SPIRE: Efficient Data Inference and Compression over RFID Streams

Yanming Nie‡, Richard Cocci†, Zhao Cao†, Yanlei Diao†, and Prashant Shenoy†
†Department of Computer Science, University of Massachusetts Amherst, U.S.A.
‡School of Computer Science, Northwestern Polytechnical University, China

***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 inference and compression substrate over RFID streams to address these challenges. Our substrate employs a time-varying graph model to efficiently capture possible object locations and inter-object relationships such as containment from raw RFID streams. It then employs a probabilistic algorithm to estimate the most likely location and containment for each object. By performing such online inference, it enables online compression that recognizes and removes redundant information from the output stream of this substrate. We have implemented a prototype of our inference and compression substrate and evaluated it using both real traces from a laboratory warehouse setup and synthetic traces emulating enterprise supply chains. Results of a detailed performance study show that our data inference techniques provide high accuracy while retaining efficiency over RFID data streams, and our compression algorithm yields significant reduction in output data volume.**

***Index Terms*—RFID, data streams, data cleaning, compression, supply-chain management**

I. 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, healthcare, pharmaceuticals [1], postal services and surveillance 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 [2], [3], [4] existing research has mostly focused on sensor data that captures continuous environmental phenomena. RFID data—a triplet <tag id, reader id, timestamp> in its 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, an RFID stream does not directly reveal whether flammable objects are secured in a fire-proof container, or foods with and without peanuts are not packaged in the same container, even though all items and containers are affixed with RFID tags.

Incomplete data: Despite technological advances, RFID readings are inherently noisy with observed read rates significantly below 100% in actual deployments [5], [6]. This is largely due to the intrinsic sensitivity of radio frequencies (RFs) to environmental factors such as occluding metal objects [7] and contention among tags [8]. Missed readings result in lack of information about an object's location, significantly complicating the tasks of determining object location and containment and detecting anomalies such as missing objects.

High volume streams: RFID readers are often configured to read frequently when they are deployed in wired, powered environments. Large deployments of such readers can create excessively large volumes of data, e.g., over terabytes of data in a single day [9]. The resulting data, however, may encode significant amounts of redundant information such as an unchanged object location. Hence, it is crucial that data be filtered and compressed close to the hardware while preserving all useful information.

Recent research on RFID data cleaning [10], [6], [11] has employed smoothing techniques to clean individual tag streams and estimate tag counts in a given location in the presence of missed readings. These techniques, however, do not capture inter-object relationships such as containment or identify anomalies such as missing objects. Recent research on probabilistic query processing [12], [13] has not focused on the derivation of information mentioned above, such as containment or missing objects, but its query processing can be enriched once such information is made available as input. Furthermore, none of the above work has addressed the data compression problem. Compression techniques for RFID warehouses use expensive disk-based operations such as sorting and summarization [14] or employ application-specific logic [15]. Hence, they are unsuitable for fast online compression of RFID streams close to the hardware.

In this paper, we present SPIRE, a system that addresses the above three challenges by building an ***inference and compression substrate*** over RFID data streams. This substrate enables accurate inference of observed data, even though the raw data is incomplete. Further, it infers inter-object relationships such as co-location and containment as well as anomalies such as missing objects. Finally, by performing online inference, it enables online compression that discards redundant data such as an unchanged object location or an unchanged containment between objects. Online compression

A preliminary version of this work appeared as a 3-page poster paper at ICDE 2008 significantly reduces data volume, thereby expediting query processing and reducing transfer costs in distributed systems.

The SPIRE system employs three key techniques, which are also the main contributions of this paper:

- We propose a time-varying graph model that captures possible object locations and containment relationships with its efficient construction from raw RFID streams.
- We further develop an online probabilistic algorithm that estimates the most-likely locations of objects and containment relationships among objects (which subsume co-location relationships) from the information captured in the graph model.
- We finally devise an online compression algorithm that transforms an input raw RFID stream into a compressed yet richer output event stream with both location and containment information.

We have implemented our inference and compression substrate in a prototype system and evaluated it using both real traces from a laboratory warehouse setup and synthetic traces emulating enterprise supply chains. Our results show that our data inference techniques achieve error rates below 15% for location estimates for a wide range of RFID read rates, and within 20% for containment estimates when the read rate reaches 80%. In addition, these techniques can be performed efficiently on high-volume RFID streams. Furthermore, our compression techniques can encode rich location and containment information using only 20% or less of the raw input data size when the read rate reaches 80%. Finally, we compare our system with SMURF [11], a state of the art system for RFID data cleaning, that can be used to produce object location information but not containment information. For object location updates, our system outperforms SMURF in both the error rate and the resulting compression ratio.

The rest of the paper is organized as follows. Section II formulates the problem. Sections III, IV, and V describe the three key techniques of our system. Section VI presents results of a detailed performance study. Finally, Section VII presents related work, and Section VIII concludes the paper.

II. 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 objects o_i and o_j are both 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, exits the world through a designated channel (e.g., an exit door), or 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. Such readings, however, are inadequate for capturing the containment between objects. Furthermore, the observation of the world may be incomplete since some objects may not respond to reader queries due to technological limitations. As a result, both the location and containment of an unobserved object becomes unclear.

The data inference 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 for given arguments, return probabilistic values representing the likelihood of the function being true. Then the data inference problem can be formulated as: given the time now and an object o_i, report

1) the most likely location of the object, denoted by $\arg\max_k\ resides(o_i, l_k, now)$, and
2) the most likely container of the object, denoted by $\arg\max_{j,k}\ contained(o_i, o_j, l_k, now)$.

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 inference to obtain such knowledge. The combination of inference 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 Fig. 1, where RFID readers are installed above the loading dock, the conveyor belt, and two shelves. Each arriving pallet is scanned at the loading lock, together with the cases on the pallet and items in the cases. An RFID tag is attached to every pallet, case, and item. In this example, at time t=1, the reader at the loading dock reports objects 1 to 6, denoted by the black nodes. These nodes are arranged according to the packaging levels that the reported tag ids indicate [16]. Object 7 is also present but was missed by the reader, denoted by a white 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 containers of items 4, 5, 6, which can be either case 2 or case 3 based on the readings received.

After time t=1, the pallet stays at the locking dock while the cases transition to other places for scanning. At time t=2, case 2 is scanned individually on the conveyor belt. It is possible

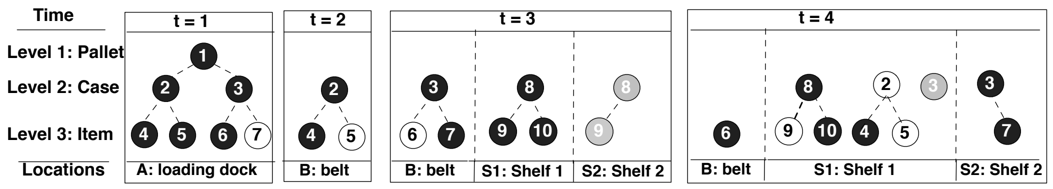


Fig. 1. Example readings of RFID-tagged objects in a warehouse. Nodes in black, gray, and white represent objects read in ithe true location, objects read by a nearby reader, and unobserved objects, respectively. A dashed edge denotes a containment relationship between objects, which cannot be directly observed.

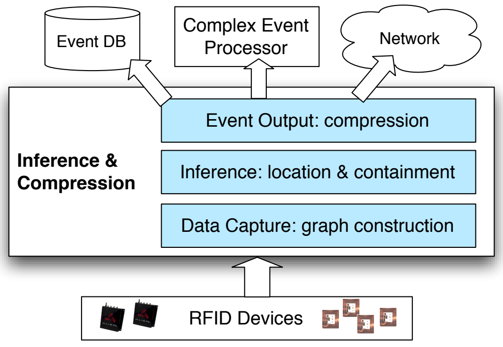

Event DB
Complex Event Processor
Network
Event Output: compression
Inference & Compression
Inference: location & containment
Data Capture: graph construction
RFID Devices

Fig. 2. Architecture of the SPIRE system.

to confirm the containment between the case and its item(s) if we know for sure that the belt reader scans cases one at a time. At t=3, case 3 is scanned on the belt with its contained items. A new case 8 with items 9 and 10 are read at shelf 1, which is their true location. In addition, case 8 and item 9 are also read at shelf 2 by a nearby reader, which are called *duplicate readings*.

At t=4, item 6 is read at the belt again (it fell off its case at t=3 and stayed here). Case 2 was placed onto shelf 1. However, case 2 and item 5 are missed by the reader at shelf 1. Case 3 was placed onto shelf 2 instead. It then receives both a reading from shelf 2 and a duplicate one from shelf 1. Finally, case 8 is read at shelf 1 again but the reading of item 9 is missed.

System architecture. The SPIRE system employs a data inference and compression substrate to address the above issues. The substrate, epicted in Fig. 2, consists of (i) a *data capture* module that implements a stream-driven construction of a time-varying graph model to encode possible object locations and containments, (ii) an *inference* module that employs a probabilistic algorithm to estimate the most likely location and containment for an object, and (iii) a *compression* module that outputs stream data in an compressed format. The next three sections describe these techniques in detail.

III. Data Capture

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

A. 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 Fig. 1 are shown in Fig. 3.

The node set V denotes all RFID-tagged objects in a physical world. In a supply-chain environment, the RFID standard [16] requires that an object have a packaging level of an *item*, *case*, or a *pallet*, and this information be encoded in the object's tag ID. Given such information, our graph is arranged into layers, with one layer for each packaging level. In addition, each node has a set of colors that denote its observed locations, with one color for each observed location. The node colors are updated using the stream of readings in each epoch (each color assigned to a node represents the location where it is observed by an RFID reader). If an object is not read by any reader in a given epoch, its node becomes uncolored. However, every node retains memory of its recent colors in the last k observations (k=1 as depicted in Fig. 3, but k=3 actually used in this work), denoted by a list $\{(\texttt{recent color}, \texttt{seen at})\}$. If the node obtained the same color several times in the last k observations, the most recent observation is used to set the `seen at` attribute.

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 inference method will subsequently choose only one of these possibilities). More generally, edges can exist between different combinations of colored and uncolored nodes, except that an edge cannot connect two colored nodes for which all available colors indicate a distance between the two objects beyond the read range of an RFID reader (e.g., over 40 feet)—in such cases, there cannot exist a containment relationship between the two objects. As such, our colored graph can capture a wide variety of containment relationships.

To enable probabilistic analysis, our graph also encodes rich statistics. Each edge maintains a bit-vector `recent co-locations` to record recent positive and negative evidence for the co-location of the two objects. A bit is set every time the two nodes connected by an edge are assigned the same color, i.e., the two objects are both observed in the same location. Furthermore, each node records the `confirmed parent`, i.e., the most recent confirmed container as a result of a highly likely estimate, the time of confirmation, and the number of conflicting observations obtained thus far. Among all incoming edges to a node, at most one edge can be chosen as the confirmed edge (which we detail in Section 4).

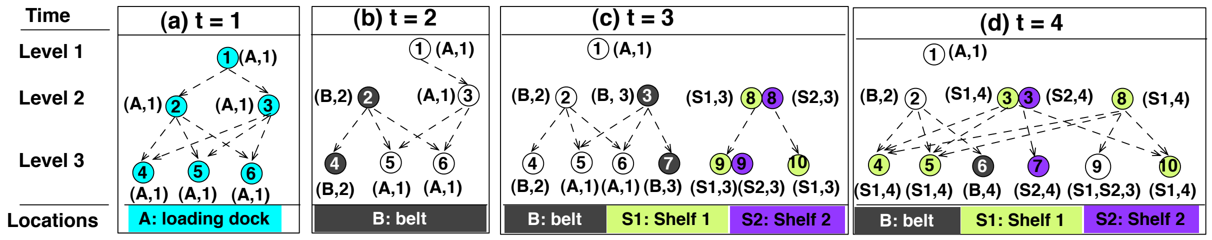


Fig. 3. Evolution of the time-varying colored graph model as RFID readings arrive in each epoch.

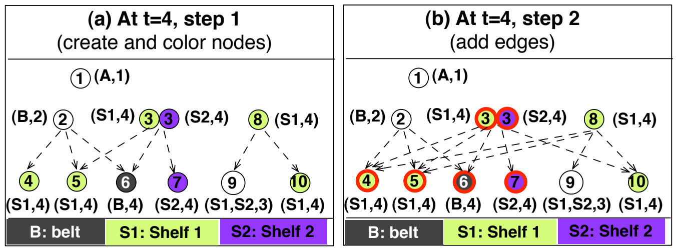


Fig. 4. Intermediate steps of the graph update procedure.

B. 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 takes the graph G from the previous epoch and a set of readings R_k from each reader k in the current epoch, and produces a new graph G^*. The graph update procedure proceeds in four steps (the detailed pseudo-code is provided in our technical report [17]).

Step 1. *Create and color nodes:* If a new object is observed for the first time, a new node is created in the graph. For each observed object, the color of the location in which it was observed is added to the color set of the corresponding node. Fig. 4(a) shows the result of this step when it is applied at time t=4 to the previous graph (Fig. 3(c)), using the readings from the conveyor belt, shelf 1, and shelf 2.

Step 2. *Add edges:* Next, if two nodes in adjacent layers share a common color (e.g., nodes 3 and 4 share the color for Shelf 1) an edge is added between them if it does not already exist. Doing so enumerates all possible containment relationships (e.g., an item observed at shelf 1 can be contained in any of the cases that are also observed at shelf 1). This step may require each node in a layer to be compared with all the nodes in the adjacent layers. An optimization for this step is to restrict such comparisons with adjacent layers only to the nodes that have just been assigned a new color. This is because if neither node of an edge is assigned a new color, then both objects are either in original locations or unobserved, offering no information for establishing a new containment relationship. Fig. 4(b) illustrates the result of this step of adding edges to the graph in Fig. 4(a). The bold circles represent the nodes that have changed their colors at t=4. Hence, new edges are created only for these nodes, e.g., between nodes 3 and 4, 3 and 10, 8 and 4, and 8 and 5.

Step 3. *Remove edges:* While the previous step adds new edges to the graph, in this step we remove outdated edges from the graph. An edge is removed if both nodes of the edge are colored and every possible pair of colors of these nodes indicates a large distance between the locations of the corresponding objects (i.e., beyond the read range of a single reader). In our example, assume that the readers for the belt and shelf 1 are far away from each other. Then the edge between nodes 3 and 6 in Fig. 4(b) is removed because the colors of these nodes indicate the belt and shelf 1, respectively. The resulting graph is in Fig. 3(d). Additional edges can be pruned if we can confirm the parent edge (container) of a node, hence eliminating other possibilities. We discuss the inference method that enables such edge pruning in the next section.

Step 4. *Update edge statistics:* 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 share a common color, `recent co-locations` of e is updated by setting the most recent bit to True. If one of the linked nodes is uncolored, the most recent bit is set to False. In this case, we also check if e was set before as the confirmed parent edge of the child node, and if so count the current epoch as a conflicting observation of the confirmation. These statistics play a key role in containment inference as we shall show shortly.

Complexity analysis. We finally analyze the complexity of the graph update procedure. The total cost of updating a graph $G(V, E)$ using reading sets $R_1, \ldots, R_K$ is the sum of the individual update costs for R_k, $1 \leq k \leq K$. For each reading set R_k, only the colored nodes and their incident edges are processed. The cost analysis is as follows.

Step 1: Given the reading set R_k, the cost of coloring nodes is simply $|R_k|$. Steps 2, 3, and 4: After processing all reading sets, $R_1, \cdots, R_K$, the next three steps share the process of examining every edge linked to a colored node. Two observations hold: First, consider those edges for which the two linked nodes share a common color. Given a reading set R_k, the maximum number of edges that may have both nodes colored by R_k is the size of the largest bipartite graph covered by R_k, that is, $(|R_k|/2)^2$. Other edges either have different colors assigned to the two linked nodes or have one node colored but the other uncolored. These edges must have already existed in the input graph G. Such edges can be bounded by the *projection* of the input graph onto the subset of edges that are linked to at least one node colored in the current epoch, denoted by $\pi_{R_1,\ldots,R_K}(G)$. Given that each edge is visited at most twice, one from each linked colored node, the cost of steps 2 to 4 is at most $\sum_k |R_k|^2/2 + 2|\pi_{R_1,\ldots,R_K}(G)|$.

So, the total cost of graph update for all readers in an epoch is $O(\sum_k |R_k|^2 + |\pi_{R_1,\ldots,R_K}(G)|)$. This upper bound includes a cost no more than the input graph size and some local costs quadratic in the size of the subgraph colored by each reader. It is important to note that the local quadratic costs are rare because there are usually more objects than containers, hence in practice, we see close to a linear cost in R_k. These costs are also bounded because the anti-collision protocol of an RFID reader limits the number of tags that it can read in an epoch, e.g., up to 400 using latest RFID readers such as ThingMagic Mercury 6e. The actual numbers of tags placed in real deployments are significantly less in order to ensure stable performance.

IV. Data Inference

The graph constructed from the data capture step can result in nodes that are uncolored or multicolored and possess multiple parent nodes. The data inference step estimates the most likely location of an object if it is unreported (uncolored) or is reported in multiple locations (multicolored). It also estimates the most likely container (parent) of each object. We present probabilistic techniques that include edge inference to address ambiguous containment, node inference to address uncertain locations, and an iterative procedure that applies both to the entire graph in an alternating fashion.

A. Edge inference

Edge inference is applied to all incoming edges of a node v (i.e., edges from the parent nodes of v) regardless of whether the node is colored. 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.

Edge Confirmation Algorithm. Our first technique aims to find a sequence of readings, called the *critical region*, that distinguish the true container from others. An example would be when a reader at the conveyor belt reads a case with its contained items, with no other case being observed at the same time. Moreover, the belt reader is far away from other readers so the case and its items cannot be observed elsewhere. If this trend continues for a few epochs, one shall be able to confirm the (only) case at the belt to be the container of the observed tags. This period may be short in general as the case can be soon put on a shelf together with many other cases.

We next present an information-theoretic approach to detect short critical regions when the received readings distinguish the true container from others. Given a node v, our detection algorithm works as follows:

Step 1. *Assign weights.* The first step computes a weight w_{e_i} for each incoming edge as follows:

$$\bar{w}_{e_i} = \frac{\sum_{i=0}^{W} \textit{recent\_co-locations}[i]}{\sum_{i=0}^{W} 1}, \tag{1}$$

where `recent co-locations`$[i]$ indexes the i^{th} bit of co-location bit vector and W is the size of the window used in critical region detection. We consider a small window (W=4 used in our work) since larger W values could add noise and hence make these edge weights less useful.

Step 2. *Compute probabilities.* This step computes a probability p_{e_i} for each edge by normalizing the weight.

$$\bar{p}_{e_i} = \frac{\bar{w}_{e_i}}{\sum_{j=0}^{m} \bar{w}_{e_j}} \tag{2}$$

Step 3. *Compute entropy.* This step computes the normalized entropy of the probability distribution of all incoming edges of a node:

$$h = \sum_{j=1}^{m} \bar{p}_{e_j} \log \bar{p}_{e_j} / \sum_{j=1}^{m} \frac{1}{m} \log \frac{1}{m} \tag{3}$$

As is known in information theory, the entropy is a measure of uncertainty associated with a random variable—the true parent of a node in this work. A low value of the entropy indicates less uncertainty about the true parent. Hence, whenever the entropy is significantly low, we can consider the current window as a critical region and return the edge with the highest probability as the confirmed parent. However, it is also known that a random variable with 10 equally possible values has a higher degree of uncertainty than a random variable with 2 equally possible values; that is, the entropy is sensitive to the number of possible values. To mitigate this effect, we normalize the entropy of a node with m parent edges with the maximum entropy of a m-valued random variable, i.e., when each value has probability $1/m$. Then we compare the normalized entropy, h, with a threshold, λ; if $h < \lambda$, the edge with the highest probability is chosen to be the confirmed parent. As our evaluation in Section VI-B shows, using the normalized entropy makes it easy to choose a reasonably low threshold that offers stable performance.

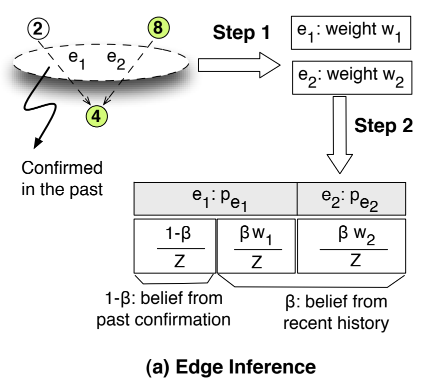
Step 4. *Edge Pruning.* Once an edge is confirmed to be the parent of the node v, all other incoming edges pointing to v can be dropped. In the example in Fig. 3(b), assume that the readings from the belt reader constitute a critical region and the edge from node 2 to node 4 is confirmed to be the parent of node 4. Then, the edge from node 3 to node 4 can be safely removed. If we further know (using domain knowledge) that a case is the top level container on a belt and indeed the belt reader only observes a case but not a pallet, we can further drop all parent edges of a confirmed container, e.g., the edge from node 1 to node 2 in Fig. 3(b).

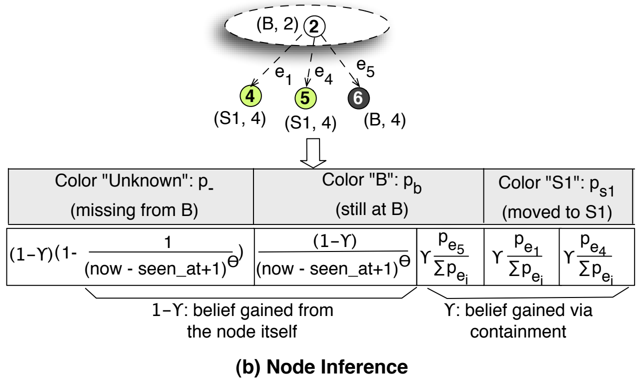
Edge Inference Algorithm. If the edge confirmation did not succeed, we perform edge inference for node v to select the most likely parent edge as the estimated container. Performing edge inference requires the use of history that includes (i) the recent history of co-locations, stored in the bit-vector `recent co-locations`, and (ii) the last confirmed parent of v, captured in `confirmed parent`. Such use of history makes edge inference less sensitive to missed readings at present time. Edge inference at a node consists of two steps, as illustrated in Fig. 5(a).

Step 1. *Assign weights:* The first step computes a weight w_{e_i} for each incoming edge as follows:

$$w_{e_i} = \frac{\sum_{i=0}^{S} \frac{\textit{recent\_co-locations}[i]}{i^{\alpha}}}{\sum_{i=0}^{S} \frac{1}{i^{\alpha}}}, \tag{4}$$

where S is the full size of the co-location bit vector. This entire co-location history is weighted using the parameter α and them normalized. α essentially implements a Zipf distribution,





(B, 2)
(S1, 4)
(S1, 4)
(B, 4)
Color "Unknown": p_- (missing from B)
Color "B": p_b (still at B)
Color "S1": p_{s1} (moved to S1)
$(1-\gamma)(1-\frac{1}{(now - seen\_at+1)^\theta})$
$\frac{(1-\gamma)}{(now - seen\_at+1)^\theta}$
$\gamma\frac{p_{e_5}}{\Sigma p_{e_i}}$
$\gamma\frac{p_{e_1}}{\Sigma p_{e_i}}$
$\gamma\frac{p_{e_4}}{\Sigma p_{e_i}}$
1-γ: belief gained from the node itself
γ: belief gained via containment
(b) Node Inference

Fig. 5. Examples of edge inference and node inference.

where $\alpha > 0$ assigns a higher weight to recent history, while $\alpha = 0$ weighs all prior co-location information equally.

Step 2. *Compute probabilities:* The next step builds a probability distribution across all incoming edges of node v. 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 as the parent of v. A parameter β is used to weigh these two factors. The probability p_{e_i} of the edge e_i is:

$$p_{e_i} = \frac{(1-\beta)m(e_i) + \beta w_{e_i}}{Z} \tag{5}$$

The memory function $m(e_i)$ takes the value '1' if e_i is the last confirmed edge and '0' otherwise. 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. Z is a normalization factor for yielding the final distribution, which is the sum of the probabilities of all incoming edges of node v. Fig. 5(a) shows a distribution across two parent edges, e_1 and e_2, of node 4, with e_1 assigned the additional weight $1-\beta$ due to its past confirmation.

Edge inference involves three parameters: (1) S, the size of the co-location history, (2) α, the zipf parameter for weighting the history, and (3) β, the partition of beliefs between the recent history and past confirmation. Section VI quantifies the sensitivity of edge inference to these parameters. In particular, we will show that the choices for S and α are quite constant but that for β can be variable. Furthermore, the value of β can be dynamically determined using an adaptive method that sets β to be the ratio of the instances that *only one* of the object and its confirmed container is observed against the instances that any of them is observed. This method is shown to perform well in Section VI.

B. Node inference

Node inference is applied to all the nodes in the colored graph. If a node is multicolored, node inference chooses the most likely color as the estimated location of the corresponding object. If a node is uncolored instead, node inference attempts to infer the most likely location of the object or confirm its absence from any known location. A key challenge in node inference arises from a three-way tradeoff among *continued stay*, *movement to a new location*, and *absence from any known location*. These situations are depicted in Fig. 5(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 (continued stay); it moved to location S_1 with its contained objects and its reading was missed in S_1 (movement to a new location); it disappeared from B and its current location is unclear (absence from any known location).

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

Formally, the probability of the node v having color c_i is:

$$p_{c_i}(v) = (1-\gamma)\frac{\delta'(v,c_i)}{(now - seen\_at + 1)^\theta} + \gamma \sum_{e_i \to c_i} \frac{p_{e_i}}{Z_2} \tag{6}$$

$$\delta(v,c_i) = \begin{cases} 1, c_i \text{ is a recent color of v} \\ 0, \text{otherwise} \end{cases}, \ \delta'(v,c_i) = \frac{\delta(v,c_i)}{\sum_j \delta(v,c_j)}$$

Here $\delta(v,c_i)$ is an indicator function that takes the value 1 if c_i is one of the recent colors for the node v, and 0 otherwise; $\delta'(v,c_i)$ is the value normalized across all colors. The parameter θ controls the rate of fading of a recent color.

We further consider the influence of the colors of the neighboring nodes, hence taking advantage of the containment relationship. If a node v has acquired multiple colors, we consider these colors observed at the neighboring nodes and include them in the node inference at v. If the node is not colored in the current epoch, we take into account both the observed and inferred colors at the neighboring nodes. In the above formula, $e_i \to c_i$ means that the edge e_i propagates the color c_i to v, and Z_2 is the normalization factor across all edges of v that propagate colors to v. Of particular interest is the parameter γ that weighs the colors that originate from the node against the colors that propagate through the edges.

Finally, the probability of the special color "unknown" is:

$$p_{unknown}(v) = (1-\gamma)(1 - \sum_{j\in[1,m]} \frac{\delta'(v,c_j)}{(now - seen\_at + 1)^\theta}) \tag{7}$$

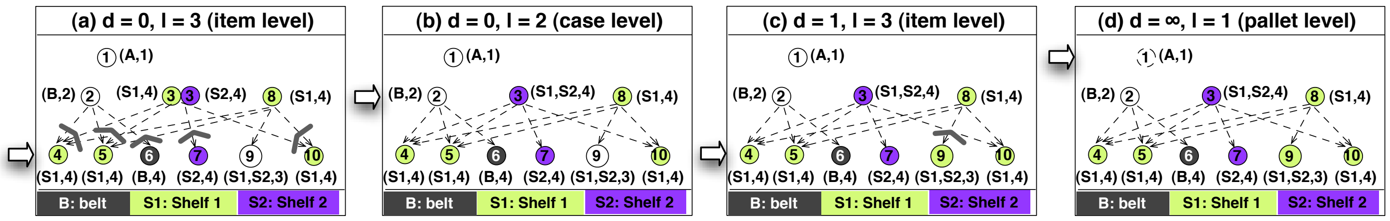

(a) d = 0, l = 3 (item level)
(b) d = 0, l = 2 (case level)
(c) d = 1, l = 3 (item level)
(d) d = ∞, l = 1 (pallet level)
(A,1)
(B,2)
(S1,4)
(S2,4)
(S1,S2,4)
(B,4)
(S1,S2,3)
B: belt
S1: Shelf 1
S2: Shelf 2

Fig. 6. Illustration of iterative inference across the graph in increasing distance from the colored nodes.

where m denotes the number of parent edges of v. As can be seen, a colored node always has the "unknown" probability 0 because `now` equals `seen_at` and the sum of $\delta'(v, c_j)$ is 1.

It is evident that the quantities computed for all colors using Eq. 6 and Eq. 7 sum up to 1, hence forming a probability distribution. Fig. 5(b) shows the resulting probability distribution over three colors, B, S_1, and "unknown".

In summary, node inference is influenced by two parameters: (1) γ weighs the node colors assigned based on the assumption that the object is independent of other objects, against the colors that are propagated from edges based on the containment relationships; and (2) for the former set of colors, θ, the exponent of the function $(now - seen\_at + 1)^{-\theta}$, further adjusts the distribution of the probability mass between the fading color and the "unknown" color. We quantify the sensitivity of node inference to γ and θ in Section VI.

C. *Iterative Inference*

Iterative inference combines node and edge inference to iterate over 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 involves the colors of its neighboring nodes and the probabilities of the edges between those nodes, so it cannot begin until these dependencies are first resolved.

The key idea of our iterative algorithm is to start inference from the colored nodes—the nodes with observed 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.

To run iterative inference, we classify nodes based on their closest distance, d, from a colored node in the graph, and visit the nodes in increasing value of d. For the nodes of the same distance d, we visit them in decreasing value of the packaging level ℓ (e.g., from the items to the cases and then to the pallets). For each node visited, we first perform edge inference among all incoming edges of the node, and then node inference involving all neighboring nodes at distance d-1 or less, except that if a node is colored, i.e., at $d = 0$, we consider all colored neighboring nodes, which are also at $d = 0$. Fig. 6 illustrates this process for the graph in Fig. 3(d). (The pseudocode is left to [17] due to space constraints.)

The algorithm first considers the nodes with the distance $d = 0$ (colored) and the packaging level $\ell = 3$ (at the item level), including nodes 4, 5, 6, 7, and 10 as shown in Fig. 6(a). Edge inference is performed for these nodes to estimate their most likely parents, where a gray bar marks the edges considered in the edge inference at a node. Since each of the nodes has only one color, the node inference is trivial. Then the algorithm considers the nodes with $d = 0$ and $\ell = 2$ (at the case level), including nodes 3 and 8 as shown in Fig. 6(b). Since these nodes do not have incoming edges, edge inference is simply skipped. During node inference, node 3 is assigned the color for shelf 2. This is because the edge from node 3 to node 7 was confirmed as a parent edge of node 7 when they were scanned on the belt, and now the high probability of this edge transfers from node 7 significant evidence for the color for shelf 2, outweighing the color for shelf 1. Next, the algorithm considers the nodes labeled with $d = 1$ (uncolored) and $\ell = 3$ (at the item level), including only the node 9 as shown in Fig. 6(c). The node inference chooses the color for shelf 1 due to the containment relationship with node 8, which is observed at shelf 1. Finally, the algorithm considers node 1 with $d = \infty$ (disconnected from any colored node) and $\ell = 1$. It has not been observed since t=1 and hence gains the highest probability for the unknown location, i.e., reported missing from the physical world observable by the existing readers.

Complexity and Optimizations. The complexity of the iterative algorithm is bounded by the number of edges examined in the graph $G(V, E)$. Given that each edge is visited at most twice, once from each linked node, the complexity is $O(|E|)$.

To improve time and space efficiency, we can further use a graph pruning routine in the iterative inference procedure. First, if an object exits the physical world through a proper channel, e.g., through an exit door of a warehouse, and is detected by the reader at that place, after inference at the node representing this object, our system removes the node and any associated edges from the graph. Second, after edge inference at a node, we can also use the edge weights to prune edges that are unlikely to be the true containment. To do so, we use a threshold (with a default value 0.25) to remove edges whose weights are below the threshold.

Partial Inference. A practical issue to consider is that RFID readers may read at different frequencies. In a warehouse, for instance, belt readers may read once every second while shelf readers may read once every 10 seconds. If we run inference for the objects whose closest readers are inactive for a while, the inferred locations are likely to be the "unknown" location, which are different from the true locations. To address the issue, our system performs partial inference in the epochs when not all the readers are active, by (1) restricting inference to the subset of the graph up to 1 hop away from the colored nodes, and (2) withholding the inference result of the "unknown" location until a later time when all readers are active to run complete inference and output accurate results.

Conflicts Resolution. A final issue with the iterative inference algorithm is that it may result in different colors inferred for the two nodes of an edge. This is because the colors of the two nodes were inferred individually in steps d and $d+1$. If the edge between the two nodes is also chosen to represent their containment relationship, then the inference is yielding conflicting information: the container and the contained object are reported in different locations. In the example in Fig 3(d), node 7 is observed at shelf 2. Suppose that the edge from node 3 to node 7 is inferred to be node 7's parent but the location of node 3 is inferred to be shelf 1 (which is unlikely but used for the sake of an example). Now we have a conflict between location and containment inference. In our system, we preserve the graph model with all the statistics as is, and resolve conflicts in a post-processing step after inference.

Our guideline on conflict resolution is to give priority to a containment relationship. This is because the containment is often based on the confirmed parent that was derived with high probability before, as described in Section IV-A. Hence, given a reported containment relationship, if the parent and child nodes have different inferred colors, we use the parent's location to override the child's location—here we favor the parent's location because its inference has taken into account the locations of all its children, in particular, those that have confirmed containment relationships with this parent node.

V. Stream Output with Compression

The output module of the SPIRE system takes the results of data inference and transforms them into a compressed event stream for output. Compared to the raw RFID stream, the output based on inference results adds location information for unobserved objects and containment information not available in the input stream. The key idea behind our compression methods is that only those readings that indicate a *state change*, such as the change of an object's location or containment relationships with others, need to be included in the output stream. In the absence of a state change, all readings merely confirm the current state of the world and hence are redundant; these readings can be safely discarded. Hence, the compressed output stream contains richer information yet with a reduced data volume. Below, we describe the data format of a compressed stream and two compression techniques.

A. Data Format of a Compressed Event Stream

A compressed output stream contains location and containment events that occur in a time interval, called the event's *validity interval* [18]. The validity interval is represented by two timestamps, V_s for the start time and V_e for the end time. Our compressed output format represents these events using the following five messages:

- StartLocation(object, location, V_s, V_e = ∞)
- EndLocation(object, location, V_s, V_e)
- StartContainment(object, container, V_s, V_e = ∞)
- EndContainment(object, container, V_s, V_e)
- Missing(object, locationMissingFrom, V_s, V_e = V_s)

Start and end location messages always occur in pairs and encapsulate the time period when an object is inferred to be present at a particular location. The difference is that the start location message of an event sets only the V_s timestamp, leaving V_e with the default value ∞, while the end location message later sets V_e. Similarly, start and end containment messages encapsulate the time period of a containment relationship. Missing messages are singletons always output after an endLocation event for the object's previous location.

In this work, we call a compressed stream *well-formed* if for a given object, every start location (containment) message has a matching end location (containment) message and a missing message appears outside any start-end location pair. Our system guarantees well-formed output, and at the same time, allows location and containment update events to be nested in the most flexible way. 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.

B. Range Compression

Our first compression method, which we refer to as range compression or level-1 compression, leverages the fact that if an object is stationary—resident at the same location for a period of time—its entire stay at this location can be represented by a single ranged location event. Likewise, if an object has a stable containment—contained in the same case or pallet for a period of time—this containment relationship can also be represented by a single ranged containment event. The method is implemented by simply comparing an object's newly inferred state (either location or containment) to its previously reported state. For an object that is inferred to be missing, we output an EndLocation message to complete the previous location event and then a singleton Missing message.

The output stream of range compression has two properties. First, location compression and containment compression are performed separately. Hence, it is possible to split the output into separate location and containment update streams, and suppress the output of one stream if not needed. Second, the result stream of range compression includes complete location and containment information for each object, and hence is directly queriable by event systems such as [18], [19].

C. Location Compression using Containment

Our second compression method, referred to as level-2 compression, uses the additional knowledge that under stable containment, location readings of child objects can be further suppressed because they are identical to that of the parent.

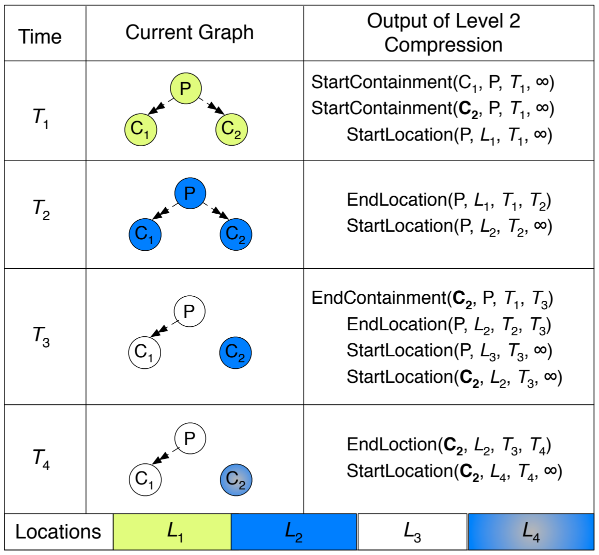

Time
Current Graph
Output of Level 2 Compression
T_1
P
C_1
C_2
StartContainment(C_1, P, T_1, ∞)
StartContainment($\mathbf{C_2}$, P, T_1, ∞)
StartLocation(P, L_1, T_1, ∞)
T_2
EndLocation(P, L_1, T_1, T_2)
StartLocation(P, L_2, T_2, ∞)
T_3
EndContainment($\mathbf{C_2}$, P, T_1, T_3)
EndLocation(P, L_2, T_2, T_3)
StartLocation(P, L_3, T_3, ∞)
StartLocation($\mathbf{C_2}$, L_2, T_3, ∞)
T_4
EndLoction($\mathbf{C_2}$, L_2, T_3, T_4)
StartLocation($\mathbf{C_2}$, L_4, T_4, ∞)
Locations
L_1
L_2
L_3
L_4

Fig. 7. An example of level-2 compression for a group of objects.

The benefit of dosing so is to minimize the location output to only the location of top-level containers. This compression is lossless because the location of a contained object can be recovered from its containment relationship and the location of its top-level container.

An example for level-2 compression is shown in Fig. 7. At time T_1, a pallet P and two cases C_1 and C_2 are observed at location L_1 (for simplicity of presentation, we omit items in this example). A StartContainment is output for each of the contained cases. Given the containment relationships, only a StartLocation is output for P, the single top-level container. At time T_2, the three objects move as a group to L_2. Only the location of P is updated due to level 2 compression. At time T_3, the three objects are split to two groups. P and C_1 move to location L_3 while C_2 stays at L_2. As a result, the containment between C_2 and P is broken, signaled by the EndContainment for C_2. Then location updates for C_2 are soon output since C_2 is no longer contained. In contrast, C_1 is still contained in P so only location updates are sent for P.

This compression method has different properties from the range compression method. First, the location and containment output streams are no longer independent. In particular, a reported containment and the related location updates of the container need to be correlated to recover the locations of the contained objects. Second, the output stream is not directly queriable by event processors due to the lack of location information of some objects. To facilitate query processing, our system offers a decompression routine that transforms a level-2 compressed stream to a level-1 compressed stream. This routine can be plugged into the front end of a query processor to decompress the input stream *on demand*, e.g., to retrieve locations of objects in a certain period of time as requested by the queries.

The routine works as follows. For each time step, it first processes all containment updates to reconstruct the current object containment hierarchy. For each StartContainment event received, the child object specified in the event is added to the children list of the parent. For each EndContainment event, the child object is removed from the parent's list. After processing all containment updates, the routine then processes the location updates in this time step. For each location update, which can be StartLocation, EndLocation, or Missing, the routine copies the event to the new output stream. If the object specified in the event has child objects in the containment hierarchy, this location update is also copied to the output stream for each child object and recursively for their contained objects.

A subtlety is that the routine also needs to remember each object's current location to suppress duplicate events in output. Revisit Fig. 7. At time T_3, object C_2 was no longer contained in P so we output a StartLocation to report its location in L_2. However, when we decompress the stream from level-2 compression, we will output a StartLocation for C_2 at L_2 at an earlier time T_2 (this update was compressed before due to the containment with P). Then the SartLocation at T_3 that reports the same object at the same location becomes a duplicate. Our decompression routine will remove all such duplicates.

TABLE I
PARAMETERS USED FOR GENERATING RFID STREAMS.

Parameter	Value(s) used
Duration of simulation	3 – 24 hours
Rate of pallet injection	1 per 4 – 600 seconds
Cases per pallet	8
Items per case	20
Read rate (RR) of readers	0.5 – 1 (default 0.85)
Overlap rate (OR) for shelf readers	0 – 0.8 (default 0.25)
Non-shelf reader frequency (fixed)	1 (interrogation) per sec
Shelf reader frequency (variable)	1 per sec – 1 per min (default)

VI. Performance Evaluation

We have implemented a prototype of our inference and compression substrate in Java. In this section, we evaluate the accuracy and efficiency of our inference techniques using both synthetic traces emulating enterprise supply chains and real traces from a laboratory warehouse setup. We also explore the benefits of compression based on results of inference.

A. Simulation Design

We first developed a simulator that emulates deployments of RFID readers in a large warehouse. Pallets arrive at the warehouse at a certain rate. They are first read at the entry door (using reader group 1). They then become unpacked. The contained cases are scanned on the receiving belt (using reader group 2), placed onto shelves for a period of stay (scanned by reader group 3), and then repackaged (scanned by reader group 4). The newly assembled pallets are rescanned on the belt (using reader group 5) and finally read at the exit of the warehouse (using reader group 6). The parameters for the simulation are shown in Table I. The read rate (RR) parameter specifies the probability that a tag is read by its closest reader. The overlap rate (OR) parameter is applied to shelf readers and specifies the probability that a tag is read by a nearby reader to its left or right. The read frequencies of shelf readers and non-shelf readers are controlled separately. This design allows flexible settings of the simulation where items may stay on shelves for hours and shelf readers may read less

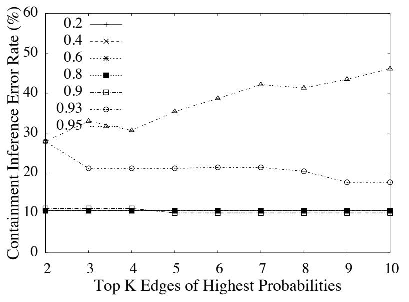


Fig. 9. Evaluation results of the edge confirmation algorithm.

frequently than other readers. Synthetic data streams from the simulator are fed to our inference and compression substrate. Data inference is performed every epoch (which is 1 second). We use entry door readings to "warm up" the graph model but do not run inference at this location.

B. Accuracy of Data Inference

We first evaluate the accuracy of our inference techniques. We created data streams with 6 pallets injected per hour, an average shelving period of 1 hour, and a total simulation time of 3 hours. An inference result is marked as an error if it is inconsistent with the ground truth.

Containment Inference. We first study the effects of the edge inference parameters, S, α, and β (shown in Eq. 4 and Eq. 5), on containment inference. Our results show that the two parameters, S and α, on the recent history of co-locations can be tuned easily: The size of the history, S, limits the inference accuracy when it is small, e.g., 4, 8, but offers no additional benefit after the point of 32. The zipf parameter, α, yields best accuracy when set to 0, indicating that recent co-location instances are equally important to inference. Hence, we use $S = 32$ and $\alpha = 0$ in the rest of experiments.

The parameter β governs the beliefs between the recent history, which can be noisy, and the past edge confirmation, which may be obsolete. $\beta = 1$ gives all the weight to recent history and $\beta = 0$ does the opposite. We compare the use of fixed β values and an adaptive method that sets β based on the number of conflicting observations (see Section IV-A).

Fig. 8(a) shows the results as the read rate (RR) is varied. The adaptive method performs the best for different RR values tested. Among the fixed values, large β values tend to produce more errors in this workload. This is because many cases can stay on the same shelf for an extended period of time and their shelf readings are a main source of noise in containment inference. Putting too much weight on such noisy recent history causes many errors. In contrast, the edge confirmation algorithm confirms the true parent from the readings from the belt reader. The adaptive method can automatically apportion its belief on such past confirmation based on the number of conflicting observations received. We have also demonstrated the effectiveness of the adaptive method when various overlap rates, read rates, and read frequencies are used. The details are available in the appendix.

We next evaluate our technique for edge confirmation. As described in Section IV-A, at each node we compute the normalized entropy h over all parent edges and check if h is lower than a threshold λ (supposed to be relatively small). To choose a universal λ for all nodes, it is helpful to model each node using a k-valued random variable: if a node has more than k parent edges, we consider the top k edges with the highest probabilities; if a node has less than k edges, we add a few virtual edges and assign them the smallest non-zero probability. In this experiment, we vary k and λ values. As Fig. 9 shows, all λ values under 0.9 offer similar accuracy around 10%, irrespective of the choice of k. This is because the critical information for parent edge confirmation is often the sharp difference in probability between the top two edges. However, when λ is set too high, over 0.9 here, more false positives of edge confirmation occur, and the accuracy becomes sensitive to the choice of k. Similar observations hold for various read rates and overlap rates (see the appendix for more details). Hence, we use λ=0.75 and k=2 in the rest of study.

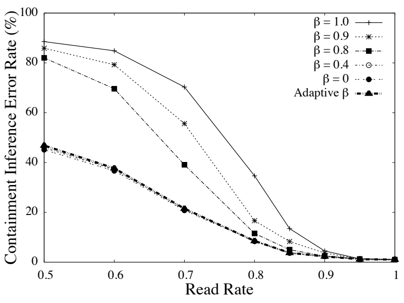
Location Inference. Location inference uses the node inference method defined in Eq. 6 and Eq. 7. We now study the effects of two parameters on location inference.

The parameter γ weighs the belief of an object's recent locations (favored by low γ values) against the belief of its location inferred via containment (favored by high γ values). Fig. 8(b) shows the results for varied γ values. Very low γ values place most emphasis on the recent locations (which function as fading colors). As such, if an object has experienced several missed readings, it is likely to be inferred to be in the "unknown" location even if its container has been observed. High γ values place too much weight on the containment information of an object, which can be unreliable when containment is uncertain. Overall, we observe γ values in [0.2, 0.4] to be favorable from this plot and many others which cover a wide range of read rates, overlap rates, and read frequencies (for more see the appendix). γ values in this range offer a balance between an object's recent colors and containment relationships, with some more weight on the former.

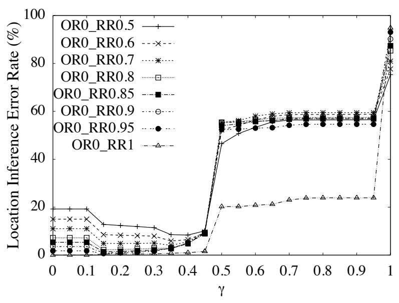
The parameter θ is the dampening factor on the belief of the continued existence of an unobserved object in a recently reported location. High θ values cause more quickly reduced belief, making it more likely to refer the object to be in the "unknown" location (e.g., in transit or missing). Fig. 8(c) shows that as θ increases, the error rate quickly declines from over 90%, flattens in the mid-range from 1 to 2, and then degrades slightly with higher values. The initial decline occurs because, with very low θ values, the inference takes long to reduce its belief of the continued presence of an object even if the object left a while ago. The deterioration with high θ values occurs because the inference now drops its belief of continued existence too quickly and identifies an object as being away after just a few missed readings. Similar trends are observed for different overlap rates and read frequencies.

Based on the above results, we use the adaptive algorithm to set the β value but fixed values γ = 0.4 and θ = 1.25 in the rest of the study. We demonstrate the validity of these values for the real traces from our lab deployment in the next section.

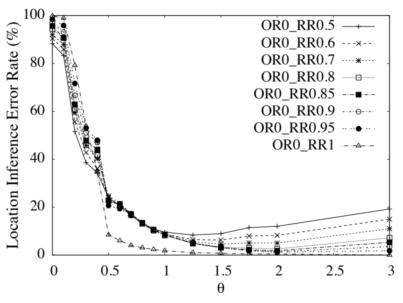
Sensitivity to Read Rate. The next experiment reports the effect of the read rate (RR) parameter on the accuracy of both



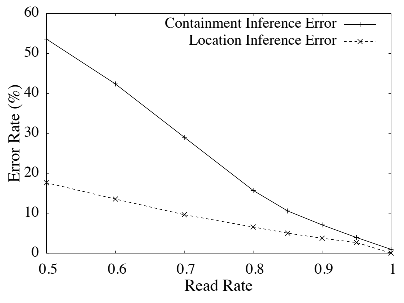
(a) Containment inference error for β



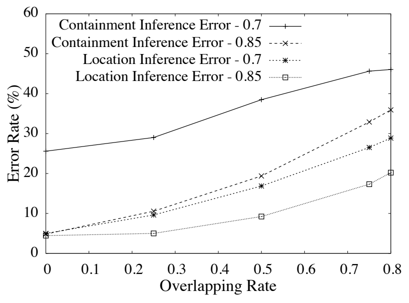
(b) Location inference error for γ



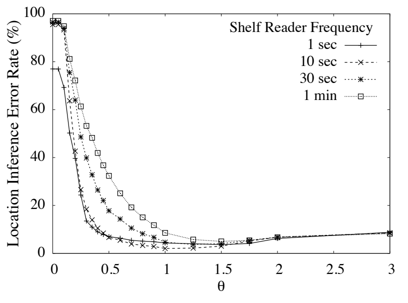
(c) Location inference error for θ



(d) Inference error for varied read rates



(e) Inference error for varied overlap rates



(f) Inference error (1 anomaly/100 sec)

Fig. 8. Containment and location inference results.

location and containment inference. We varied the read rate uniformly for all readers. As Fig. 8(d) shows, when the read rate is above 0.8 (observed in our lab deployment when the environment is clear of metal objects), our location error rate is less than 10% and the containment error rate is less than 20%. As the read rate decreases, the accuracy of containment inference degrades faster than that of location inference. This is because the location inference relies more on the recent locations given our parameter setting, while the containment inference suffers both the loss of parent edge confirmation due to poorer readings from the belt readers and lack of consistent observations in the recent history.

Sensitivity to Overlap Rate. We next vary the overlap rate (OR) in a wide range [0, 0.8] with the read rate (RR) set to 0.85 and 0.7, respectively. We note that in practice one would expect OR to be significantly less than 0.5, or otherwise would consider improving the RFID setup to avoid excessive overlap between readers. Fig. 8(e) shows the inference error rates. First, for common setups where RR is in [0.8, 1] and OR is in [0, 0.3], our system offers both location and containment error rates within 10%. When stress testing our system with higher OR values, we see that the location inference error rates degrade modestly to 9% (RR=0.85) and 17% (RR=0.7) when OR becomes 0.5. The main effect of OR on location inference is to add more possible locations for each object and in certain cases the readings from a nearby reader may outnumber those from the closest reader, hence causing more errors. In comparison, the containment error rates are higher because containment inference heavily relies on co-location information and significant overlap between readers makes this information noisy. The reduced read rate (from 0.85 to 0.7) makes the co-location information even noisier and allows fewer true containment relationships to be confirmed from the belt readings, hence the worst performance in the top line.

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. We next 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 in Fig. 8(f) as the most relevant parameter, θ, is varied. This figure exhibits similar trends as Fig. 8(c) and confirms that the θ values between 1 and 2 also work well for anomaly detection. We also measured the delay of anomaly detection as θ varies (the graph is shown in the appendix). In general, large θ values yield short delay whereas small θ values cause high delay. We observe that the region of 1 and 2 gives the delay as good as larger θ values. Thus, $\theta \in$ [1,2] is shown to offer both accuracy and short detection delay.

C. Accuracy Results using a Lab Deployment

To evaluate our system in real-world settings, we developed an RFID lab with 7 readers based on ThingMagic's Mercury5 RFID system. We used 20 cases containing 5 items each, and attached Alien squiggle tags to all cases and items. We used the 7 readers to implement 1 entry door reader, 1 belt reader, 4 shelf readers, and 1 exit reader. Cases with contained items transitioned through the readers in that order, receiving around 5 interrogations from each nonshelf reader and dozens from a shelf reader. The shelf readers had overlapping read ranges as they were placed close to each other. We created a collection of traces with distinct characteristics regarding the environmental noise, overlap between readers, and tag orientations. We observed that tag orientations had little impact on the read rate, verifying that squiggle tags are orientation-insensitive when used with circularly-polarized readers. Hence, we used the following 8 traces in our experiments:

- T_1 represents the combination of good read rates, an

TABLE II
INFERENCE ERROR RATES USING LAB TRACES

(RR , OR) =	(0.85, 0.25)	(0.85, 0.5)	(0.7, 0.25)	(0.7, 0.5)
	T1	**T2**	**T3**	**T4**
Loc.	7.0%	8.4%	13.3%	16.5%
Cont.	9.0%	14.2%	14.3%	17.3%
	T5	**T6**	**T7**	**T8**
Loc.	11.7%	15.4%	16.1%	18.4%
Cont.	14.0%	15.5%	16.8%	19.7%

average of 0.85 across all readers, and limited overlap rates, an average of 0.25 for shelf readers, which we obtained by using low power of readers.

- T_2 represents the combination of good read rates, an average of 0.85, and significant overlap rates, an average of 0.5, which we obtained using high power of readers.
- T_3 differs from T_1 by including severe noise in the sensing environment. More specifically, we placed a metal bar on each shelf that is 1/3 the length of the shelf, causing the average read rate to drop to 0.7.
- T_4 differs from T_2 with an average read rate of 0.7.
- T_5 to T_8 extend T_1 to T_4, respectively, with added anomalies. With 20 cases placed on shelves, we randomly selected 4 cases and removed one item from each case to generate missing objects in the trace.

Each trace is 15 minutes long with 15,000 to 22,000 readings. We ran inference every 3 minutes and manually collected ground truth, including object locations and containment relationships, at those time points.

Table II reports the inference results. We make several observations. First, despite the added environmental noise and anomalies of missing objects, our inference techniques still offer the location and conference inference error rates within 20%. Second, under the normal conditions of RR=0.85 (without the metal bars placed on shelves), both error rates are within or around 15%. Third, for the traces without anomalies (T_1 to T_4), our inference results are similar to our simulation results shown in Fig. 8(d) and Fig. 8(e), hence demonstrating the validity of our simulation results. In particular, we observe that the containment inference error rates are somewhat better than those in Fig. 8(e). This is because the belt reader in our lab deployment produced more readings (5-6 readings on the average) for each tag than in our simulation (3-4 per tag). These extra readings allowed our edge confirmation algorithm to confirm more containment relationships. In some cases, the location inference error rates are slightly higher than the simulation results. The main reason, as we observed, is that for some tags more readings were returned from a nearby reader than from the closest reader in the lab traces. Such irregular data makes location inference more difficult.

We next discuss two limitations of our lab data set and the methods that we employ to overcome them in our evaluation. First, our data set with 100 items and 20 cases is not large enough for evaluating the efficiency and scalability of inference. In the next section we will extend our evaluation with high-volume synthetic streams that contain hundreds of thousands of objects. Second, our data set covers several, but not all, combinations of read rates and overlap rates across readers. Due to limited resources, it is hard to produce all possible combinations through manual configuration. Fortunately, our results using the lab traces match the simulation results presented above. Hence, our simulation results obtained for a broad range of parameter settings can serve as an indicator of the performance expected to be seen in real deployments.

TABLE III
COSTS OF UPDATE AND INFERENCE OPERATIONS (SEC)

Num. Objects	**Update**	**Inference**	**Total**
25068	0.0055	0.0638	0.0693
55118	0.0079	0.1518	0.1597
75066	0.0096	0.2295	0.2392
115144	0.0141	0.4249	0.4389
155044	0.0199	0.6313	0.6512
172550	0.0229	0.7345	0.7573

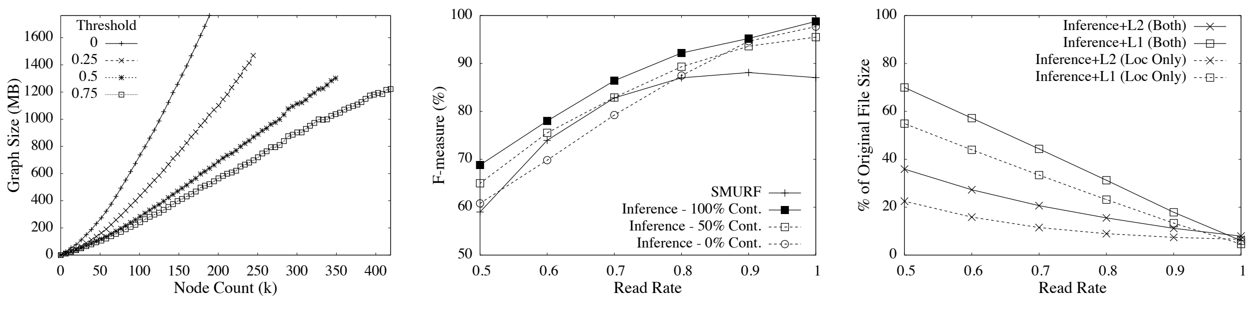
D. Efficiency of Data Capture and Inference

We next evaluate the efficiency of our system in both memory usage and processing speed. To do so, we used a high pallet injection rate of 1 every 4 seconds (900 pallets per hour). The tests were performed on a linux server with Intel 2.33GHz Xeon CPU and 8GB memory running JVM 1.6.0. The maximum Java allocation pool size was set to 5.5GB.

Processing Speed. Table III reports the processing time for graphs of different sizes. With the increasing graph size shown in the first column of the table, the cost of graph update for all active readers and the cost of inference on the graph in each epoch (which is a second) are reported in the second and third columns. As can be seen, both the update and inference costs are less than a second, with the inference being the dominant cost. The total cost is reported in the last column. As is shown, the total cost is 0.76 second for the largest graph size, which uses an already high injection rate of one pallet every 4 seconds. These results show that our data capture and inference techniques can keep up with high-volume RFID streams, a key requirement of RFID monitoring and tracking.

Memory Usage. The memory usage in our system is dominated by the size of the graph. In this experiment, we measured the memory usage with varied graph sizes. Note that when an object leaves a warehouse, we remove its node and all of the associated edges to keep the graph small. We also observe that the graph size can be reduced by pruning edges for which the containment inference yields low confidence. The confidence value here is the value in Eq. 5 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 0.75.

Fig. 10(a) shows the rate of memory usage increase as the graph size grows (until our system runs out of memory). First, we see that the memory usage increases fast without edge pruning but less so with increased thresholds for pruning. With a threshold of 0.75, pruning is able to keep the size of the graph under 1.2GB, even with 400,000 objects present in the system. In addition, the memory growth of using the 0.5 and 0.75 thresholds is shown to be close to linear, rather



(a) Memory usage (b) Event error rate (c) Compressed location and containment output

Fig. 10. Memory test and evaluation of the output event stream.

than the worse-case quadratic expansion in the number of nodes. Finally, we note that the pruned edges have little effect on the location inference error rate (less than 1% difference between no pruning and pruning with the threshold 0.75), but may cause up to 8.2% increase in error rate for containment inference, which is a small cost to pay if memory is scarce.

E. Accuracy and Data Reduction of the Output Event Stream

After data inference, our system translates inference results into output events using level 1 or level 2 compression (Section V). We next compare the final output of our system against SMURF [11], a state-of-the-art RFID data cleaning system. SMURF applies smoothing with an adaptive window to mitigate the missed reading problem. To enable a comparison to our output, we extend SMURF as follows: (i) We use the static reader locations to estimate object locations as readings are smoothed in. (ii) If an object is observed by multiple readers, we compute the *strength* of each reader as the percentage of readings in its current window and report the location of the reader with the highest strength as the object's location. (iii) We finally apply level 1 compression to produce a compressed event stream. *SMURF does not support containment inference or level 2 compression*, which is unique to our system. For these reasons, we only consider object location events in the output when compared to SMURF.

In our experiments we used a 16 hour trace with the steady-state volume of 2860 objects (when the numbers of arriving and departing pallets are equal). We varied the read rate for all the readers from 0.5 to 1 with the overlap rate set to 0.25.

Accuracy of Output Events. Our accuracy metric for the output stream is event-based: for each event in the output, we determine if it is present in a compressed event stream of the ground truth. We use *precision* to capture the percentage of returned events that exist in the ground truth stream, and *recall* to capture the percentage of events in the ground truth stream that are returned in our output. We combine them into the metric *F-measure* = 2*precision*recall/(precision+recall).

Fig. 10(b) compares SMURF and our system in F-measure for location inference. Since the compression technique does not affect accuracy, we use level 2 compression here. The two solid lines in the figure show that our system outperforms SMURF by more than 10% in both the low and high read rate ranges. In the low range, this is largely due to SMURF being simply a smoothing technique: it can smooth in readings in certain cases when an object is missed by its reader. However, given several consecutive missed readings, it tends to believe that this object has moved away from its location. In contrast, our system exploits stable containment, i.e., using the location of the container or contained objects to infer this object's location, thereby overcoming the problem of consecutive missed readings. As the read rate approaches 100%, the accuracy of SMURF remains below 90%. This is due to the de-duplication process employed by SMURF. When an object arrives at a new location, its adaptive window for smoothing starts small. If it is observed by multiple readers, there is a good chance that the strength computed from the windows for these readers is all the same. Then the choice between these readers is random, hence causing the errors.

Furthermore, we consider the cases where only a percentage (e.g., 50% and 0%) of objects are placed in a container, hence permitting limited benefits of using the containment information in location inference. In the extreme case of 0% containment (which is rare in practice), SMURF outperforms our inference in the range [0.6, 0.8] because without containment information, SMURF's dynamic window technique is more adaptive than our location inference technique that records the last n observed colors and employs color fading. However, given larger read rates (> 0.8), SMURF produces more errors because its dynamic window produces more false positives than our technique and its de-duplication process causes errors as explained above. The curve for 50% containment lies in between those for 100% and 0% containment.

Compression Ratio. To study the effect of compression, we measure the size of a compressed event output against the size of the initial input of raw RFID readings (i.e., *compression ratio*). Fig. 10(c) shows results for two output formats.

First, we only include location information in the output (shown by the two dashed lines in Fig. 10(c)). Since our system supports containment inference, besides level 1 compression it can also apply level 2 compression to suppress location updates of contained objects. From the figure we can see that level 2 compression offers a greater reduction in output size than level 1 for almost all read rates, resulting in a compression ratio of 10% when the read rate exceeds 0.8. We note that SMURF's output size is always larger than level 2 compression, hence omitted in the graph for readability.

Second, we further include the containment information in the output (shown by the two solid lines in the figure). We again observe that level 2 compression significantly outperforms level 1 compression. It is also interesting to see that including both location and containment information using level 2 compression yields less output than including only location information using level 1 compression. It takes less than 20% of the raw input data size to include such rich information when the read rate reaches 80%.

In summary, our system outperforms SMURF for object location updates in both error rate and compression ratio. Moreover, containment inference is unique to our system and allows level 2 compression that produces rich location and containment information yet at a much reduced data volume.

VII. Related Work

RFID stream processing. Several techniques have been proposed recently to clean noisy RFID data streams [10], [6], [11], [20]. The most relevant to our work is the HiFi system [10], [6] that performs per-tag smoothing using the SMURF algorithm [11] and multi-tag aggregation, but does not capture containment relationships between objects or estimate object locations via containment. We have experimentally demonstrated the benefits of our techniques over SMURF. Our prior research considered the use of a single *mobile* reader to scan objects repeatedly from different angles and distances, and developed inference techniques to derive precise object locations [21]. Our work presented in this paper focuses on a network of *static* readers and infers both object location and containment relationships. Other research on probabilistic RFID query processing has focused on the architectural design [12] or event pattern detection [13], but has not addressed combined location and containment inference. Since our system produces an event stream with rich location and containment information, we can feed our output stream to probabilistic query processing to derive useful high-level information.

RFID databases. General RFID data management issues including inference are discussed in [22]. Siemens RFID middleware [15] uses application rules to archive RFID data streams into databases. The Cascadia system [23] offers an infrastructure for specifying event patterns, extracting events from raw RFID data, and storing them into a database. Insides RFID databases, advanced techniques are available to integrate data cleansing with query processing [24], to recover high-level information from incomplete, noisy data by exploiting known constraints and prior statistical knowledge [25], and to use effective path encoding schemes to answer tracking queries and path oriented queries. Furthermore, effective compression is available through the use of disk-based sorting and summarization operations [14]. These techniques, however, are not designed for fast low-level inference and compression of raw RFID streams. Furthermore, none of them supports containment inference or has demonstrated performance for inference over high volume RFID streams.

Sensor data management. Recent work on GPS sensor readings is related to our work since it supports use-defined views using model-based probabilistic inference [26]. However, GPS data differs from RFID data because it already reveals object locations and GPS applications are not concerned object containment relationships.

VIII. Conclusions

In this paper, we presented a novel data inference 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 containment, and employs a probabilistic inference algorithm to determine the most likely location and containment for each object. Our results showed that our techniques achieve error rates below 15% for location estimates for a wide range of RFID read rates, and within 20% for containment estimates when the read rate reaches 80%. For future work, we plan to extend our inference and compression substrate to a mix of static and mobile readers and to handle query processing in distributed environments.

References

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

[2] Y. Yao and J. Gehrke, "Query processing in sensor networks." in *CIDR*, 2003.

[3] S. Madden, M. J. Franklin, et al., "The design of an acquisitional query processor for sensor networks," in *SIGMOD*, 2003, pp. 491–502.

[4] A. Deshpande, C. Guestrin, et al., "Model-driven data acquisition in sensor networks." in *VLDB*, 2004, pp. 588–599.

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

[6] S. R. Jeffery, G. Alonso, et al., "Declarative support for sensor data cleaning." in *Pervasive*, 2006, pp. 83–100.

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

[8] C. Floerkemeier and M. Lampe, "Issues with RFID usage in ubiquitous computing applications." in *Pervasive*, 2004, pp. 188–193.

[9] B. Violino, "RFID opportunities and challenges." http://www.rfidjournal.com/article/articleview/537.

[10] M. J. Franklin, S. R. Jeffery, et al., "Design considerations for high fan-in systems: The HiFi approach." in *CIDR*, 2005, pp. 290–304.

[11] S. R. Jeffery, M. N. Garofalakis, and M. J. Franklin, "Adaptive cleaning for RFID data streams." in *VLDB*, 2006, pp. 163–174.

[12] M. N. Garofalakis, K. P. Brown, et al., "Probabilistic data management for pervasive computing: The data furnace project." *IEEE Data Eng. Bull.*, vol. 29, no. 1, pp. 57–63, 2006.

[13] C. Ré, J. Letchner, et al., "Event queries on correlated probabilistic streams," in *SIGMOD*, 2008, pp. 715–728.

[14] H. Gonzalez, J. Han, et al., "Warehousing and analyzing massive RFID data sets." in *ICDE*, 2006, p. 83.

[15] F. Wang and P. Liu, "Temporal management of RFID data." in *VLDB*, 2005, pp. 1128–1139.

[16] EPCglobal Inc., "EPCglobal tag data standards version 1.3." http://www.epcglobalinc.org/, Mar 2006.

[17] Y. Nie, R. Cocci, et al., "Spire: Efficient data interpretation and compression over RFID streams," Department of Computer Science, University of Massachusetts Amherst, Tech. Rep., 2009. [Online]. Available: http://spire.cs.umass.edu/pubs/tkde-all.pdf

[18] R. S. Barga, J. Goldstein, et al., "Consistent streaming through time: A vision for event stream processing." in *CIDR*, 2007, pp. 363–374.

[19] W. M. White, M. Riedewald, et al., "What is "next" in event processing?" in *PODS*, 2007, pp. 263–272.

[20] N. Khoussainova, M. Balazinska, and D. Suciu, "Towards correcting input data errors probabilistically using integrity constraints," in *MobiDE*, 2006, pp. 43–50.

[21] T. Tran, C. Sutton, et al., "Probabilistic inference over RFID streams in mobile environments," in *ICDE*, 2009, pp. 1096–1107.

[22] S. S. Chawathe, V. Krishnamurthy, et al., "Managing RFID data." in *VLDB*, 2004, pp. 1189–1195.

[23] E. Welbourne, N. Khoussainova, et al., "Cascadia: a system for specifying, detecting, and managing RFID events," in *MobiSys*, 2008, pp. 281–294.

[24] J. Rao, S. Doraiswamy, et al., "A deferred cleansing method for RFID data analytics," in *VLDB*, 2006, pp. 175–186.

[25] J. Xie, J. Yang, et al., "A sampling-based approach to information recovery," in *ICDE*, 2008, pp. 476–485.

[26] B. Kanagal and A. Deshpande, "Online filtering, smoothing and probabilistic modeling of streaming data," in *ICDE*, 2008, pp. 1160–1169.

Yanming Nie is currently pursuing the Ph.D. degree in the School of Computer Science at Northwestern Polytechnic University, China. He was a visiting student at the University of Massachusetts Amherst. His research interests include data stream processing, uncertain data management, and RFID data management.

Richard Cocci received the B.S. degree and the M.S. degree in Computer Science from the University of Massachusetts Amherst in 2006 and 2008, respectively. His research interests lie in RFID data management.

Zhao Cao received the B.S. degree in computer science from Beijing Institute of Technology, China, in 2004. He is currently pursuing a Ph.D in Computer Science at Beijing Institute of Technology. He was a visiting student at the University of Massachusetts, Amherst from September 2008 to March 2010.

Yanlei Diao is an Assistant Professor in the Department of Computer Science at the University of Massachusetts. Her research interests are in information architectures and data management systems. She was a recipient of the National Science Foundation Career Award and a finalist for the Microsoft Research New Faculty Fellowship. Her PhD dissertation won the 2006 ACM-SIGMOD Dissertation Award Honorable Mention.

Prashant Shenoy is currently a Professor of computer science at the University of Massachusetts. His research interests are in operating and distributed systems.He has been the recipient of the National Science Foundation Career Award, the IBM Faculty Development Award, the Lilly Foundation Teaching Fellowship, the UT Computer Science Best Dissertation Award, and an IIT Silver Medal.

Appendix

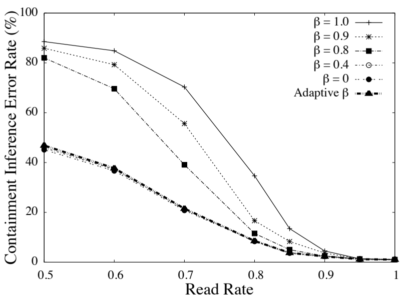
In this section, we provide additional results of our performance evaluation.

The adaptive β method in containment inference. In containment inference, the parameter β governs the beliefs between the recent history, which can be noisy, and the past edge confirmation, which may be obsolete. Fig. 11(a) to Fig. 11(d) compare the performance of using fixed values of β and the performance of an adaptive method that sets β based on the number of conflicting observations obtained, where each plot corresponds a different overlap rate (OR) used in the simulation. These plots confirm that the adaptive method performs the best for a wide range of read rates and overlap rates tested.

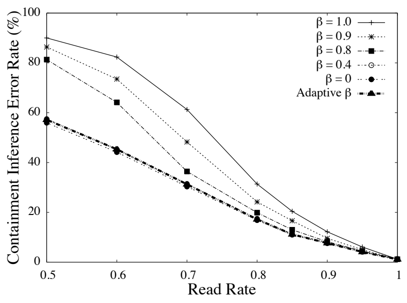
Recall that the major source of noise in containment inference is the co-location of multiple cases for an extended period of time on the shelves. To capture such noise, we further generated traces with different shelf reader frequencies. Fig. 11(e) shows the results. When the noise is high (e.g., the shelf reader frequency is once per sec), high β values ($\beta > 0.85$) give worse accuracy due to its emphasis on recent history. As the shelf reader frequency decreases, the noise from the shelf readings reduces, the recent history becomes more useful, and hence high β values improve their accuracy. The lower β values, favoring the past confirmation, tend to work well across different reader frequencies. Their accuracy degrades somewhat as fewer shelf readings are generated because the remaining readings mostly involve containment changes, making it harder to infer containment. Finally, the adaptive β algorithm is shown to work the best across different read frequencies.

Choosing the γ value in location inference. In location inference, the parameter γ weighs the belief of an object's recent locations (favored by low γ values) against the belief of its location inferred via containment (favored by high γ values). Fig. 14(a) to Fig. 12(d) report the location inference error rates as the γ value varies, where each plot shows multiple lines for different read rates (RR) and corresponds to a particular overlap rate (OR) used. Fig. 12(e) reports the effect of γ on location inference as the read frequency is varied (where the RR and OR parameters were set to their default values). These results confirm that there is a stable region of γ, [0.2, 0.4], where location inference provides the best performance.

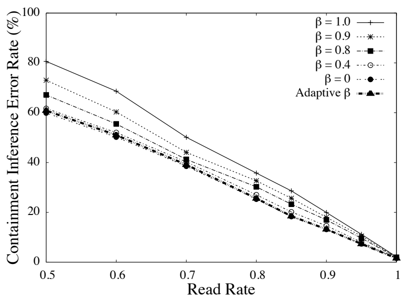
Choosing the θ value in location inference. In location inference, the parameter θ is the dampening factor on the belief of the continued existence of an unobserved object in a recently reported location. High θ values cause more quickly reduced belief, rendering it more likely to refer the object to be in the "unknown" location (e.g., in transit or missing). Fig. 13(a) to Fig. 13(d) report the location inference error rates as the θ value varies, where each plot shows multiple lines for different read rates (RR) and corresponds to a particular overlap rate (OR) used. Fig. 13(e) reports the effect of γ on location inference as the read frequency is varied (where the RR and OR parameters were set to their default values). These results confirm that there is a stable region of θ, [1, 2], where



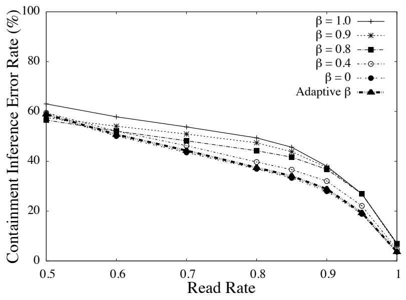
(a) Varying the read rate (OR=0)



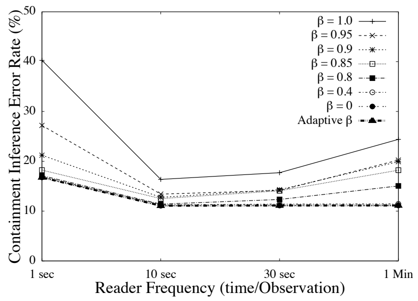
(b) Varying the read rate(OR=0.25)



(c) Varying the read rate (OR=0.5)



(d) Varying the read rate (OR=0.8)



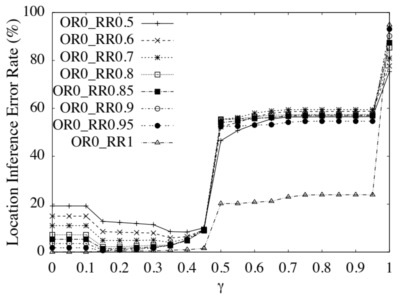
(e) Varying the read frequency

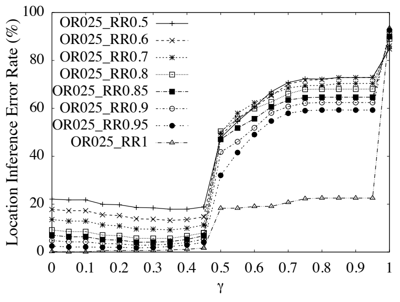
Fig. 11. Choosing appropriate values of the β parameter: comparison between fixed β values and the adaptive β method as the read rate (RR), the overlap rate (OR), and the read frequency vary.

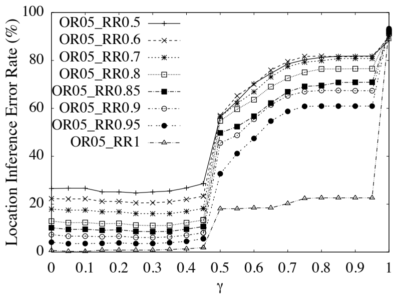
location inference provides the best performance.

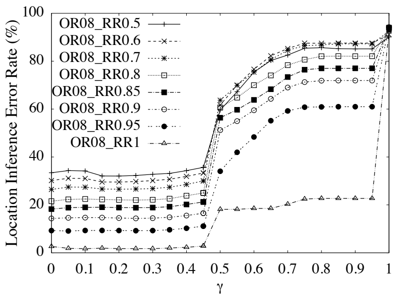
Choosing the k and λ values in edge confirmation. As described in Section IV-A, at each node we compute the normalized entropy h over all parent edges and check if h is lower than a threshold λ (supposed to be relatively small). To choose a universal λ for all nodes, it is helpful to model each node using a k-valued random variable. Fig. 14 verifies through different read rate and overlap rate combinations that all λ values under 0.9 offer the best accuracy, irrespective of the choice of k. When λ is set over 0.9 here, the accuracy degrades and becomes sensitive to the choice of k.

Results of anomaly detection. Fig. 15(a) exhibits similar trends as Fig. 8(c) and confirms that the θ values between 1 and 2 also work well for anomaly detection. We also measured the delay of anomaly detection as θ varies as shown in Fig. 15(b). In general, large θ values yield short delay whereas small θ values cause high delay. We observe that the region of 1 and 2 gives the delay as good as larger θ values. Hence the region of 1 and 2 for θ is shown to provide good performance for both accuracy and detection delay.


(a) Varying the read rate (OR=0)


(b) Varying the read rate(OR=0.25)


(c) Varying the read rate (OR=0.5)


(d) Varying the read rate (OR=0.8)

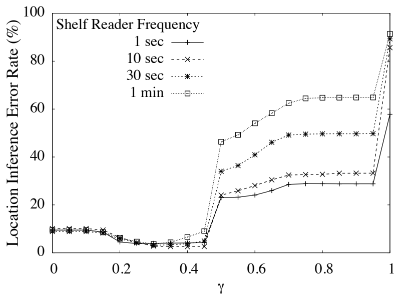
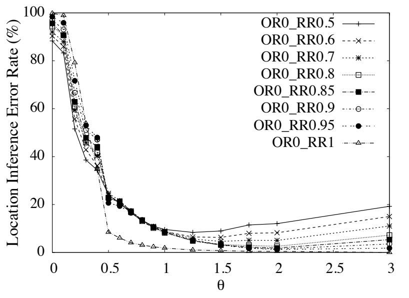
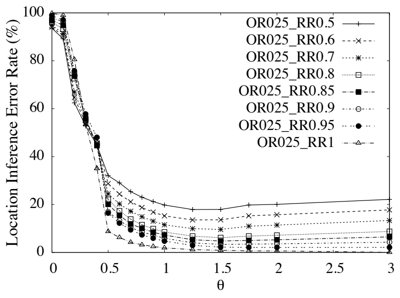
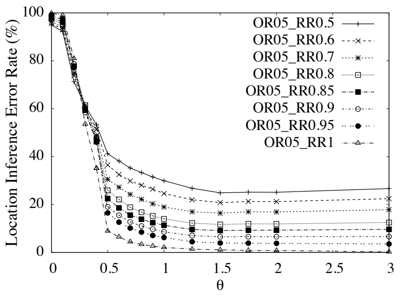
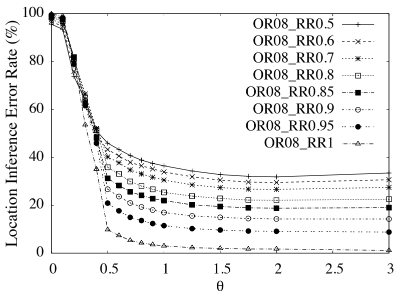

(e) Varying the read frequency

Fig. 12. Choosing appropriates values of the γ parameter as the read rate (RR), the overlap rate (OR), and the read frequency vary.


(a) Varying the read rate (OR=0)


(b) Varying the read rate(OR=0.25)


(c) Varying the read rate (OR=0.5)


(d) Varying the read rate (OR=0.8)

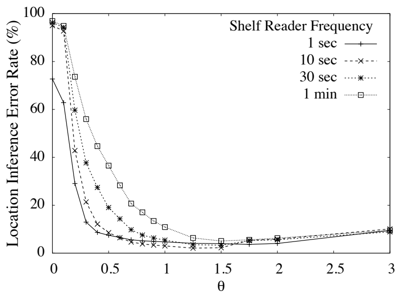
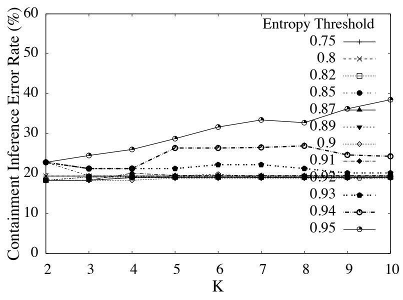
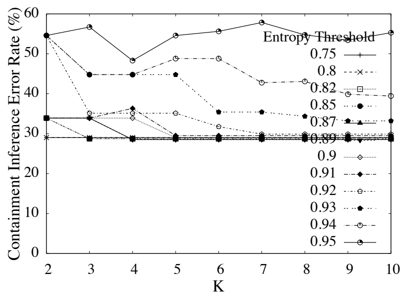

(e) Varying the read frequency

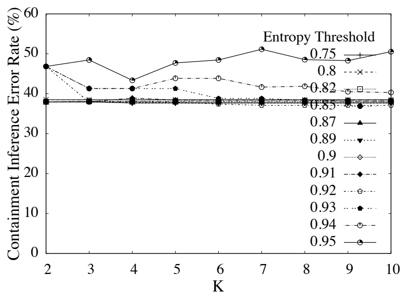
Fig. 13. Choosing appropriates values of the θ parameter as the read rate (RR), the overlap rate (OR), and the read frequency vary.



(a) Varying k and λ (RR=0.85, OR=0.5)

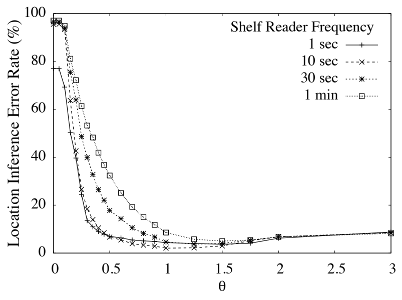


(b) Varying k and λ (RR=0.7, OR=0.25)

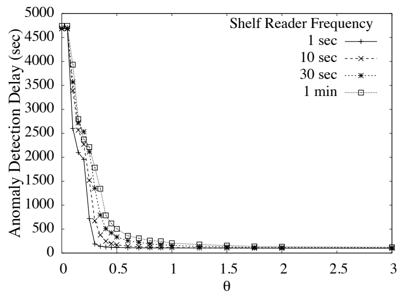


(c) Varying k and λ (RR=0.7, OR=0.5)

Fig. 14. Effects of choosing the top-k edges of the highest probabilities and the threshold λ on edge confirmation.



(a) Inference error (1 anomaly/100 sec)



(b) Detection Delay (1 Anomaly/100 sec)

Fig. 15. Accuracy and delay of anomaly detection.