SPIRE: Efficient Data Interpretation and Compression over RFID Streams

Richard Cocci†, Yanming Nie‡, 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 interpretation 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 interpretation, it enables online compression that recognizes and removes redundant information from the output stream of this substrate. We have implemented a prototype of our interpretation and compression substrate and evaluated it using synthetic RFID streams emulating a warehouse environment. Results of a detailed performance study show that our data interpretation 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 [13], healthcare [13], pharmaceuticals [13], postal services [16], and surveillance [17] 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 [34], [23], [6], [7], [26], [27], 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 [9], [18]. This is largely due to the intrinsic sensitivity of radio frequencies (RFs) to environmental factors such as occluding metal objects [10] and contention among tags [11]. 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 [29]. 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 [12], [18], [19] 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 [14], [24] 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 [15] or employ application-specific logic [30]. 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 ***interpretation and compression substrate*** over RFID data streams. This substrate enables accurate interpretation 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 interpretation, 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 [3].

significantly reduces data volume, thereby expediting processing and reducing data transfer costs.

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 interpretation and compression substrate in a prototype system and evaluated it using RFID streams from a simulated warehouse environment. Our results show that our data interpretation techniques achieve error rates around or below 10% for location estimates for a wide range of RFID read rates, and within 10% 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. Finally, we compare our system with SMURF [19], 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 is shown to outperform 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 interpretation problem is to construct an approximate yet accurate estimate of the state of the world based on the observations thus far. We define an approximation using functions $resides$ and $contained$ that for given arguments, return probabilistic values representing the likelihood of the function being true. Then the data interpretation 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 $argmax_k\ resides(o_i, l_k, now)$, and
2) the most likely container of the object, denoted by $argmax_{j,k}\ contained(o_i, o_j, l_k, now)$.

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

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

A running example. A warehouse scenario is depicted in Fig. 1, where RFID readers are installed above the loading dock, the conveyor belt, and the packaging area. At time t=1, the reader at the loading dock reports objects 1 to 6, denoted by the shaded nodes. These nodes are arranged according to the packaging levels that the reported tag ids indicate [8]. Object 7 is also present but was missed by the reader, denoted by an unshaded node, i.e., a *missed reading*. Containment between objects, depicted by the dashed edges, is not reported by the readings and often uncertain. Examples of *ambiguous containment* are the containers of items 4, 5, 6, which can be either case 2 or case 3 based on the readings received.

At time t=2, case 2 is scanned individually on the conveyor belt. It is now possible to confirm the containment between the case and its item(s) if the domain knowledge of the deployment reveals such *special readers* that scan containers

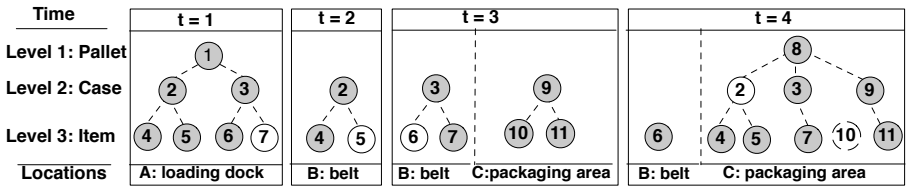


Fig. 1. A sequence of observations of RFID-tagged objects in a warehouse setting. A shaded node represents an observed object and an unshaded node denotes an unobserved one. A dashed edge represents a containment relationship between objects, which cannot be directly observed.

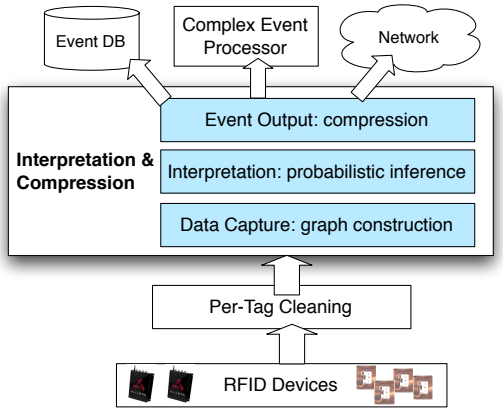


Fig. 2. Architecture of the SPIRE system.

of a particular type only one at time. At t=3, case 3 is scanned on the belt and a new case 9 is read in the packaging area.

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

System architecture. The SPIRE system employs a data interpretation 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 *interpretation* 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.

Finally, note that this substrate runs on a low-level device data cleaning module such as [18]. The only required functionality of device data cleaning for this work is deduplication, which corrects for duplicate readings caused by the close proximity of the readers. At each time step, deduplication detects the tags that are read by several nearby readers and assigns each tag to the reader that read the tag most recently.

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 the physical world. In a supply-chain environment, the RFID standard [8] requires that an object have a packaging level of an *item*, *case*, or a *pallet*, and the packaging level information be encoded in the tag ID of the object's tag. Given such information, our graph is arranged into layers, with one layer for each packaging level. In addition, each node has a color that denotes its location; a node may be assigned one of $(L-1)$ colors, one for each known location. A node is uncolored if its location is currently unknown. The node colors are updated using the stream of readings in each epoch (the color of a node is the color of the location where it is observed by an RFID reader). If an object is not read by any reader in a particular epoch, its node becomes uncolored. However, uncolored nodes retain memory of their most recent color and the observation time denoted by (`recent color`, `seen at`).

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

To enable probabilistic analysis, the 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. Furthermore, each node records the `confirmed parent`, that is, the last

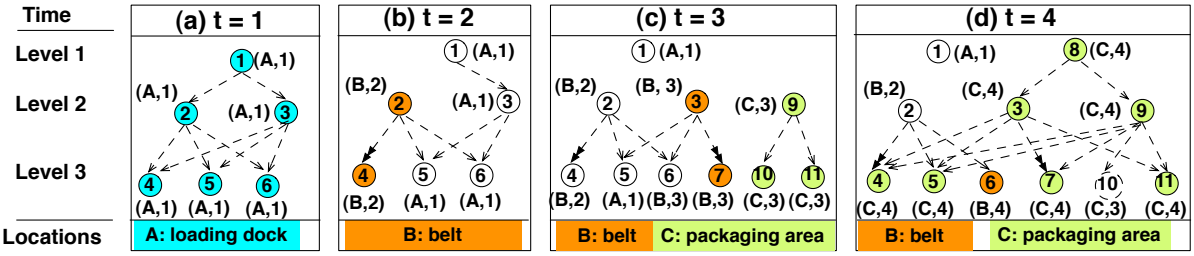


Fig. 3. Evolution of the time-varying colored graph model as RFID readings arrive in each epoch.

confirmed parent node (container) revealed by a special reader, the time of confirmation, and the number of conflicting observations obtained thus far. Among all incoming edges from the possible parents, at most one edge can be the confirmed edge, denoted by an edge with double arrows in Fig. 3.

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, shown in Fig. 4, 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^*. An important feature of the algorithm is that it proceeds incrementally as readings arrive from each reader and guarantees a consistent output G^* after seeing the readings from all readers in an epoch. This ensures that the algorithm works even when the various readers are coarsely synchronized in time.

Given the set of readings R_k from each reader, the graph update procedure proceeds in four steps, as shown in Fig. 4.

Step 1. *Create and color nodes (lines 2-6):* If a new object is observed for the first time, a new node is created in the graph. For each observed object, the corresponding node is colored with the color of the location in which it was observed. Fig. 5(a) shows the result of step 1 when the graph update procedure is applied at time t=4 to the previous graph (Fig. 3(c)), first with the readings from the conveyor belt, which has the color B, and then with the readings from the packaging area, which has the color C.

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

This step potentially requires 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. 5(b) illustrates the result of this step when

```
procedure graph_update(G, Rk)
G: current graph, Rk: set of readings from reader k
begin
1.    C = location of the reader k

2.    foreach reading r ∈ Rk do                          // create and color nodes
3.      v = node in G corresponding to the object of the reading r
4.  (1) if v == null then
5.        v = a new node created in G for this object
6.      v.recent_color = C, v.seen_at = now

7.    foreach packaging level L (starting from 1) present in Rk do
8.      foreach each node v in color C at level L do
9.        if C is a new color of v then                  // create new edges
10.         above = closest level above L containing nodes colored in C
11    (2)   below = closest level below L containing nodes colored in C
12.         create edges from the nodes in level above to v if not existent
13.         create edges from v to the nodes in level below if not existent
14.     foreach edge e incident to v do
15.       if e was created in a previous epoch then    // remove edges
16.         v' = the other node of e
17.         if v' has a different color from v then
18.   (3)     drop the edge e from G
19.         if reader k confirms that the child node of e is a top level container
              or has a different confirmed parent edge than e then
20.         drop the edge e from G
21.       if e.update_time < now then                  // update statistics
22.         right shift e.recent_history to expire old information
23.         if v' has the same color as v then
24.           e.recent_colocations[0] = True
25.           if v' is a child of v and reader k confirm that v contains v' then
26.   (4)       v'.set_confirmed_parent(e, now)
27.         else
28.           e.recent_colocations[0] = False
29.           if e was set as the confirmed edge of v (or v') before then
30.             v(v').record_conflict(e);
31.       e.update_time = now
end
```

Fig. 4. Algorithm for stream-driven graph update.

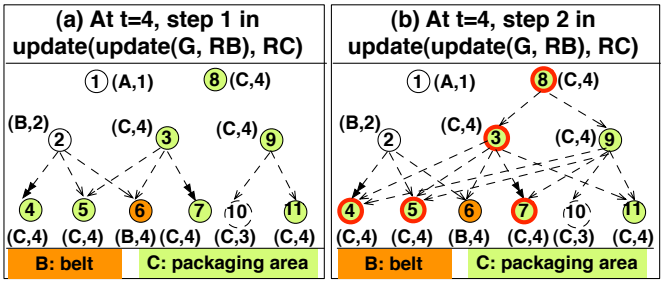


Fig. 5. Intermediate steps of the graph update procedure.

the readings from the packaging area are applied to the graph in Fig. 5(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 8 and 3, 8 and 9, 3 and 11, and so on.

Step 3. *Remove edges (lines 16-20):* 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 the corresponding nodes have different colors—this happens when two previously co-located objects are now reported in different locations. For example, the edge between nodes 3 and 6 in Fig. 5(b) is removed in this step as these nodes are assigned different colors. The resulting graph is that shown in Fig. 3(d).

Similarly, edges can be dropped when special readers confirm a top-level container and its contained objects, which allows us to eliminate other possibilities. For example, if a case is confirmed to be a top-level container, any parent edge of the case can be dropped. In addition, if the case is also confirmed to be the container of any read item, then all other edges between each read item and other possible cases can be dropped. Special readers such as belt readers that read cases one at a time allow us to prune unnecessary edges. Edge pruning using a belt reader is depicted in Fig. 3(b), where we remove the edge from node 1 to node 2 since node 2 is known to be the top-level container, and the edge from node 3 to node 4 since node 4 now has node 2 as the confirmed container.

Step 4. *Update edge statistics (lines 22-31):* This step updates statistics of the edges that have at least one node colored in step 1. Given an edge e, if the two linked nodes have the same color, `recent co-locations` of e is updated by setting the most recent bit to True; further, if a reader is able to confirm the containment denoted by e, it is used to update the `confirmed parent` of the child node. If one of the linked nodes is uncolored, the most recent bit of `recent co-locations` is set to False. In this case, we also check if e was set as the confirmed parent edge of the child node, and if so count the current observation as a conflicting observation of the confirmation. These statistics play a key role in containment inference as we shall show shortly.

Complexity of graph update. 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, ..., R_K is the sum of the individual costs $cost(R_k)$, $1 \leq k \leq K$. For each reading set R_k, only the colored nodes and their incident edges are processed, and its update cost can be analyzed by considering the four steps of the update procedure in turn:

Step 1: Given the reading set R_k, the cost of coloring nodes is simply $|R_k|$. Step 2: The cost of creating edges is bounded by the largest size of the bipartite graph that the nodes in R_k cover, that is, $(|R_k|/2)^2$. Steps 3 and 4: As Fig. 4 shows, these two steps share the process of examining every edge linked to a colored node. Since these edges may connect to an uncolored node or a node of a different color, they cannot be bounded only using $|R_k|$. However, if we combine steps 3 and 4 across all readers, the following observations hold:

- If an edge is assigned two different node colors or has one node colored but the other uncolored, it must have existed in the input graph G. Then in the current epoch, it is visited at almost twice, once from each incident colored node. To capture this cost, let us use $\pi_{R_1,...,R_K}(G)$ to denote the *projection* of the input graph onto the subset of edges that are linked to at least one node colored in the current epoch. Then the cost of accessing each of these edges at almost twice is $2 \cdot \pi_{R_1,...,R_K}(G)$.
- Next, consider edges that have the same color for the two nodes. Given a reading set R_k, the maximum number of edges that may have both nodes in R_k (hence assigned the same color) is $(|R_k|/2)^2$. Through careful programming, we can access each of these edges only once, i.e., from the node with a higher packaging level. So the cost of accessing these edges is bounded by $(|R_k|/2)^2$.

So, the total cost of graph update for all readers is:

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

This yields a complexity $O(\sum_k |R_k|^2 + |\pi_{R_1,...,R_K}(G)|)$ for the complete graph update in an epoch. This indicates that the graph update procedure includes a cost that is no more than the input graph size and some local costs quadratic in the size of the subgraph colored by each reader.

IV. Data Interpretation

The graph constructed from the data capture step can result in nodes that are uncolored or possess multiple parent nodes. The data interpretation step estimates the most likely location of an unreported (uncolored) object and the most likely container (parent) of an (either reported or unreported) object. We present a probabilistic technique that includes edge inference to address ambiguous containment, node inference to address unknown locations, and an iterative algorithm 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. Computing the probability values requires the use of history that includes (i) the recent history of co-locations, as represented by the bit-vector `recent co-locations`, and (ii) the last confirmed parent of v by a special reader, captured in `confirmed parent`. Such use of history makes edge inference less sensitive to missed readings at present time.

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

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

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

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

Step 2. *Compute Probabilities:* The next step builds a probability distribution across all incoming edges of node v.

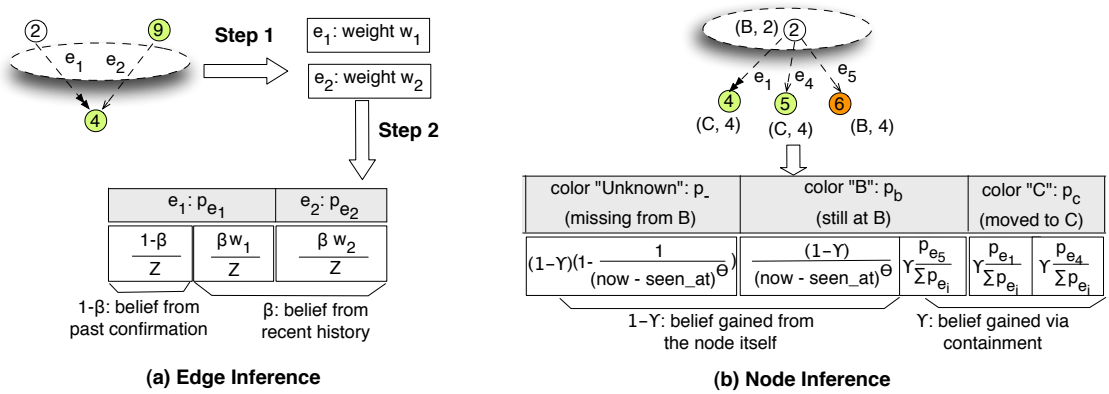


Fig. 6. Examples of edge inference and node inference.

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:

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

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 across all incoming edges that yields the final distribution. Fig. 6(a) shows such distribution across the 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 based on the number of conflicting observations since the last confirmation of an object's container by a special reader. A simple heuristic exploiting this idea is evaluated in Section VI.

B. Node inference

Node inference is applied to an uncolored node v—an object with an unknown location—and attempts to infer the most likely location of the object or confirm its absence from any known location. The key challenge in node inference arises from a three-way tradeoff among *continued stay*, *movement to a new location*, and *absence from any known location*. These situations are depicted in Fig. 6(b) for node 2 at time t=4. This object was last seen in location B at time t=2 and has a few possibilities for its current location: it is still in location B but the reading in this location was missed (continued stay); it moved to location C with its contained objects and its reading was missed in C (movement to a new location); it disappeared from B and its current location is unclear (absence from any known location).

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

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

$$p_c(v) = (1-\gamma)\frac{\delta(v,c)}{(now - seen\_at)^\theta} + \gamma \sum_{e_i \to c} \frac{p_{e_i}}{Z_2} \quad (3)$$

$$\delta(v,c) = \begin{cases} 1, \text{ c is the most recent color of v} \\ 0, \text{ otherwise} \end{cases}, \; Z_2 = \sum_{e_i \to c_j} p_{e_i}.$$

Here $\delta(v,c)$ is an indicator function that is 1 for the most recent color of v and 0 otherwise, and the parameter θ controls the rate of fading of the most recent color. $e_i \to c$ means that the edge e_i propagates the color c 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:

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

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

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 $(now - seen\_at)^{-\theta}$, further adjusts the distribution of the probability mass between the fading color and the "unknown" color. We again quantify the sensitivity of node inference to these parameters 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

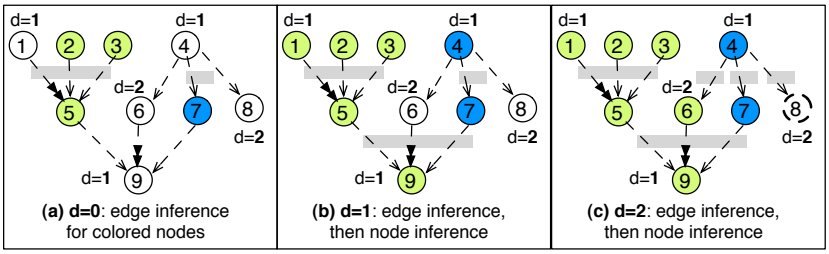


Fig. 7. Illustration of iterative inference across the graph in increasing distance from the colored nodes.

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

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

For ease of composition, we classify nodes based on their closest distance, d, from a colored node in the graph. The iterative inference algorithm runs in increasing value of d. Fig. 7 illustrates this process for an example graph (the pseudo code is omitted due to space constraints).

The algorithm first runs edge inference for the nodes with $d = 0$. These nodes are the observed objects and are colored in two colors, dark (blue) and light (green), in the example shown in Fig. 7(a). Edge inference is performed for these nodes to estimate their most likely parents. A gray bar represent the edges considered in the edge inference at a node. Next, the algorithm considers the nodes labeled with $d = 1$ (i.e., unobserved) and runs edge inference, followed by node inference, to infer both the likely parent and the likely location for each node. In the example shown in Fig. 7(b), nodes 1 and 9 gain the light (green) color and node 4 gains the dark (blue) color from their node inference. Finally, the algorithm runs for the nodes labeled with $d = 2$, shown in Fig. 7(c). At this point, node 6 gains the light (green) color and node 8 is identified as a missing object.

Complexity analysis. The complexity of the iterative inference algorithm is bounded by the number of edges examined. Given that each edge can be visited at most twice, once from each node, the overall complexity is $O(|E|)$.

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.

D. Partial and Complete Inference

An important issue that needs to be resolved in inference is that RFID readers can read at different frequencies. In a typical warehouse, for instance, belt readers may read once every second while shelf readers may read once every 10 seconds. Suppose that an object moved to a shelf at time 5 but is not read by the shelf reader until time 10. Then the graph models obtained for time units (also called *epochs*) 6, 7, 8, and 9 do not present a complete view of the physical world. If we run inference in these epochs, the inferred location for the object is likely to the "unknown" location, which is different from the object's true location, and such inference work is wasted.

To address this issue, our system uses both partial inference (over a subset of the graph) and complete inference (over the entire graph) as follows: We first obtain the read frequencies of all the readers from a system configuration file and calculate the least common multiple, M, of those frequencies. Given the epochs numbered sequentially from 1, we perform complete inference in those epochs whose numbers are a multiple of M, and partial inference otherwise.

Complete inference applies the iterative algorithm to the entire graph as described above. Partial inference modifies the complete inference algorithm in two aspects: (1) it restricts inference to a subset of the graph that contains only those nodes that are at most l hops away from the colored nodes (by default, $l = 1$), and (2) if the inferred color of an uncolored node is "unknown", we withhold the inference result from output since this result is based on the readings from only a subset of the locations. In this case, it is quite likely that the object is in a known location but the reader placed in that location is not reading, and the inference algorithm cannot find useful information elsewhere. Hence, withholding such inference results avoids outputting misleading information, but the inference work for this object is wasted. Restricting partial inference to the l-hop subgraph helps limit the amount of wasted work—the intuition is that the further we move away from the colored nodes, the more likely we yield the "unknown" result in location reference. Finally, in a later epoch when all readers read, the complete inference algorithm will make a better-informed decision for the object with a previously unknown location.

TABLE I
POSSIBLE CONFLICTS IN INFERENCE RESULTS AND THEIR RESOLUTIONS

Rule #	Parent and Child Locations	Method for Resolving the Conflict Between the Differing Locations
I	Parent: Observed Child: Inferred	Give preference to the containment and override the child's location to match the parent's location.
II	Parent: Inferred Child: Observed	Poll the children. If a majority agree on location, change the parent's location to the consensus vote. If no majority, do not alter the parent's location. End all of the containments still in conflict.
III	Parent: Inferred Child: Inferred	Update parent's location using the majority vote as above. Then give priority to the containment and override the child's location.

E. Conflicts between Location and Containment Inference

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. Fig. 7(c) shows such a conflict between containment and location inference results: node 4 is the only parent of node 6 and hence is chosen to represent the container. However, these nodes are inferred with different colors, i.e., reported to be in different locations. In SPIRE, we do not prune the edge between the two nodes with conflicting colors if at least one of the colors is inferred, because an inferred color can be uncertain. Instead, we keep the edge in the graph for future use and resolve conflicting results in a post-processing step after inference for the sake of output.

The general guideline that we apply to resolve conflicts is to give priority to a containment relationship over an inferred location. This is mainly because the containment is often confirmed by a special reader and hence carries valuable information. Specifically, we consider the patterns of node coloring of an edge chosen to represent the containment between two nodes. If the two nodes have different colors, at least one of them must be inferred (otherwise an edge with both nodes observed but colored differently would have been dropped in the graph update described in Section III-B.) Table I details three patterns of node coloring for such edges:

Rule I states that if the parent node is observed by a reader in the current epoch whereas the child node is not, override the child's estimated location with the parent's location so as to be consistent with the inferred containment.

Rules II and III jointly handle the cases where the parent node is not observed. Even if some of the child nodes are observed, it would be imprudent to immediately override the parent's location because the parent has a one-to-many relationship to the child nodes. Instead, to minimize the conflicting information between the parent and all its children, we perform a polling among the children. If a majority of the children agree on the location, we override the parent's location with the majority vote, otherwise we leave the parent's location unchanged. Afterwards, we enumerate all children and resolve remaining conflicts as follows: If a child is observed and has a different location from the parent (Rule II), we terminate their containment relationship, i.e., reporting that the child does not have a container. If a child has only an inferred location and differs from the parent's location (Rule III), we override the child's location to give priority to the containment.

A final note is that due to the need of polling all the children of a node, conflict resolution cannot be performed as part of the iterative inference. This is because when we process the node in the d^{th} iteration, there is no guarantee that all its children have been processed. Therefore, conflict resolution has to be treated as post-processing. For efficiency, our implementation merges this step into the output module (detailed in the next section) in which all inference results are read once, from parent to children, to generate the output.

V. STREAM OUTPUT WITH COMPRESSION

The output module takes the results of data interpretation and transforms them into a compressed event stream for output. 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.

The SPIRE system employs two compression techniques that take the results of location and containment inference and output a compressed data stream. Compared to the raw RFID stream, these compression techniques are *lossless* in that the observed objects are always truthfully reflected in the output. Since data interpretation can add location information for unobserved objects and containment information not available in the input stream, the compressed output stream may contain richer information yet with a reduced data volume. In the rest of this section, we describe the data format of a compressed stream and then the two compression techniques in detail.

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* [1]. 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, $V_e = \infty$)
- EndLocation(object, location, V_s, V_e)
- StartContainment(object, container, V_s, $V_e = \infty$)
- 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 sets V_e with an appropriate timestamp. Similarly, start and end containment messages are also pairs to encapsulate the time period of a containment relationship. Missing messages are singletons that are 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.

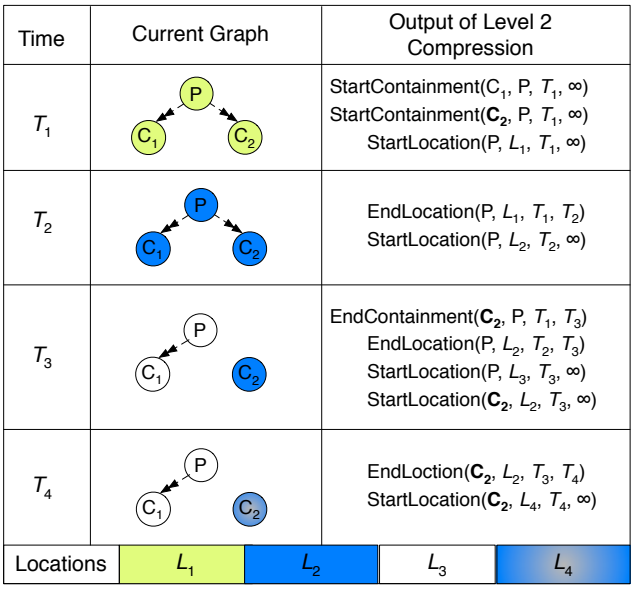


Fig. 8. An example of level-2 compression for a group of objects.

B. Range Compression

Our first compression method, which we refer to as range compression or level-1 compression in our system, 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 range compression method is straightforward: We compare an object's newly inferred state (either location or containment) to its previously reported state. If the state changes (except when the object has become missing), we output an end message to complete the event for the previous state and then a start message for the event that signals the new 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 the following properties. (i) *Independent location and containment output:* The location compression and containment compression are performed separately. Hence, it is possible to split the output into separate location update and containment update streams, and to suppress the output of one stream if it is not needed. (ii) *Queriable streams:* Complete location and containment information for each object, presented in a well-formed and ordered manner, makes the result stream of range compression directly queriable using recently developed event processors such as [1], [32].

C. Location Compression using Containment

Our second compression method, referred to as level-2 compression, uses the additional knowledge that a stable containment relationship allows further suppression of location readings of child objects. That is, if the containment relationship does not change, the child's location is identical to that of the parent and thus can be omitted from output. The benefit of dosing so is to minimize the location output to only the location of top-level containers. This compression is also 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 sequence of events generated using level-2 compression is shown in Fig. 8. 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 object 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 output as soon as they are detected since C_2 is no longer contained. In comparison, C_1 is still contained in P so only location updates are sent for P due to level 2 compression for C_1.

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 certain objects in a certain period of time as requested by the queries.

The routine works as follows. For each time step, it first

TABLE II
PARAMETERS USED FOR GENERATING RFID STREAMS.

Parameter	Value(s) used
Duration of simulation	3 - 24 hours
Rate of pallet injection	1 / 4 - 600 seconds
Cases per pallet	5 - 8
Items per case	20
Read rate of readers	0.5 - 1
Non-shelf reader frequency (fixed)	2 (interrogations) per sec
Shelf reader frequency (variable)	1 / sec to 1 / min

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 the example in Fig. 8. At time T_3, since object C_2 was no longer contained in P, 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). Hence, the SartLocation event at T_3 that reports the same object at the same location becomes a duplicate. Our decompression routine will remove all such duplicates.

VI. PERFORMANCE EVALUATION

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

A. Experimental Setup

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 II. Note that two parameters are used to control the read frequencies of shelf readers and non-shelf readers separately. This design allows flexible settings of the simulation where items may stay on shelves for hours and shelf readers may read less frequently than other readers.

Synthetic RFID data streams generated by the simulator are fed to our interpretation and compression substrate. Data interpretation is performed in every epoch (whose length 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 Interpretation

We first evaluate the accuracy of our inference techniques for data interpretation. We created data streams with 6 pallets injected per hour, 5 cases per pallet, an average shelving period of 1 hour, and a total simulation time of 3 hours. The default read rate for all the readers is 0.85. The default read frequency for shelf readers is once every minute. An inference result is marked as an error if it is inconsistent with the ground truth.

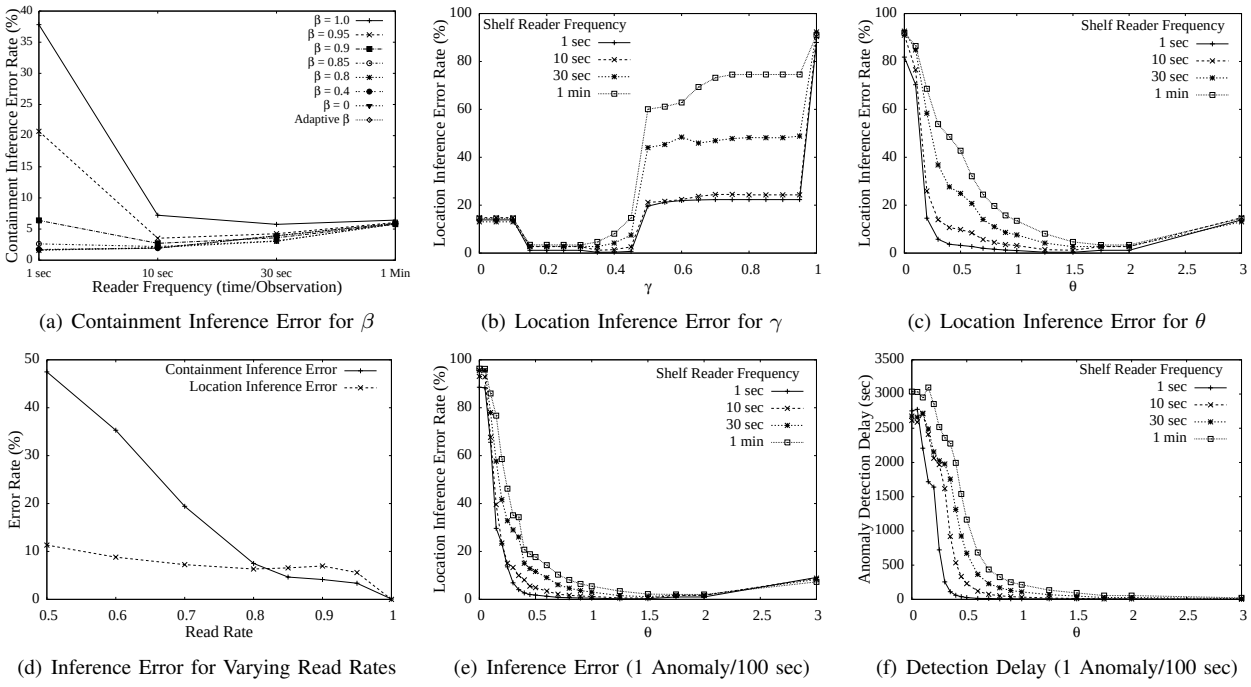
Expt 1: Containment Inference. We first study the effects of the edge inference parameters, β, S, and α, shown in Equations 1 and 2, on containment inference. Our results show that the two parameters, S and α, on the recent history of co-locations can be tuned easily: The size of the history, S, limits 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 α=0 in the rest of experiments.

The parameter β governs the beliefs between the recent history, which can be noisy, and the past confirmation, which may be obsolete. In our simulation, the major source of noise in containment inference is the co-location of multiple cases on the same shelf. To capture such noise, we generated traces with different shelf reader frequencies, and for each trace ran inference by varying β from 0 to 1, where β=1 gives all the weight to recent history and β=0 does the opposite.

As shown in Fig. 9(a), 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, we consider a simple adaptive algorithm that sets β to be the ratio between the number of instances that only one of the object and its confirmed container is read and the number of instances that any of them or both of them are read. This algorithm is shown to work as well as the low β values.

Expt 2: Location Inference. Location inference uses the node inference method defined in Equations 3 and 4. We now study the effects of two parameters on location inference.

The parameter γ weighs the belief of an object's last observed location (favored by low γ values) against the belief of its location inferred via containment (favored by high γ values). Fig. 9(b) shows the results for varied γ values. Very



(a) Containment Inference Error for β (b) Location Inference Error for γ (c) Location Inference Error for θ

(d) Inference Error for Varying Read Rates (e) Inference Error (1 Anomaly/100 sec) (f) Detection Delay (1 Anomaly/100 sec)

Fig. 9. Containment and location inference results.

low γ values place most emphasis on the last observation (the fading color). 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 relationships of an object, which can be an unreliable source of information when containment is changing or uncertain. Overall, we observe γ values in the range of 0.15 to 0.45 to be favorable because they offer a balance between an object's last observation and containment relationships, with some more weight on the former. Further, we observe that the traces with frequent shelf readings (e.g., once per sec) benefit more from containment information. This is because in the shelf area containment relationships do not change so the previously confirmed containers (by the belt reader) are helpful in determining object locations. When shelf readings are less frequent (e.g., once per minute), a larger fraction of the trace contains the readings produced during containment changes, so emphasizing containment penalizes the performance.

The parameter θ is the dampening factor on the belief of the continued existence of an unobserved object in its last reported location. High θ values cause more quickly reduced belief, rendering it more likely to refer the object to be in the "unknown" location (e.g., in transit or missing). Fig. 9(c) shows the results with varied θ values. As θ increases, the error rate quickly declines from over 90%, flattens in the mid-range, and degrades again with higher values. The initial decline occurs because, with very low θ values, the inference takes too long to reduce its belief of the continued presence of an object when if the object left some time ago. The deterioration with high θ values occurs because the inference becomes too eager to drop its belief of continued existence and identify an object as being away after a few missed readings. Similar trends are observed for different read frequencies, with the effect more pronounced for high frequency readers.

The above results provide insights into tuning inference parameters for common workloads in our target application. In the rest of the study, we set β = 0.4, γ = 0.4, and θ = 1.25, unless stated otherwise.

Expt 3: Sensitivity to Read Rate. The next experiment studies the sensitivity of our inference methods to the read rate. We varied the read rate uniformly for all readers. The shelf reader frequency was set to 1 reading per minute. As Fig. 9(d) shows, the error rates of both containment and location inference stay below 10% for read rates between 0.8 and 1. As the read rate decreases, the location inference stays fairly accurate due to its appropriate parameter settings to exploit the last reported location. The containment inference, however, loses its accuracy due to both the loss of containment confirmation provided by belt readers and lack of consistent observations in the recent history.

Expt 4: Accuracy and Delay of Anomaly Detection. The traces used so far have not captured any abnormal behaviors, which are expected to be rare but of significant interest to the application. In this experiment, we simulated unexpected removals of objects from the warehouse, representing theft or misplacement, at a rate of 1 removal every 100 seconds with random selection from all objects. We report on the inference error rate as well as the delay of anomaly detection in Fig. 9(e) and 9(f) as the most relevant parameter, θ, is varied.

Regarding the error rate, Fig. 9(e) exhibits similar trends as Fig. 9(c) and confirms that the θ values between 1 and 2 also work well for anomaly detection. Fig. 9(f) shows a somewhat different trend regarding the delay of anomaly detection. For a shorter delay, higher values of θ are preferred to more quickly decay the belief of the continued presence of an object. This is especially true for low reader frequencies; it otherwise takes too long to wait for the next reading, adjust the belief, and

TABLE III
COSTS OF UPDATE AND INFERENCE OPERATIONS (SEC)

Num. Objects	Update	Inference	Total
25344	0.00256	0.07080	0.07336
54915	0.00684	0.15617	0.16301
75275	0.00967	0.22159	0.23126
95049	0.01203	0.29139	0.30342
135509	0.01557	0.43624	0.45181
154893	0.01656	0.50930	0.52586
174923	0.01689	0.58413	0.60102

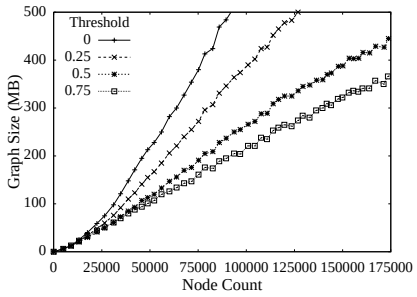


Fig. 10. Memory usage for graphs with different node counts.

finally recognize the missing object. For instance, to ensure a detection delay of 120 seconds, we need to set θ=1.5 when the shelf readers read once a minute as opposed to θ=0.35 when they read once a second. Combining both the error rate and detection delay, we observe that the 1 to 2 range of θ values remains to be the optimal choice.

C. Efficiency of Data Interpretation

We next evaluate the efficiency of data interpretation in both memory usage and processing speed. To do so, we used a higher pallet injection rate of 1 every 4 seconds, and varied the number of cases per pallet and the simulation time to obtain graphs of different sizes (node counts). 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 2GB.

Expt 5: Processing speed. Table III reports the processing time for graphs of different sizes. With the increasing node count, 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, the costs of both update and inference are much less than a second, with the inference cost being a more significant cost. The total cost is reported in the last column. As is shown, the total cost is 0.6 second for the largest graph size, which uses an already high injection rate of one pallet every four seconds. These results show that our data interpretation techniques can keep up with high-volume RFID streams, which is a key requirement of data processing in this setting.

Expt 6: Memory Usage. The memory usage in data interpretation is dominated by the size of the graph. In this experiment, we measured the memory usage with varied node counts. 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. 2 but before normalization and thus is insensitive to the presence of other edges. To explore this factor, we applied a threshold for pruning edges and varied it from 0 to 0.75.

Fig. 10 shows that as the node count grows, the memory usage increases fast without edge pruning but less so with increased thresholds for pruning. With a threshold of 0.5, pruning is able to keep the size of the graph under 500 MB, even with 175,000 objects present in the system. In addition, the memory growth of using the 0.5 and 0.75 thresholds is shown to be linear, rather than the worse-case quadratic edge 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 with or without pruning), 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.

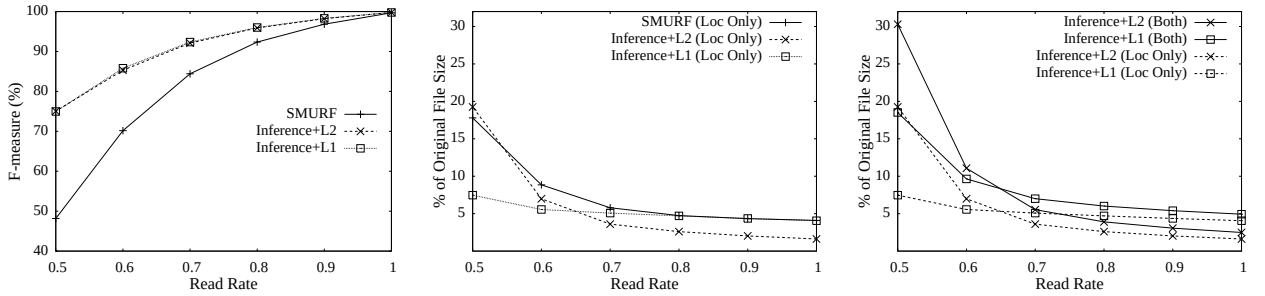
D. Accuracy and Data Reduction of the Output Event Stream

After inference, our system translates inference results into a stream of output events, first using conflict resolution (Section IV-E) and then using level 1 compression (Section V-B) or level 2 compression (Section V-C). In this set of experiments, we evaluate the accuracy and data reduction of our output event stream. We further compare our system with SMURF [19], a state of the art RFID data cleaning system. SMURF applies smoothing with an adaptive window size to mitigate the missed reading problem. To enable a comparison to our output, we extend SMURF by using the static reader locations to infer the object locations as readings are smoothed in, and then applying level 1 compression to produce a compressed event stream. SMURF, however, does not support containment inference and hence offers no containment information in the output or level 2 compression—containment inference is unique to our system. For this reason, 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. The data sizes ranged from 56 MB for the 0.5 read rate to 111 MB for a perfect read rate.

Expt 7: Accuracy of Output Events. The accuracy of the output event stream accounts for the effect of conflict resolution, which may change some inference results to ensure consistency in output. In addition, the accuracy metric used 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. Borrowing concepts from the Information Retrieval field, 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 *F-measure* = 2*precision*recall/(precision+recall).

Fig. 11(a) compares SMURF and our system in F-measure. It shows that our system significantly outperforms SMURF



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

Fig. 11. Accuracy and data reduction of the output event stream.

especially in the low read rate range. This is because SMURF is simply a smoothing technique; it can smooth in readings in certain cases when the object is still in the original location but not detected by its reader. However, it does not work well when an object has several consecutive missed readings, and hence believes this object to be away from its location. In contrast, our system exploits stable containment in addition to the fading color of the previous location. As such, our system would use the location information of the container or contained objects to infer this object's location, thereby overcoming the problem of consecutive missed readings. Finally, note that the two compression techniques have the same event error rate as compression itself does not affect accuracy.

Expt 8: Compression Ratio. To examine the data reduction 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., the compression ratio).

We first include only location events in the output, as shown in Fig. 11(b), to see how SMURF's smoothing algorithm and our inference algorithm impact the compression ratios. From Fig. 11(b), we see that SMURF is comparable to our system with level 1 compression for high read rates but becomes much worse when the read rate is low (e.g., below 0.7). This is because given a few missed readings SMURF would prematurely output an event reporting an object to be away from its previous location, and then given new readings sends another event reinstating the object's existence. Such frequent fluctuations in output render SMURF's output size larger than ours using of leve1 compression.

Since our system supports containment inference, it can also apply level 2 compression that uses stable containment 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 when the read rate exceeds 0.65, resulting in a compression ratio as low as 2%. For read rates below 0.65, however, the loss of containment accuracy results in disruptive fluctuations of events in output, hence increasing the overall the output volume.

Fig. 11(c) further includes containment information in the output, where the solid lines shows the compression ratios using level 1 and level 2 compression, and the dashed lines offer each method's location only results as a reference. We observe the same tradeoff between level 1 and level 2 compression as in Fig. 11(b) when the read rate varies; the cross point is still around 0.65. It is also interesting to see that when the read rate exceeds 0.8, the containment output represents only a small fraction of the total output volume, allowing this additional information to be included in a compressed output that is only 2.5-4% of the initial input size for level 2 compression and 5-6% for level 1 compression.

In summary, our system outperforms SMURF for object location updates in both event error rate and compression ratio, while containment updates are unique to our system, allowing both level 1 and level 2 compression. In addition, there are tradeoffs in compression ratio between level 1 and level 2 compression. Given the expected read rate in an RFID deployment, it would be possible to select level 2 for readers with high accuracy and level 1 for those with lower accuracy, hence enabling a compressed output size ranging from 20% to 2.5% of the input size.

VII. Related Work

RFID stream processing. Several techniques have been proposed recently to clean noisy RFID data streams [12], [18], [19], [21]. The most relevant to our work is the HiFi system [12], [18] that performs per-tag smoothing using the SMURF algorithm [19] 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 [28]. 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 [14] or event pattern detection [24], 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 [2]. Siemens RFID middleware [30] uses application rules to archive RFID data streams into databases. The Cascadia system [31] 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 [25], to recover high-level information from incomplete, noisy data by exploiting known constraints and prior statistical knowledge [33], 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 [15]. These techniques, however, are not designed for fast low-level interpretation 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 has focused on traditional sensor data types [34], [23], [6], [7], [26], [27], such as temperature and light. The proposed techniques, such as data acquisition [23], [6], [5], approximation [4], and sampling [35], are geared towards queries natural to such data such as selection and aggregation. In contrast, RFID data captures object identification and its processing raises challenges related to locationing and correlation of objects and data volume reduction. Recent research on GPS readings supports use-defined views using model-based probabilistic inference [20]. However, GPS data differs from RFID data because it already reveals object locations and GPS applications do not concern object containment relationships or anomaly detection like ours.

VIII. Conclusions

In this paper, we presented a novel data interpretation and compression substrate over RFID streams to address the challenges of incomplete data, insufficient information and high volumes. Our substrate employs a time-varying graph model to capture inter-object relationships such as containment. It then employs a probabilistic inference algorithm to determine the most likely location and containment for each object and an efficient stream compression algorithm to remove redundant information from the output stream.

Our results show that our data interpretation techniques achieve error rates around or below 10% for location estimates for a wide range of RFID read rates and within 10% for containment estimates when the read rate reaches 80%. These techniques can also be performed efficiently on high-volume RFID streams. Furthermore, our compression techniques yield significant reduction in data volume; they can encode rich location and containment information using only 20% or less of the raw input data size. Finally, we compare our system with SMURF, 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 is shown to outperform SMURF in both the error rate and the resulting compression ratio.

For future work, we plan to extend our interpretation and compression substrate to handle a mix of mobile and static readers. We will also broaden our system to incorporate RFID data interpretation at the low level and query processing at the high level, and further to do so in distributed environments.

References

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

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

[3] R. Cocci, T. Tran, et al., "Approximate aggregation techniques for sensor databases." in *ICDE*, 2008, pp. 1445-1447.

[4] J. Considine, F. Li, G. Kollios, et al., "Approximate aggregation techniques for sensor databases." in *ICDE*, 2004, pp. 449–460.

[5] A. Deshpande, C. Guestrin, et al., "Exploiting correlated attributes in acquisitional query processing." in *ICDE*, 2005, pp. 143–154.

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

[7] A. Deshpande and S. Madden, "MauveDB: supporting model-based user views in database systems." in *SIGMOD*, 2006, pp. 73–84.

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

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

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

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

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

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

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

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

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

[17] A. Hinze, "Efficient filtering of composite events," in *BNCOD*, 2003, pp. 207–225.

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

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

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

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

[22] C.-H. Lee and C.-W. Chung, "Efficient storage scheme and query processing for supply chain management using RFID," in *SIGMOD*, 2008, pp. 291–302.

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

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

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

[26] A. Silberstein, R. Braynard, and J. Yang, "Constraint chaining: on energy-efficient continuous monitoring in sensor networks." in *SIGMOD*, 2006, pp. 157–168.

[27] A. Silberstein, K. Munagala, and J. Yang, "Energy-efficient monitoring of extreme values in sensor networks." in *SIGMOD*, 2006, pp. 169–180.

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

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

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

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

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

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

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

[35] Y. Zhuang, L. Chen, X. S. Wang, et al., "A weighted moving average-based approach for cleaning sensor data." in *ICDCS*, 2007.