (it fell off its case at t=3 and stayed here). A new pallet 8 is assembled from the three cases in the packaging area. Of particular interest is item 10 that was removed from its case and reveals no further information, i.e. a *missing object*. A missing object should be distinguished from a missed reading of an existing object (e.g. case 2 at this time).

We have developed a data interpretation and compression substrate to address the above problem. The substrate, which is depicted in Figure 2 consists of (i) a *data capture* module that implements a stream-driven construction of a

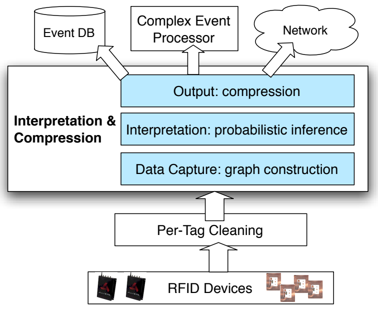

Event DB
Complex Event Processor
Network
Output: compression
Interpretation & Compression
Interpretation: probabilistic inference
Data Capture: graph construction
Per-Tag Cleaning
RFID Devices

Figure 2: Architecture of SPIRE.

time-varying graph model to encode possible object locations and containments; (ii) an *interpretation* module, which employs probabilistic inference to estimate the most likely location and containment for an object, and (iii) a *compression* module that outputs stream data in an compressed format with smoothing. The next three sections describe these techniques in detail. This substrate runs on a low-level per-tag cleaning module such as [19]. It is important to note that the only required functionality of per-tag cleaning for this work is deduplication, which removes redundant readings of a tag that resides in overlapping ranges of readers.

3 Data Capture

This section describes our data capture technique to construct a time-varying graph model from the raw stream.

3.1 A Time-Varying Colored Graph Model

Our graph model $G = (V, E)$ encodes the current view of the objects in the physical world, including their reported locations and (unreported) possible containment relationships. In addition, the model incorporates statistical history about co-occurrences between objects. Example graphs for the observations in Figure 1 are shown in Figure 3.

The node set V denotes all RFID-tagged objects in the physical world. Since we are assuming a supply-chain environment, an object has a packaging level of an *item*, *case* or a *pallet*; the packaging level is encoded in the RFID tag ID [9]. Our graph is arranged into layers, with one layer for each packaging level. Each node has a color that denotes its location; thus a node may be assigned one of $(L - 1)$ colors, one for each known location. A node is uncolored if its location is currently unknown. The node colors are updated using the stream of readings in each epoch (the color of a node is the color of the location where it is observed by a RFID reader). If an object is not read by any reader in a particular epoch, its node becomes uncolored. However, uncolored nodes retain memory of their most recent color and the observation time denoted by (*recent_color*, *seen_at*).

The directed edge set E encodes possible containment relationships between objects. A directed edge $o_i \rightarrow o_j$ denotes that o_i contains object o_j (e.g., a case i contains item j). We allow multiple outgoing and incoming edges to and from each node, indicating an object such as a case may contain multiple items, and conversely, an item may be contained in multiple possible cases (our probabilistic inference will subsequently chose only one of these possibilities). More generally, edges can exist between different combinations of colored and uncolored nodes, with the exception that an edge cannot connect two nodes of different colors; that is, containment is prohibited for two objects resident in two different locations. We also allow edges to cross layers, for instance to (temporarily) capture the containment between objects in non-adjacent layers when the reader fails to read any of the objects in the adjacent layer at some time. Such flexibility allows the graph to capture a wide variety of containment relationships.

To enable inference, the graph also encodes additional statistics. Each edge maintains a bit-vector *recent_collocations* to record recent positive and negative evidence for the collocation of the two objects. The bit is set every time the two nodes connected by the edge are assigned the same color. Further, each node maintains *past_certain_parent* statistics to remember the last confirmed parent edge, either revealed by a special reader or as a result of inference with high certainty, the time of confirmation, and the number of conflicting observations obtained thus far. Among all incoming edges, only one can be a confirmed edge, denoted by the edge with double arrows in Figure 3.

3.2 Stream-Driven Graph Construction

We assume that time is divided into epochs and the graph is updated using stream data from each epoch. Our construction algorithm, shown in Figure 4, takes the graph G from the previous epoch and a set of readings R_k from each reader k ($1 \leq k \leq K$) in the current epoch, and produces a new graph G^*. An important feature of the algorithm is that it proceeds incrementally as readings arrive from each reader, guaranteeing a consistent output G^* after seeing the readings from all readers in an epoch. This ensures that the algorithm works even when the various readers are coarsely synchronized in time.

Given R_k of each reader, the graph update procedure proceeds in four steps, as shown in Figure 4.

Step 1. *Update and color nodes (lines 2-6):* If a new object is observed for the first time, a new node is created in the graph. For each observed object, the corresponding node is colored with the color of the reader that observed it (a reader is colored with the color of the location where it resides). Figure 5 (a) shows the result of step 1 when the graph update procedure is applied to $G_{t=3}$ with R_C.

Step 2. *Update edges (lines 9-13):* Next, if two nodes in adjacent layers have the same color, an edge is added between them if one does not exist. Doing so enumerates all possible containment relationships (e.g., a blue item can be contained in any of the blue cases that are present on a shelf). If an adjacent layer does not contain a node of a particular color (due to missed readings), an edge may be drawn to a node of that color in the next higher layer. Thus, if a blue item is observed but no blue cases are present in the graph, the item is assigned to a blue pallet.

This step potentially requires us to compare each node in a layer to all nodes in the adjacent layer. A slight optimization

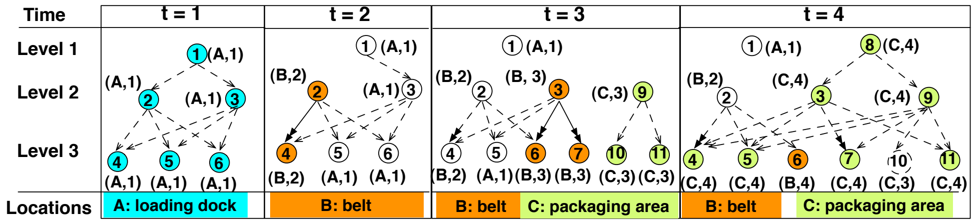


Figure 3: Examples of the time-varying colored graph model.

```
procedure graph_update(G, Rk)
G: current graph, Rk: set of readings from reader k
begin
1.       C = location of the reader k
         /* create and color nodes */
2.       foreach reading r ∈ Rk do
3.           v = node in G corresponding to the object of the reading r
4.  (1)      if v == null then                          // create a new node
5.               v = a new node created in G for this object
6.           v.recent_color = C, v.seen_at = now        // color each node
         /* create, remove, and update statistics of edges */
7.       foreach packaging level L (starting from 1) present in Rk do
8.         foreach each node v in color C at level L do
9.             if C is a new or initial color of v or then    // create new edges
10.              above = closest level above L containing nodes colored in C
11. (2)          below = closest level below L containing nodes colored in C
12.              create edges from the nodes in level above to v if they don't exist
13.              create edges from v to the nodes in level below if they don't exist
14.          foreach edge e incident to v do
15.              v' = the other node of e
16.              if v' has a different color as v then         // remove an edge
17.                drop the edge e from G
18. (3)          if v' is a parent of v and the reader k confirms v as the only
                     top-level container then                  // remove an edge
19.                drop the edge e from G
20.              if e.update_time = now then
21.                continue
22.              right shift e.recent_history to expire old information
23.              if v' has the same color as v then           // update statistics
24.                e.recent_collocations[0] = True
25.                if v' is a child of v and the reader k confirm that v contains v' then
26. (4)              v'.set_past_certain_parent(e, now);
27.              else                                          // update statistics
28.                e.recent_collocations[0] = False
29.                if e is set as the past certain edge of v (or v') then
30.                  v(v').weaken_past_certain_parent(e);
31.              e.update_time = now
end
```

Figure 4: Algorithm for stream-driven graph update.

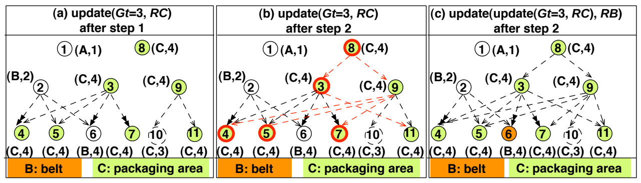

(a) update(Gt=3, RC) after step 1
(b) update(Gt=3, RC) after step 2
(c) update(update(Gt=3, RC), RB) after step 2
B: belt
C: packaging area

Figure 5: Intermediate steps of graph update.

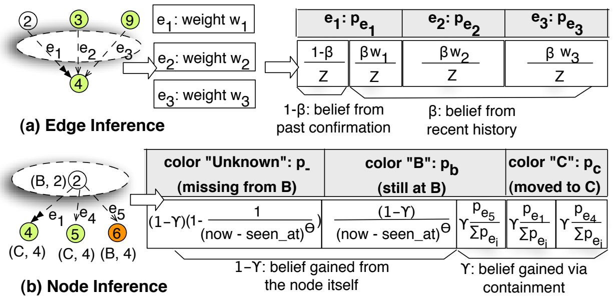
is to restrict such comparisons to the nodes that are assigned the color $\mathcal{C}$ for the first time. Figure 5 (b) illustrates this step.

Step 3. *Prune graph (lines 16-19):* Whereas the previous step adds new edges to the graph, in this step, we remove outdated edges from the graph. An edge is removed if the corresponding vertices have different colors – this happens when two previously co-located objects are now reported in different locations. Similarly, edges can be dropped when special readers confirm a containment, allowing us to eliminate other possibilities (e.g., if a pallet is confirmed to be the container of a case, then all other edges between this case and other pallets can be dropped). Special readers – such as a belt reader that reads exactly one pallet at a time — improve the accuracy of the graph model, while helping us prune unnecessary edges. Figure 5 (c) shows the removal of an edge between nodes 3 and 6 using readings R_B from the conveyor belt. Removing this edge results in the final graph shown for t=4 in Figure 3.

Step 4. *Update Statistics (lines 20-30):* This step updates statistics of the edges that have at least one node colored in step 1. Given an edge e, if the two linked nodes have the same color, *recent_collocations* of e is updated by setting the most recent bit to True. Furthermore, if the reader k is able to confirm the containment denoted by e, it is used to update the *past_certain_parent* of the child node. If one of the linked node is uncolored, the most recent bit of *recent_collocations* is set to False. In this case, we also check if e was set as the certain parent edge of the child node, and if so count the current observation as a conflicting observation of the confirmation. These statistics play a key role in containment inference as we shall show shortly.

Complexity of graph update. The total cost of updating a graph $G(V, E)$ using reading sets $R_1, \ldots, R_K$ is $\sum_k$ cost(R_k). For each reading set R_k, only the colored nodes and their incident edges are considered. As can be seen from the above algorithm, The number of colored nodes in step 1 is $|R_k|$. The cost of step 2 is dominated by the size of the largest complete bipartite graph $(|R_k|/2)^2$.

Steps 3 and 4 together examine every edge incident to a colored node. Node that these edges may connect to an uncolored node or a node of a different color, and thus cannot be bounded only using $|R_k|$. However, if we examine steps 3 and 4 across all readers, the following observations hold: an edge with two different node colors is visited twice, one for each color, and then removed from the graph; an edge with the two nodes in the same color or only one node colored is visited only once; finally, an edge with neither node colored is not visited. We use $\pi_{R_1,\ldots,R_K}(G)$ to denote the projection of the original graph onto the nodes belonging to $R_1, \ldots, R_K$ and the edges incident to those nodes. Then the cost of steps 3 and 4 across all readers is bounded by the number of newly created edges plus twice the size of $\pi_{R_1,\ldots,R_K}(G)$.



2
3
9
e_1
e_2
e_3
4
e_1: weight w_1
e_2: weight w_2
e_3: weight w_3
e_1: p_{e_1}
e_2: p_{e_2}
e_3: p_{e_3}
$\frac{1-\beta}{Z}$
$\frac{\beta w_1}{Z}$
$\frac{\beta w_2}{Z}$
$\frac{\beta w_3}{Z}$
1-β: belief from past confirmation
β: belief from recent history
(a) Edge Inference
(B, 2) 2
e_1
e_4
e_5
4
5
6
(C, 4)
(C, 4)
(B, 4)
color "Unknown": p_- (missing from B)
color "B": p_b (still at B)
color "C": p_c (moved to C)
$(1-\Upsilon)(1-\frac{1}{(\text{now - seen\_at})^{\Theta}})$
$\frac{(1-\Upsilon)}{(\text{now - seen\_at})^{\Theta}}$
$\Upsilon\frac{p_{e_5}}{\Sigma p_{e_i}}$
$\Upsilon\frac{p_{e_1}}{\Sigma p_{e_i}}$
$\Upsilon\frac{p_{e_4}}{\Sigma p_{e_i}}$
1−ϒ: belief gained from the node itself
ϒ: belief gained via containment
(b) Node Inference

Figure 6: Examples of edge inference and node inference.

So, the total cost becomes:

$$= \sum_k (cost_1(R_k) + cost_2(R_k)) + \sum_k cost_{3,4}(R_k)$$
$$\leq \sum_k (|R_k| + |R_k|^2) + (\sum_k |R_k|^2 + 2|\pi_{R_1,\ldots,R_K}(G)|)$$

This yields a complexity $O(\sum_k |R_k|^2 + |\pi_{R_1,\ldots,R_K}(G)|)$ for complete graph update in an epoch.

4 Data Interpretation

The graph constructed from the data capture step can result in nodes that are uncolored or possess multiple parent nodes. The data interpretation step estimates the most likely location of an unreported (uncolored) object and the most likely container (parent) of an (either reported or unreported) object. We present a probabilistic inference technique that includes edge inference to address ambiguous containment, node inference to address unknown locations, and an iterative algorithm that iteratively applies both to the entire graph in an alternating fashion.

4.1 Edge inference

Edge inference is applied to each incoming edge of a node v regardless of whether the node is colored or not. It assigns a probability value p_{e_i} to each edge; the edge with the highest probability value is then chosen as the most likely container of this object. Past history is used to compute probability values, which includes (i) the recent history of co-locations, as represented by the bit-vector *recent_collocations* and (ii) the last confirmation of an edge either by a special reader or as a result of inference with high certainty, as captured in the data structure *past_certain_parent*. The use of past history makes edge inference less sensitive to missed readings.

Probabilistic Framework. Edge inference at a node consists of two steps, as illustrated in Figure 6(a).

Step 1. *Assign weights:* The first step computes a weight w_{e_i} for each incoming edge using the bit-vector *recent_collocations*.

$$w_{e_i} = \frac{\sum_{i=0}^{S} \frac{recent\_collocations[i]}{i^{\alpha}}}{\sum_{i=0}^{S} \frac{1}{i^{\alpha}}} \quad (1)$$

where S is the size of the bit-vector and *recent_collocations*$[i]$ indexes into the i^{th} bit of this bit vector. This history is weighted using the parameter α and then normalized. α essentially implements a Zipf distribution, where $\alpha > 0$ assigns a higher weight to recent history, while $\alpha = 0$ weighs all prior co-locations equally.

Step 2. *Compute Probabilities:* The next step builds a probability distribution across all incoming edges. It computes a probability p_{e_i} for each edge by balancing the relative weight on this edge against the last confirmation of this edge. A parameter β is used to weigh these two factors. The probability p_{e_i} of the edge e_i is:

$$p_{e_i} = (1-\beta)m(e_i) + \beta w_{e_i} \quad (2)$$

p_{e_i} is then normalized across all incoming edges to yield the final distribution. The memory function m_{e_i} takes the value '1' if e_i is the last confirmed edge and '0' otherwise; recall that an edge may be confirmed by special readers such as belt readers or as a result of inference with high certainty. Since at most one parent edge of a node can be a confirmed edge, such an edge gains an extra weight and is favored over other possibilities until other edges gain sufficient history to outweigh it. Figure 6(a) shows such distribution across three parent edges, e_1, e_2, and e_3, with e_1 assigned the additional $1-\beta$ due to its past confirmation.

Edge inference involves several parameters: (1) S the size of the collocation history, (2) α, the zipf parameter, (3) β, the partition of beliefs between the recent history and the past confirmation, and (4) $m(e_i)$, the memory of the past confirmation of an edge e_i. Section 6 quantifies the sensitivity of the inference to these parameters.

4.2 Node inference

Node inference is applied to an uncolored node v—an object with an unknown location—and attempts to infer the most likely location of the object or confirm its absence from any known location. The key challenge in node inference arises from a three-way tradeoff among *object dynamics*, *continuation of past state*, and *absence with no other knowledge*. These situations are depicted in Figure 6(b) for node 2 at time t=4. This object was last seen in location B at time t=2 and has a few possibilities for its current location: it is still in location B but the reading in this location was missed (continuation of past state); it moved to location C with its contained objects and its reading was missed in C (object dynamics); it disappeared from B and its current location is unclear (absence with no other knowledge).

Probabilistic Framework. To account for all these possibilities, the node inference builds a probabilistic distribution over all possible colors of a node v, including (1) the most recent color of the node, (2) the colors of its neighboring nodes that can be propagated through the edges, and (3) a special color "unknown". Among all possible colors, the one with the highest probability represents the most likely estimate of this object's location.

The probability of the node v having color c is given as

$$p_c(v) = (1-\gamma)\frac{1}{(now - seen\_at)^{\theta}}\delta(v,c) + \gamma \sum_{e_i \to c} \frac{p_{e_i}}{Z} \quad (3)$$

$$\delta(v,c) = \begin{cases} \text{1, c is the most recent color of v} \\ \text{0, otherwise} \end{cases}, \; Z = \sum_{e_i \to c_j} p_{e_i},$$

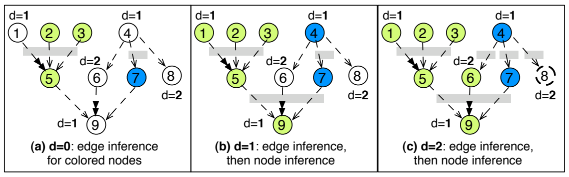

d=1
1
2
3
d=1
4
d=2
5
6
7
8
d=2
d=1
9
(a) d=0: edge inference for colored nodes
(b) d=1: edge inference, then node inference
(c) d=2: edge inference, then node inference

Figure 7: Iterative inference in the graph.

where $e_i \to c$ means that the edge e_i propagates the color c to v, $\delta(v, c)$ is 1 for the most recent color of v and 0 otherwise, and Z is the normalization factor across all edges of v that propagate colors to v. Parameter γ weighs colors that originate from the node against colors that propagate through the edges. Parameter θ controls the rate of fading for a color. The probability of the special color "unknown" is:

$$p_{unknown}(v) = (1-\gamma)(1 - \frac{1}{(now - seen\_at)^{\theta}}) \quad (4)$$

Figure 6(b) shows the resulting probability distribution over three colors, B, C, and "unknown".

Node inference is influenced by parameters (1) γ which weighs the previous node colors against colors propagating from edges; and (2) θ exponent of the function $W_{fading} = (now - seen\_at)^{-\theta}$ that controls the probability distribution between the fading color and the "unknown" color.

4.3 Iterative Inference

Iterative inference combines node and edge inference to iteratively cover the entire graph $G(V, E)$ and derive the most likely location and containment for each object. Traditional graph traversal algorithms such as breadth-first and depth-first search can not be applied here due to the dependency between edge and node inference. Specifically, node inference at an uncolored node involves the colors of its neighboring nodes and the probabilities of the edges of those nodes, and can not begin until these dependencies are first resolved.

The key idea of our inference algorithm is to start inference from the colored nodes—the nodes with known locations—and run it iteratively across the graph, through the edges linked to the colored nodes, to the uncolored nodes incident to these edges, to the edges linked to these nodes, and so on. In this way, inference sweeps through regions of the graph in increasing distance from the colored nodes; the colors and edge probabilities determined at nodes in a shorter distance can contribute to the inference at nodes in a larger distance.

For ease of composition, we classify nodes based on their closest distance, d, from a colored node in the graph. The iterative inference algorithm runs in increasing value of d, as illustrated in Figure 7 (we omit detailed pseudo code due to space constraints). The algorithm first runs edge inference on all incoming edges into nodes labeled $d = 0$. These nodes represent observed objects and are colored in two colors, dark and light, in this example as shown in Figure 7(a). Only edge inference for containment is made for them. After that, the algorithm runs edge inference, followed by node inference on nodes labeled $d = 1$, as shown in Figure 7(b). In this example, node 9 gains the light color and node 4 gains the dark color from their node inference. Finally, the algorithms runs for nodes labeled $d = 2$, shown in Figure 7(c). At this point, node 6 gains the light color and node 8 is identified as a missing object. Note that node 4 and node 6 now have different colors. However, since both the colors are inferred, i.e. uncertain, the edge between them is preserved for use in future inference.

Complexity analysis. Since each node of the graph is visited only once in the iterative inference algorithm, it is not hard to see that the complexity of inference is bounded by the number of edges examined. Given that each edge can be visited at most twice, once from each node, the overall complexity is $O(|E|)$.

5 Stream Output with Compression

The output module takes the results of data interpretation and transforms them into an compressed output event stream. The key idea behind compression is that only those readings that indicate a *state change* need to be included in the output stream. The state of an object is said to have changed if its location or its containment changes. In the absence of a state change, all readings merely confirm the current state of the physical world and are redundant (and can be safely discarded). This idea leads to two compression techniques, *location compression* for stationary objects, and *containment compression* for stationary and mobile objects alike. Note that such compression is a superset of per-tag smoothing [19, 20], which is restricted to a single location and based on past observations of a tag in that location.

The output algorithm takes the results of inference, i.e. the most likely estimates of the location and container of each object, and generates output events in a compressed format. If an object is stationary—resident at the same location for a period of time—only an initial event needs to be output to indicate its first arrival at this location and all subsequent readings in the same location can be safely discarded. This compression technique is called *location compression* and works for stationary objects. Separately, if the containment between two objects has not changed for a period of time, only one event needs to be output to indicate the start of this relationship. The compression technique, called *containment compression*, exploits stable containment between objects and works for both stationary and mobile objects.

The compressed output can comprise five event types. For location update, there are three types of events, *startLocation*(object, location, startTime), *endLocation*(object, location, endTime), and *missing*(object, location, time). Start and end location events always occur in pairs. Missing events are singletons appearing outside any start-end location pair. For containment updates, we use events *startContainment*(object, container, startTime) and *endContainment*(object, container, endTime); these events always occur in pairs.

The compression algorithm proceeds in two steps after each inference.

Step 1. *Containment compression:* If the current inferred container c of an object differs from the previously reported container c', the previous containment is ended using an *e*ndContainment event and a new containment is begun using *s*tartContainment.

Step 2. *Location compression:* If the currently inferred location l differs from the previous location l' then there are three scenarios to consider. (i) both l and l' are known locations: The algorithm checks if l is an inferred location and further if it conflicts with an existing containment that was output or a new containment to be output. If l passes the conflict check, its previous location is ended by an *endLocation* event, and a new location is started with a *startLocation* event. (ii) l' is a known location and l is unknown: the previous location is ended and a missing event is generated. (iii) l' is unknown and l is a known location. Simply a new location is started.

The events in the compressed format encode lasting phenomena and such streams can be fed into recently developed event processors [1] for further processing.

Observe that the above algorithm allows location update events and containment update starts to be nested in flexible ways. For an object, a start-end containment pair can span multiple start-end location pairs, representing an unchanged containment as the two objects move together through various locations. In addition, when an object is reported missing, the existing containment is not ended. That is, a start-end containment pair can also enclose the missing events. On the other hand, it is also possible that a start-end location pair covers multiple start-end containment pairs, capturing the containment changes in the same location. Finally, potential conflicts arise when the two objects participating in an existing containment have different locations to output (as illustrated in Figure 7). In this scenario, we know that at least one of the objects has an inferred location—if both locations were actually reported by RFID readings, the data capture and interpretation algorithms guarantee that the existing containment would end and be replaced by a new one. To resolve potential conflicts in output, we give more weight to the containment inference and use the existing containment to suppress the output of an inferred location that is causing a conflict.

6 Performance Evaluation

We have implemented a prototype of our interpretation and compression substrate in Java. In this section, we evaluate the accuracy and efficiency of the interpretation techniques under a variety of workloads in a simulated enterprise supply-chain environment. We also explore the benefits of compression based on results of interpretation.

6.1 Experimental Setup

To generate synthetic RFID streams, we developed a simulator that emulates deployments of RFID readers in large distribution centers. For a given distribution center, pallets arrive at a certain rate, are read at the entry door (reader group 1), and then become unpacked. Cases are scanned on the receiving belt (reader group 2), placed onto shelves for a period of stay (reader group 3), and then repackaged (reader group 4). The newly assembled pallets are rescanned on the belt (reader group 5) and finally exit the distribution center (reader group 6). Synthetic RFID stream traces from this environment are then fed to our interpretation and compression substrate.

Parameter	**Value(s) used**
Duration of Simulation	3 - 24 hours
Number of Warehouses	1
Rate of Pallet Injection	3 - 30 per hour
Cases Per Pallet	5
Items Per Case	20
Read Rate of Readers	0.5 - 1
Non-Shelf Reader Frequency	2 (interrogations) / sec
Shelf Reader Frequency	1/sec, 1/10 sec, 1/30 sec, 1/min, 1/10 min, 1/30 min
Avg. Duration of Shelving	1 - 10 hours
Frequency of Anomaly	1 / 100 sec, if turned on

Table 1: Parameters used for generating RFID streams.

The workload parameters for generating synthetic RFID stream shown in Table 1. Two separate parameters are used to control the read frequency of shelf and non-shelf readers. This design allows flexible settings of the simulation where items stay on shelves for hours and shelf readers can be configured to read less frequently than other readers.

The default epoch length is 1 second. Data interpretation is performed after every epoch. To deal with unsynchronized readers, given each reader active in this epoch, we run inference for the objects that are either reported by the reader or believed to be in the reader's location in the last inference. We treat door readings as the warm-up for graph construction and do not run inference for this location.

6.2 Accuracy of Data Interpretation

In the first set of experiments, we evaluate the accuracy of our inference techniques used in interpretation. We created data traces with 6 new pallets injected per hour, an average shelving period of 1 hour and a total simulation time of 3 hours. The metric used in most experiments is *inference error rate*, which is the percentage of the inference results, including both location and containment estimates, that are inconsistent with the ground truth.

Expt 1: Edge Inference. We first study the effects of the edge inference parameters, β, S, and α, shown in Equations 1 and 2, on containment inference. Our results show that the two parameters on the recent history of collocations can be tuned easily. The size of the history, S, limits the inference accuracy when it is small, e.g. 4, 8, and offers no additional benefit after the point of 32. The zipf parameter, α, yields best accuracy when set to 0, indicating that recent instances of collocations are equally important to inference. Hence, we use S=32 and α=0 in the rest of experiments.

The parameter β governs the beliefs between the recent history, which can be noisy, and the past confirmation, which may be obsolete. In our simulation, the major source of noise in containment inference is the collocation of cases

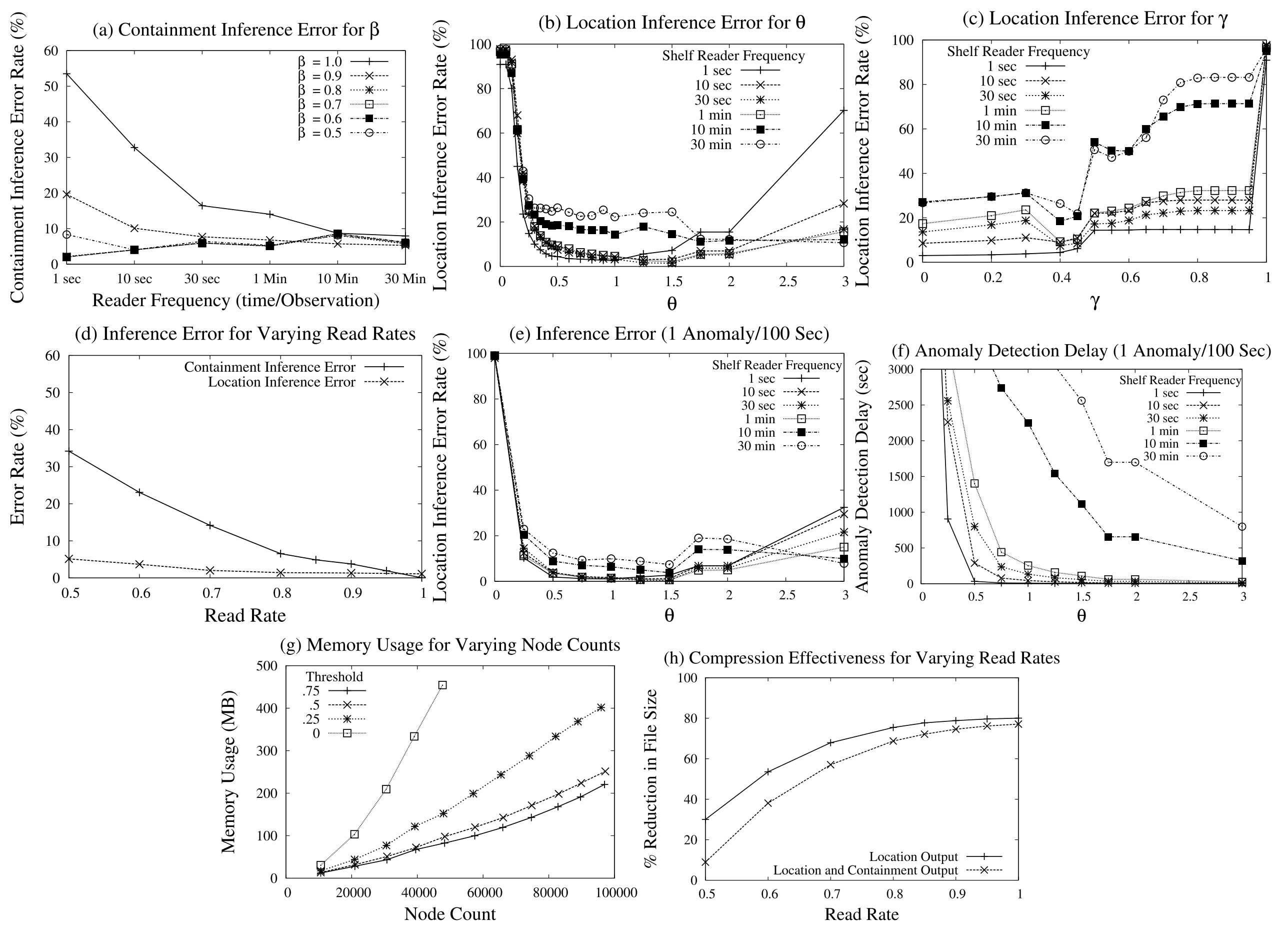


Figure 8: Experimental Results

on shelves for an extended period of time. To capture such noise, we generated traces with different shelf reader frequencies, and for each trace, varied β from 0.5 to 1. β=1 gives all the weight to recent history, and β=0.5 results in an estimate always based on past confirmation. As shown in Figure 8(a), when the noise is high, e.g. the shelf reader frequency is 1 per second, high β values give significantly worse accuracy due to its emphasis on recent history; low β values are much more accurate. As the noise reduces, i.e., the recent history becomes more useful, the accuracy using high β values improves while the accuracy of lower β value deteriorates, resulting in a slightly reversed trend for very infrequent shelf readings.

Expt 2: Node Inference. Location inference uses the method defined in Equations 3 and 4. We varied the two parameters θ and γ to study their effects on location inference.

θ is the dampening factor on the belief of the continued existence of an unobserved object in its last reported location. High θ values cause more quickly reduced belief. Figure 8(b) shows the results. As θ increases, the error rate quickly declines from nearly 100%, flattens in the mid-range values, and degrades again with higher values. The initial decline occurs because with very low θ values, the inference takes too long to reduce its belief of the continued presence of an object, causing many errors in inference. The deterioration with high θ values occurs because the inference becomes too eager to drop its belief and identify an object as missing after a few missed readings. Similar trends are observed for different read frequencies with the effect more pronounced for high frequency readers.

γ weighs the belief of an object's last observed location (with low γ values) against the belief of its location inferred via containment (with high γ values). Figure 8(c) shows the results for varied γ values. For this particular workload, low γ values work well due to the fact that inference is incurred mostly by the shelf readings and they represent stationary objects. Around the point of 0.5, the inference error rate increases quickly due to the shifting of belief to rely more on containment, which is a less reliable source for this workload. With all emphasis given to containment, most inference results are wrong. Again, similar trends are observed for different reader frequencies.

The above experiments provide insights into tuning inference parameters for common workloads in our target domain. In the rest of performance study, we use the following setting unless stated otherwise: β = .7, γ = .4, and θ = 1.25.

Expt 3: Sensitivity to Read Rate. The next experiment studies the sensitivity of our inference methods to the read rate. The shelf reader frequency was set to 1 reading per minute. As Figure 8(d) shows, the error rates of both containment and location inference stay below 10% for read rates between .8 and 1, a common range in many reported deployments. As the read rate decreases, the location inference stays fairly accurate due to its appropriate parameter settings to exploit the last reported location. The containment inference, however, loses its accuracy due to both the loss of containment confirmation provided by belt readers and lack of consistent observations in the recent history.

Expt 4: Accuracy and Delay of Anomaly Detection. The traces used so far have not captured any abnormal behaviors, which are expected to be rare but of significant interest to the application. In this experiment, we simulated unexpected removals of objects from the warehouse, representing theft or misplacement, at a rate of 1 removal every 100 seconds with random selection from all objects. We report on the inference error rate as well as the delay of anomaly detection in Figure 8(e) and 8(f) as θ is varied.

Regarding the error rate, Figure 8(e) exhibits similar trends as Figure 8(b) and confirms that the θ values between 0.5 and 1 also work well for anomaly detection. Figure 8(f), however, shows a different trend regarding the delay of anomaly detection. For a shorter delay, higher values of θ are preferred to more quickly decay the belief of the continued presence of an object. This is especially true for low reader frequencies, where it otherwise takes too long to wait for the next reading, adjust the belief, and finally recognize the missing object. These results show an inherent tension between the optimal setting for short delay of anomaly detection and that for the overall location inference accuracy.

6.3 Efficiency of Data Interpretation

We next evaluate the efficiency of data interpretation in both memory usage and processing speed. To do so, we created large data traces with higher pallet injection rates, ranging from 1 per 85 seconds to 1 per 15 seconds, to simulate a very large distribution center. The tests were performed on a linux server with Intel Quad Core 2.33GHz Xeon CPU and 8GB memory running JVM 1.6.0. The maximum Java allocation pool size was set to 2GB. Memory usage was calculated using a Java utility package.

Expt 5: Processing speed To measure the processing speed, we used three injection rates, 1 per 85 seconds, 1 per 42 seconds, and 1 per 28 seconds, and warmed up the system until the number of objects in the system became stable. Table 2 reports the number of objects at these steady states in the first column. It shows the cost of graph update for all active readers and the cost of inference over the graph for each epoch (i.e. a second) in the second and third columns. The total cost is reported in the last column. As can be seen, the costs of both update and inference are small. The total cost is 31 msec for the case of the smallest injection rate, which is already high for most distribution centers, and reaches half a second for the case of the largest rate. These results show that our data interpretation techniques can keep up with high-volume RFID streams.

Expt 6: Memory Usage. The memory usage in data interpretation is dominated by the size of the graph. To capture the growth of the graph, we further increased the injection rate to 1 pallet per 15 seconds, and measured the memory usage as the number of nodes in the graph increased. We also observed that the graph size can be reduced by pruning edges for which the containment inference yields very low confidence. The confidence value here is the value in Formula 2 but before normalization and thus is insensitive to the presence of other edges. To explore this factor, we applied a threshold for pruning edges and varied it from 0 to .75.

Num. Objects	Update	Inference	Total
25547	0.02178	0.00934	.03112
50327	0.06655	0.05225	.11880
95455	0.22429	0.29507	.51936

Table 2: Costs of Update and Inference Operations (sec)

Figure 8(g) shows the results. As the node count grows, the memory usage with no pruning increases very fast and less so with increased values of the threshold. After the point of .5, the threshold has a limited effect on memory usage. With such decent pruning, the memory usage was kept under 300 MB despite the large number of pallets injected and objects present in the center. Further, these curves increase close to linearly, rather than the worse case of edge expansion quadratic in the number of nodes. Finally, we note that the pruned edges have a small effect on the inference error rate, with the best accuracy achieved with the pruning threshold at .5. Details are omitted in the interest of space.

6.4 Compression

Expt 7: Compression Ratio. The last experiment explores the benefit of compression based on results of interpretation. Figure 8(h) presents the compression ratio for the small traces when the read rate was varied. The ratio was computed by comparing the size of our output to that of the original data trace. We report on two ratios, one considering only the location output, and the other including also the containment output which is the additional information provided by interpretation. As can be seen, the overall compression ratio is good with commonly observed read rates between .8 and 1, even with the containment output. As the read rate decreases, the ratio degrades, exhibiting its sensitivity to this factor. Similar trends are observed for larger traces.

Summary of results. Overall, our results show that the our framework for inference is flexible enough to capture a variety of workloads in an RFID-based supply chain environment. Accuracy of the inference is sensitive to the choice of various parameters. Further, the optimal values of these parameters are dependent on the workload. However, when chosen carefully, the error rate of the inference is small ($< 5\%$). Our results also show that our interpretation technique easily scales to high stream volumes seen in real-world warehouse environments and that the memory overhead of our technique is under 300MB when graph pruning is used. Finally, our compression technique yields 75-80% reduction in volume for commonly observed read rates.

7 Related Work

Data cleaning has traditionally been the purview of data warehousing. In this context, it addresses a small set of well-defined tasks, such as rule-based transformation, and uses offline processing techniques [2]. The nature of RFID data and its stream-based processing precludes the use of traditional data cleaning techniques.

RFID stream processing. The most relevant work to our own is HiFi [13, 19, 20], a declarative framework for RFID data cleaning and processing. Its techniques focus on per-tag smoothing or multi-tag aggregation on different temporal and geographical scales, but do not capture relationships between objects or estimate object locations. Probabilistic RFID processing has been recently proposed for a more general solution [15] but the research is still underway.

RFID warehouses. Real-world examples are offered in [3] to motivate RFID management problems including inference. Location and containment compression techniques are proposed in [16] for RFID warehouses but use expensive disk-based operations such as sorting and summarization. Siemens RFID middleware [30] uses application-specific rules to compress and archive data into databases. These techniques are unsuitable for fast low-level compression of RFID streams and further neglect the interpretation problem.

Sensor data management has been an area of intensive recent research [31, 24, 7, 8, 27, 28]. The sensor data considered captures environmental phenomena such as temperature and light, and the proposed techniques, such as data acquisition [24, 7, 6], approximation [5], and sampling [32], are all geared towards queries natural to such data, e.g. selection and aggregation, and energy-efficient in-network processing. In contrast, RFID data captures object identification and its processing raises challenges related to locationing and correlation of objects and data volume reduction.

Machine Learning. Interpretation of RFID streams is related to probabilistic inference over time [26]. When applied to RFID streams, the state-of-the-art inference techniques such as Dynamic Bayesian Networks [22, 21, 25] have significant drawbacks including large state spaces, expensive slow parameter learning, and limited efficiency of online inference. SPIRE is designed for stream-based inference by employing a memory-efficient graph model and an inference framework with a few tunable parameters.

8 Conclusions

In this paper, we presented a novel data interpretation and compression substrate over RFID streams to address the challenges of incomplete data, insufficient information and high volumes. Our substrate employs a time-varying graph model to capture inter-object relationships such as co-location and containment. It then employs a probabilistic inference algorithm to determine the most likely location and containment for each object and an efficient stream compression algorithm to remove redundant information from the output stream. We implemented a prototype of our interpretation and compression substrate and evaluated it using synthetic RFID streams that emulate a warehouse and supply-chain environment. Our results show that our inference techniques provide good accuracy while retaining efficiency, and our compression algorithm yields significant reduction in data volume. For future work, we plan to enhance our interpretation and compression substrate to infer location and containment information for mobile readers.

References

[1] Roger S. Barga, Jonathan Goldstein, et al.. Consistent Streaming Through Time: A Vision for Event Stream Processing. In *CIDR*, pages 363–374, 2007.

[2] Surajit Chaudhuri and Umeshwar Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, 1997.

[3] Sudarshan S. Chawathe, Venkat Krishnamurthy, et al. Managing RFID Data. In *VLDB*, pages 1189–1195, 2004.

[4] R. Cocci, T. Tran, Y. Diao, and P. Shenoy. Efficient Data Interpretation and Compression over RFID Streams. Tech Report, UMass Amherst, 2007. `http://www.cs.umass.edu/~yanlei/spire.pdf`.

[5] Jeffrey Considine, Feifei Li, et al. Approximate Aggregation Techniques for Sensor Databases. In *ICDE*, pages 449–460, 2004.

[6] Amol Deshpande, Carlos Guestrin, et al. Exploiting Correlated Attributes in Acquisitional Query Processing. In *ICDE*, 143–154, 2005.

[7] Amol Deshpande, Carlos Guestrin, et al. Model-Driven Data Acquisition in Sensor Networks. In *VLDB*, pages 588–599, 2004.

[8] Amol Deshpande and Samuel Madden. MauveDB: supporting model-based user views in database systems. In *SIGMOD*, 73–84, 2006.

[9] EPCglobal Tag Data Standards Version 1.3. `http://www.epcglobalinc.org/`, Mar 2006.

[10] B.J. Feder. Despite Wal-Mart's edict, radio tags will take time. `http://www.epcglobalinc.org/`, Dec 2004.

[11] Klaus Finkenzeller. *RFID handbook: radio frequency identification fundamentals and applications*. John Wiley and Sons, 1999.

[12] Christian Floerkemeier and Matthias Lampe. Issues with RFID Usage in Ubiquitous Computing Applications. In *Pervasive*, 188–193, 2004.

[13] Michael J. Franklin, Shawn R. Jeffery, et al. Design Considerations for High Fan-In Systems: The HiFi Approach. In *CIDR*, pages 290–304, 2005.

[14] S. Garfinkel and B. Rosenberg, editors. *RFID: Applications, Security, and Privacy*. Addison-Wesley, 2005.

[15] Minos N. Garofalakis, Kurt P. Brown, et al. Probabilistic Data Management for Pervasive Computing: The Data Furnace Project. *IEEE Data Eng. Bull.*, 29(1):57–63, 2006.

[16] Hector Gonzalez, Jiawei Han, et al. Warehousing and Analyzing Massive RFID Data Sets. In *ICDE*, page 83, 2006.

[17] P. Harrop and G. Holland. RFID for postal and courier services. `http://www.idtechex.com/pdfs/en/R2646Q5829.pdf`, Nov 2005.

[18] Annika Hinze. Efficient Filtering of Composite Events. In *BNCOD*, pages 207–225, 2003.

[19] Shawn R. Jeffery, Gustavo Alonso, et al. Declarative Support for Sensor Data Cleaning. In *Pervasive*, pages 83–100, 2006.

[20] Shawn R. Jeffery, Minos N. Garofalakis, and Michael J. Franklin. Adaptive Cleaning for RFID Data Streams. In *VLDB*, 163–174, 2006.

[21] Michael I. Jordan, editor. *Learning in graphical models*. MIT Press, Cambridge, MA, USA, 1999.

[22] Michael I. Jordan, Zoubin Ghahramani, et al. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37(2):183–233, 1999.

[23] Samuel Madden, Michael J. Franklin, et al. TAG: a Tiny AGgregation service for Ad-Hoc sensor networks. In *OSDI*, pages 131–146, 2002.

[24] Samuel Madden, Michael J. Franklin, et al. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, pages 491–502, 2003.

[25] Kevin Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning.* Ph.d. thesis, University of California, Berkeley, July 2002.

[26] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2003.

[27] Adam Silberstein, Rebecca Braynard, and Jun Yang. Constraint chaining: on energy-efficient continuous monitoring in sensor networks. In *SIGMOD*, pages 157–168, 2006.

[28] Adam Silberstein, Kamesh Munagala, and Jun Yang. Energy-efficient monitoring of extreme values in sensor networks. In *SIGMOD*, pages 169–180, 2006.

[29] Bob Violino. RFID opportunities and challenges. `http://www.rfidjournal.com/article/articleview/537`.

[30] Fusheng Wang and Peiya Liu. Temporal Management of RFID Data. In *VLDB*, pages 1128–1139, 2005.

[31] Yong Yao and Johannes Gehrke. Query Processing in Sensor Networks. In *CIDR*, 2003.

[32] Yongzhen Zhuang, Lei Chen, et al. A Weighted Moving Average-based Approach for Cleaning Sensor Data. In *International Conference on Distributed Computing Systems (ICDCS)*, 2007.