Adaptive Cleaning for RFID Data Streams

Shawn Ryan Jeffery
Minos Garofalakis
Michael Franklin

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-29
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-29.html

March 27, 2006

Adaptive Cleaning for RFID Data Streams

Shawn R. Jeffery
UC Berkeley

Minos Garofalakis
Intel Research Berkeley

Michael J. Franklin
UC Berkeley

ABSTRACT

A major impediment to the widespread adoption of RFID technology is the unreliability of the data streams produced by RFID readers; a 30% drop rate is not uncommon for RFID deployments. To compensate, most RFID middleware systems provide a "smoothing filter", a sliding-window aggregate that interpolates for lost readings. Typically, these middleware systems require the application to fix the size of the smoothing window in order to produce clean RFID data. Window-size selection, however, is a non-trivial problem: the window must be large enough to smooth lost readings but small enough to accurately capture tag movement. Furthermore, the ideal size may change over the course of the RFID deployment.

In this paper, we propose SMURF, the first declarative, adaptive smoothing filter for RFID data cleaning. SMURF models the unreliability of RFID readings by viewing RFID streams as a statistical sample of tags in the physical world, and exploits techniques grounded in sampling theory to drive its cleaning processes. Through the use of tools such as binomial sampling and π-estimators, SMURF continuously adapts the smoothing window size in a principled manner to provide accurate RFID data to applications.

1. INTRODUCTION

RFID (Radio Frequency IDentification) technology promises revolutions in areas such as supply chain management and ubiquitous computing enabled by pervasive, low-cost sensing and identification [17]. One of the primary factors limiting the widespread adoption of RFID technology is the *unreliability* of the data streams produced by RFID readers [8, 22]. The observed read rate (i.e., percentage of tags in a reader's vicinity that are actually reported) in real-world RFID deployments is often in the $60 - 70\%$ range [8, 20, 22]; in other words, over 30% of the tag readings are routinely dropped. Even higher drop rates are possible depending on environmental characteristics (e.g., in the presence of metal [18]).

Unfortunately, such error rates render raw RFID streams essentially useless for the purposes of higher-level applications (such as accurate inventory tracking). Instead, *RFID middleware systems* are typically deployed between the readers and the application(s) in order to correct for dropped readings and provide "clean" RFID readings to application logic. The basic data-cleaning mechanism in most such systems is a *temporal "smoothing filter"*: a sliding window over the reader's data stream that interpolates for lost readings from each tag within the time window [19, 23]. The goal, of course, is to reduce or eliminate dropped readings by giving each tag more opportunities to be read within the smoothing window. While the APIs for RFID middleware systems vary, smoothing filter functionality can be expressed as a simplified stream query (e.g., in CQL [6]) as shown in Query 1 (for a 5 second window).

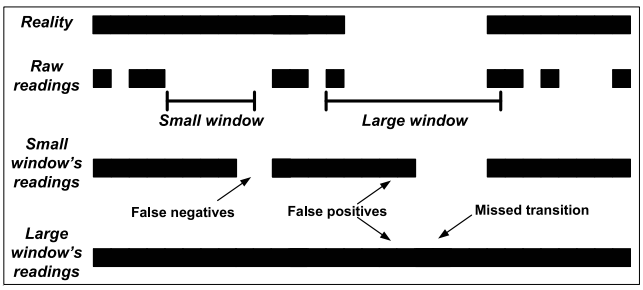


Figure 1: Tension in setting the smoothing-window size for tracking a single tag (dark bars indicate the tag is present/read): small windows fail to fill in dropped readings (false negatives); large windows fail to capture tag movement (false positives).

Query 1 *CQL Smoothing Filter to Correct for Dropped Readings.*

```
SELECT    distinct tag_id
FROM      rfid_readings_stream  [Range '5 sec']
GROUP BY  tag_id
```

Typically, the RFID middleware system requires the application to fix the smoothing window size (as in the above CQL statement). Setting the window size, however, is a non-trivial task: the ideal smoothing-window size needs to carefully balance two opposing application requirements (as shown in Figure 1): *ensuring completeness* for the set of tag readings (due to reader unreliability) and *capturing tag dynamics* (due to tag movements in and out of the reader's detection field).

– *Completeness:* To ensure that all tags in the reader's detection range are read, the smoothing window must be large enough to correct for reader unreliability. Small window sizes cause readings for some tags to be lost, leading to *false negatives* (i.e., tags mistakenly assumed to have exited the reader's detection range) and, consequently, a large underestimation bias (e.g., always under-counting the tag population). Adjusting the window size for completeness depends on the reader's read rate, which, in turn, depends on both the type of reader and tag as well as physical surround-

ings [13, 18].

– *Tag Dynamics:* Using a large smoothing window, on the other hand, risks not accurately detecting tag movements within the window, leading to *false positives* (i.e., tags mistakenly assumed to be present after they have exited the reader's detection range). In the worst case, the window may smooth over one or more tags leaving and returning, thus completely missing the variation in the underlying "signal" (Figure 1). Adjusting the window size for tag dynamics depends on the movement characteristics of the tags, which, in turn, can vary significantly depending on the application; for instance, a tag sitting on a shelf exhibits a very different movement pattern from a tag on a conveyor belt.

Any RFID deployer must seriously consider and study the factors governing the window size as discussed above when designing a cleaning scheme for raw RFID streams; in fact, ascertaining environment characteristics and configuring the hardware and middleware to account for these factors represents a large portion of the monetary and time cost of such deployments [31]. Furthermore, no single window size is expected to be effective over the lifetime of a deployment; thus, either the window size must be repeatedly reconfigured, or the quality of the data suffers.

The fundamental issue with any static windowing approach is that the window size is a non-declarative, low-level parameter that should not be exposed to the application level. Conceptually, what the application expects from the RFID middleware is a stream of readings that represent *an accurate picture of reality*; in other words, the application is only interested in accurately capturing a *true underlying "signal"* (such as individual tag readings or tag population counts) over time. Requiring the application to fix a smoothing-window size, however, essentially forces the application to decide beforehand exactly *how* to produce this "accurate" data stream.

Our Contributions. In this paper, we introduce SMURF (*Statistical sMoothing for Unreliable RFid data*), the first declarative, adaptive smoothing filter for cleaning raw RFID data streams. Unlike conventional techniques, SMURF does not expose the smoothing window parameter to the application; instead, it determines the "right" window size automatically and continuously adapts it over the lifetime of the system based on observed readings.

The main challenge for an adaptive smoothing scheme is to distinguish between periods of dropped readings and periods when a tag has moved. To address this problem, SMURF uses a sampling-based approach. One of the key ideas behind SMURF's adaptive algorithms is that RFID data streams can be modeled as a *random sample* of the tags in a reader's detection range. Through this sample-based view of observed RFID readings, SMURF employs algorithms grounded in statistical sampling theory to drive its adaptive smoothing techniques. More concretely, our contributions can be summarized as follows.

• **A Sampling-based View of RFID Data Streams.** SMURF exploits a novel view of RFID unreliability by modeling observed RFID readings as an *unequal-probability random sample* of tags in the physical world. This allows SMURF to balance the tension between reader unreliability and tag dynamics in a principled, statistical manner, by continuously adapting the smoothing strategy to provide accurate, unbiased data to applications. (Section 3)

• **An Adaptive Smoothing Filter for RFID Data.** Building on SMURF's sampling-based foundation, we propose two novel, adaptive smoothing mechanisms for (a) cleaning the readings of single tag using techniques based on *binomial sampling* [12] (*per-tag cleaning*), and (b) cleaning an aggregate signal (e.g., count) over a tag population based on π- (or *Horvitz-Thompson*) *estimators* [28] (*multi-tag cleaning*). (Section 4)

• **An Experimental Study Validating the Effectiveness of SMURF's Cleaning Algorithms.** We present a detailed experimental study using various schemes to clean both synthetic and real RFID data streams. First, these tests show that there is no single static window size that works well in all environments (reader and tag behavior), motivating the need for an adaptive approach. Second, we demonstrate SMURF's ability to adapt its data-cleaning strategy to a wide range of reader characteristics and tag behaviors; in an environment with changing conditions, SMURF reduces overall error by a factor of more than 3 compared to the best environment-specific static window. (Section 5)

SMURF is designed to be a component in a pipeline of operators responsible for low-level RFID data processing tasks such as cleaning, filtering, and spatial processing (see proposals such as ALE [5] and ESP [20, 21]). SMURF would be responsible for smoothing RFID readings from each reader before the streams are sent to other modules for additional processing. We believe that SMURF's sampling-based foundation offers a powerful conceptual framework for effective RFID data-cleaning tools. The set of techniques proposed in this paper can be directly incorporated in RFID middleware platforms to yield systems that (1) are substantially easier to configure and maintain; and, (2) produce more reliable RFID data, regardless of the deployment environment.

In the next section, we provide a general background on RFID technology and detail RFID reader unreliability.

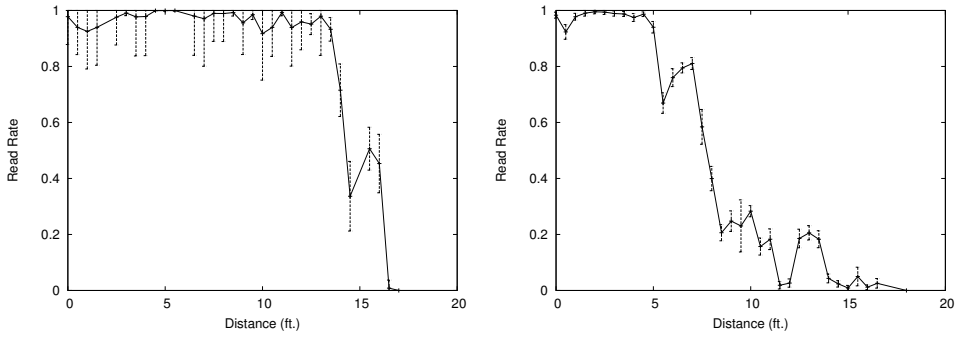
2. RFID BACKGROUND

RFID Technology Primer. RFID is an electronic tagging and tracking technology designed to provide non-line-of-sight identification. For the purposes of this paper, a typical RFID installation consists of three components: readers, antennae, and tags.

A *reader* uses *antennae* to communicate with *tags* using RF signals to produce lists of IDs in its detection field. Tags may either be active (battery-powered) or passive (no on-board battery). We focus on passive tags, as they are the most widespread variety of RFID tags. Tags store a unique identifier code (e.g., a 64 or 96-bit ID for EPCGlobal tags [16]). Although there exists RFID technology for multiple frequencies, we focus on 915 MHz technology, which has a long detection range (roughly 10-20 feet) and is typical of supply chain management applications.

Readers *interrogate* nearby tags by sending out an RF signal. Tags in the area respond to these signals with their unique identifier code. An *interrogation cycle* is one iteration through the reader's protocol that attempts to determine all tags in the reader's vicinity.

The results of multiple reader interrogation cycles are typ-



(a) Alien reader with Alien Squiggle tag under controlled conditions.

(b) Sensormatic reader with Alien I2 tag under noisy conditions.

Figure 2: RFID reader profiles for a single tag under different conditions. Error bars represent ± one standard deviation.

ically grouped into what we term *epochs*.[1] An epoch may be specified as a number of interrogation cycles or as a unit of time. A typical epoch range is 0.2-0.25 seconds [1, 30]. For each epoch, the reader keeps track of all the tags it has identified, as well as additional information such as the number of interrogation responses for each tag and the time at which the tag was laet read. Readers store this information internally in a *tag list* (Table 1) which is periodically transferred to readers' clients (either synchronously or asynchronously). For more information on RFID technology, see [33].

Tag ID	Responses	Timestamp
8576 2387 2345 8678	9	11:07:05
8576 4577 3467 2357	1	11:07:05
8576 3246 3267 5685	7	11:07:06

Table 1: Example reader tag list.

RFID Reader and Tag Performance. To better understand the unreliability of RFID readings, we profile two RFID readers with different tags in two environments. Our profiling methodology is as follows. We suspend a single tag at varying distances in the same plane as the antenna. For every 6-inch increment of distance from the reader, we measure the read rate (number of responses to interrogations) for 100 epochs.

Our profiling experiments use two types of readers, the Alien ALR-9780 [3] and the Sensormatic Agile 2 [29], with three types of tags (Alien "I2", "M", and "Squiggle" [4]). We test various combinations of these readers and tags in two environments. Our first environment, a large, wide-open room with little metal present, represents a controlled environment for RFID technology: we eliminate many of the causes of degraded read rates [18]. Our second profiling environment, a lab with metal objects such as desks and computer equipment, represents a noisy environment.

Figure 2 depicts the results from two different profiling experiments that are representative of the 8 different profiles we collected. (The plots show the read rate of the tag at distances ranging from 0 to 20 feet.) All of the profiles have similar properties despite being generated using different readers, tags, and environments. First, the overall detection range of all readers and tags profiled remains relatively constant at 15-20 feet. Second, within each reader's detection range, there are two distinct regions: (1) The area directly in front of the reader, termed the reader's *major detection region* [25], giving high detection probabilities (read rates at or above 95%); and, (2) The reader's *minor detection region*, extending from the end of the major detection region to the edge of the reader's full detection range, where the read rate drops off linearly (with some variation) to zero at the end of the detection range.

The main difference between our observed profiles lies in the percentage of the reader's detection range corresponding to its major detection region. For instance, the major detection region corresponds to roughly 75% of the full detection range for the profile in Figure 2(a), whereas it makes up only 25% of the range in the profile in Figure 2(b). Note that our profiles are consistent with the results of in-depth commercial studies of the performance of many different tags and readers under highly-controlled conditions [14].

We also profile the readers to determine how they respond to the presence of *multiple tags* in their detection ranges. For these tests (not shown here), we suspend 10 tags in the same plane as the reader and measure the average read rate for 100 epochs at varying distances from the reader. While the overall properties of the observed profile does not change (we still find a clear separation between a major and minor detection region), the overall read rate in the major detection region typically drops significantly to around 80%. Additional tests show that the read rate in the major detection region stays somewhat constant, at least up to 25 tags in the reader's detection range.

We use these observations in the design of some of SMURF's cleaning mechanisms and in the implementation of a realistic RFID data generator for evaluating our techniques.

3. RFID DATA STREAMS: A STATISTICAL SAMPLING PERSPECTIVE

Given the inherent unreliability of RFID readings, one of our key observations is that observed RFID data streams typically do *not* provide a complete, authoritative picture of the true population of tags in the physical world. Especially for tags outside a reader's major detection region, several readings may be missed, causing some tags to become "invisible" during a time window. These errors, of course, imply that typically only a *subset* of the tag population is actually observed. On the other hand, a lack of readings from a tag may not be due to missed readings but rather because the tag moved out of the detection field. The inherent tension between completeness of readings and captur-

[1]In ALE terms, an epoch is a *read cycle* [5].

ing tag dynamics (i.e., signal transitions) only exacerbates the problem: signals with a high degree of variability (e.g., counting highly-mobile tags) require short smoothing windows in order to capture rapid changes in the measurement data; but, obviously, a smaller window leads to more missed readings and more severe and systematic underestimation. The conventional solution of increasing the window size to guarantee completeness simply does not work here, as it can cause signal variations to be lost ("smoothed out") due to aggregation.

Rather than striving for completeness, our proposed adaptive smoothing filter, SMURF, captures tag dynamics while compensating for lost RFID readings in a principled, statistical manner. The key idea is that the observed RFID readings can be viewed as a *random sample* of the population of tags in the physical world. In the remainder of this section, we briefly explain the details of this process and the challenges in designing SMURF.

Mapping RFID Readings to a Sampling Process: SMURF Methodology and Challenges. Consider an epoch t. Recall from Section 2 that an epoch is the atomic unit of detection and is considerably smaller than the expected window size; that is, epochs represent our basic "time units", many of which make up a smoothing window [19, 23]. Without loss of generality, let N_t denote the (unknown) size of the underlying tag population at epoch t, and let $S_t \subseteq \{1, \ldots, N_t\}$ denote the subset of tags observed ("sampled") by the reader during that epoch. SMURF essentially views S_t as an *unequal probability random sample* of the tag population. Specifically, for each tag $i \in S_t$, SMURF employs the response-count information for tag i stored in the reader tag list (Table 1) in conjunction with the known number of interrogation cycles per epoch to derive a *per-epoch sampling probability* $p_{i,t}$. This sampling probability $p_{i,t}$ is empirically estimated as the observed read rate for tag i during that epoch; for instance, assuming a reader configuration with a total number of 10 interrogation cycles per epoch, the sampling probabilities for the first and second tags in Table 1 would be $p_{x78,t} = 0.9$ and $p_{x57,t} = 0.1$, respectively. Of course, these sampling probabilities differ across tags and can also vary over time as the observed tags move within reader's detection range.

Our key insight of viewing each RFID epoch as a "sampling trial" enables SMURF's novel, statistics-driven perspective on adaptive RFID data cleaning. In a nutshell, SMURF views the observed readings over a smoothing window (i.e., a sequence of consecutive epochs) as the result of repeated random-sampling trials, and employs techniques and estimators grounded in statistical sampling theory to reason about the underlying physical-world phenomena and drive its adaptive RFID data cleaning algorithms. More specifically, SMURF uses the statistical properties of the observed random sample to appropriately adapt the size of its smoothing window based on (1) completeness requirements and, (2) signal transitions detected as *"statistically-significant" changes* in the underlying tag readings. Further, even for window sizes that are necessarily small (to capture fast-varying signals), SMURF uses *sampling-based estimators* [12, 28] to provide accurate, *unbiased* estimates for tag-population aggregates (e.g., counts), and avoid the systematic under-counting of conventional smoothing techniques. Thus, SMURF's sampling-based foundation enables it to explore the tension between completeness and tag dynamics in a principled, statistical manner that continuously adapts the smoothing strategy based on statistical properties of the data to provide accurate, unbiased data to applications. Experimental results (Section 5) confirm that SMURF's sampling-based model enables it to effectively clean RFID data streams.

4. SMURF RFID DATA CLEANING

In this section, we present SMURF, our declarative, self-tuning smoothing filter. We begin by outlining SMURF's high-level components and then detail its cleaning mechanisms. SMURF's internal architecture comprises two primary cleaning mechanisms aimed at (1) producing accurate data streams for individual tag-ID readings (*per-tag cleaning*); and, (2) providing accurate aggregate (e.g., count) estimates over large tag populations (*multi-tag cleaning*). Additionally, SMURF incorporates two modules that apply to both data-cleaning techniques: a sliding-window processor for fine-grained RFID data smoothing, and an optimization mechanism for improving cleaning effectiveness by detecting *mobile tags*. We first briefly discuss how SMURF processes readings within its adaptive window, then detail the two key cleaning mechanisms used by SMURF, and finally present SMURF's mobile tag detection enhancement.

4.1 SMURF Sliding Window Processing

Window-based smoothing in SMURF closely resembles traditional sliding-window aggregate processing [2, 7, 10] as expressed, for example, in Query 1 (of course, with the fixed-size `Range` clause removed). Similar to other RFID smoothing filters, SMURF produces a tag reading for a window if there exists at least one reading for the tag within that window [19, 23]. In addition, to enable our more sophisticated data-cleaning schemes, SMURF's sliding-window processor implements two basic modifications to conventional RFID filters: (1) partitioned RFID stream smoothing, and (2) epoch-based mid-window slide.

As subsets of tagged objects may behave very differently (e.g., in a warehouse environment, some tagged items may be placed on a shelf while others are moved on forklifts), SMURF's cleaning techniques must be able to adapt the smoothing-window sizes at a *much finer granularity* compared to traditional RFID middleware systems that fix a single window size for the entire tag population. At one extreme, when tracking individual tag movements, SMURF runs its adaptive sliding-window processing *per tag ID*. In general, the granularity of SMURF's windowing mechanisms is determined by the *aggregate query of interest*. That is, by a pair `(subset, aggregate)` determining the `subset` of tags over which the `aggregate` value (e.g., count) is monitored. Note that such fine-grained processing can be expressed in a declarative fashion (e.g., through the `Partition By` clause in CQL [6]).

As epochs are a sample cycle in SMURF's sample-based model of RFID data, SMURF slides its windows by a single epoch (as opposed to a time period or by tuples). Furthermore, we set SMURF's slide point to the middle of the window. That is, SMURF produces readings with an epoch value corresponding to the midpoint of the window (after the entire window has been seen). A mid-window slide point captures the intuitive notion of smoothing: e.g., if there are reported readings at times $t-1$ and $t+1$, then there is likely a reading at time t. We experimentally validated that

a mid-window slide point yields the most reliable readings.

4.2 Adaptive Per-Tag Cleaning

To clean readings from a single tag, the fundamental challenge is to distinguish between periods of dropped readings and periods where the tag has actually left the reader's detection field. SMURF must set window size such that it provides completeness (for periods of dropped readings) and accurately captures transitions (for periods where the tag has left). To help differentiate between these two behaviors and to guide subsequent window adaptations, SMURF employs statistical mechanisms based on its random-sample view of RFID data.

A Binomial Sampling Model for Single Tag Readings. Consider the simple case of cleaning the readings from a single tag (say, i) based on a reader's observations over a smoothing window of size w_i epochs (say, $W_i = (t - w_i, t]$). Assume, for the time being, that tag i is present in the reader's range throughout the window W_i, and has the same probability, p_i, of being observed in each epoch of W_i. SMURF views each epoch as an independent *Bernoulli trial* (i.e., a sampling draw for tag i) with success probability p_i. This, in turn, implies that the number of successful observations of tag i in the window is a random variable that follows a *binomial distribution* with parameters (w_i, p_i) (i.e., $B(w_i, p_i)$). In the general case, assume that tag i is seen in only a subset $S_i \subseteq W_i$ of all the epochs in W_i, and let p_i^{avg} denote the average empirical read rate over these observation epochs; that is, $p_i^{avg} = \sum_{t \in S_i} p_{i,t}/|S_i|$, where each $p_{i,t}$ is calculated based on the reader's tag list information as demonstrated in Section 3. Note that we assume that within an appropriately-sized window, the p_is will be relatively homogeneous and thus averaging is a valid estimate of the actual p_i.[2] Based on our discussion above, and under the assumption that the tag stays within the reader's detection field throughout W_i, we can view S_i as a *binomial sample* (of epochs in W_i) and $|S_i|$ as a $B(w_i, p_i^{avg})$ binomial random variable; thus, from standard probability theory, we can express the *expectation* and *variance* of $|S_i|$ as:

$$\mathrm{E}[|S_i|] = w_i \cdot p_i^{avg} \qquad \text{and} \qquad \mathrm{Var}[|S_i|] = w_i \cdot p_i^{avg} \cdot (1 - p_i^{avg})$$

Next, we discuss how SMURF employs this binomial sampling model to accurately detect transitions (e.g., departures of tag i) and dynamically adjust its smoothing window for per-tag cleaning.

Per-Tag Adaptive Window Size Adjustment. With our binomial sampling model in place, we first consider the problem of setting SMURF's window size w_i to guarantee *completeness*. In other words, we want to ensure that there are enough epochs in W_i such that tag i is observed (if it exists within the reader's range). Given the statistical nature of our model, our guarantees are necessarily probabilistic; that is, we can set w_i to ensure that tag i is read with high probability, as described in the following lemma.

LEMMA 4.1. *Let p_i^{avg} denote the observation probability for tag i during an epoch. Then, setting the number of epochs within the smoothing window to be $w_i \geq \lceil \frac{\ln(1/\delta)}{p_i^{avg}} \rceil$ ensures that tag i is observed within W_i with probability $> 1 - \delta$.* ∎

[2]In cases where this homogeneity assumption does not hold due to a tag moving rapidly away from the reader, our mobile tag detection algorithm (Section 4.4) allows SMURF to appropriately size its window to capture tag dynamics.

Proof: Based on our model of independent Bernoulli trials for observing tag i, the probability that we miss a reading from tag i over w_i sampling trials is exactly $(1 - p_i^{avg})^{w_i}$. Setting this probability $\leq \delta$ and taking log's gives $w_i \ln(1 - p_i^{avg}) \leq \ln \delta$. Combining this with the inequality $-x \geq \ln(1 - x)$ for $x \in (0, 1)$, we see that it suffices to require that $-w_i p_i^{avg} \leq \ln \delta$, or, equivalently, $w_i \geq \frac{\ln(1/\delta)}{p_i^{avg}}$. This completes the proof. ∎

Thus, a window size of $w_i = \lceil \frac{\ln(1/\delta)}{p_i^{avg}} \rceil$ is sufficient to guarantee completeness (with high probability). In general, due to the weak (logarithmic) dependence on δ, small (i.e., less than 0.1) settings for δ do not have a large effect on the overall window size. On the other hand, the δ parameter does provide a "knob" that allows an application to express its preferences with respect to balancing false positives versus false negatives. As demonstrated in our experimental numbers (Section 5), adjusting δ has little effect on the overall effectiveness of SMURF cleaning.

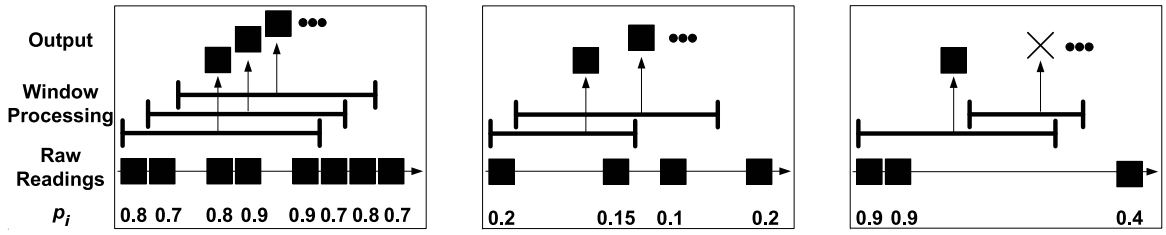
While using a smoothing-window size as suggested by Lemma 4.1 guarantees completeness (i.e., correct detection of tag i) with high probability, it can also lead to missing the temporal variation in the underlying signal (e.g., due to the movements of tag i). Note that, in the per-tag case, we are dealing with a *binary signal*: either tag i is there (value = 1) or it is not (value = 0). As discussed earlier, large smoothing windows can *miss signal transitions*, where tag i is mistakenly presumed to be present in the reader's detection range due to the interpolation of readings inside the window (Figure 1). In order to avoid smoothing over transitions and producing many false positives, SMURF needs to accurately determine when tag i exited the reader's detection range (as opposed to a period of dropped readings) and decrease the size of its window. We term this process *transition detection*.

Given the unreliability of tag readings, accurate transition detection becomes crucial: readings *will* routinely be lost (e.g., for tags outside the reader's major detection region (Figure 2)), and an overly-sensitive transition detection mechanism can result in losing the smoothing effect and emitting (useless) raw tag readings. On the other hand, a coarse detection mechanism can miss true signal transitions, resulting, once again, in false positives. SMURF employs its binomial sampling model to detect transitions in a principled manner, as *statistically-significant deviations* in the observed binomial sample size from its expected value. More formally, assuming that the current window size w_i and sampling probability p_i^{avg} are not too small, it follows from a Central Limit Theorem (CLT) argument that, assuming no transition occurred in the current window, the value of $|S_i|$ is within $\pm 2\sqrt{\mathrm{Var}[|S_i|]}$ of its expectation with probability close to 0.98. Based on this observation, SMURF flags a transition (i.e., exit) for tag i in the current window if the number of observed readings is less than the expected number of readings *and* the following condition holds: [3]

$$||S_i| - w_i p_i^{avg}| \; > \; 2 \cdot \sqrt{w_i p_i^{avg} (1 - p_i^{avg})} \qquad (1)$$

SMURF Per-Tag Cleaning Algorithm. A pseudo-code description of SMURF's adaptive per-tag cleaning

[3]More conservative, non-CLT-based probabilistic criteria, e.g., based on the Chebyshev or Chernoff bounds [24] can also be used here.



(a) Normal sliding window processing for tag i in SMURF. SMURF uses epoch-based midpoint sliding: at each epoch t, SMURF emits a reading with an epoch value corresponding to the midpoint of the window.

(b) Ensuring completeness. In the left-most window, p_i^{avg} demands a larger window such that the tag has a high probability $(1-\delta)$ of being detected. Thus, the window size is increased.

(c) Transition detection. In the left-most window, the number of readings indicates a statistically-significant deviation given the p_i^{avg}. Thus, a transition is likely to have occurred so the window is halved.

Figure 3: Graphical depiction of per-tag cleaning in SMURF.

algorithm is depicted in Algorithm 1. SMURF employs the common Additive-Increase/Multiplicative-Decrease (AIMD) paradigm [11] to adjust its window size for each tag i, based on guidance from its binomial-sampling model as discussed above.

Algorithm 1 SMURF Adaptive Per-Tag Cleaning

Require: T = *set of all observed tag IDs*
δ = *required completeness confidence*
$\forall i \in T$, $w_i \leftarrow 1$
while (`getNextEpoch`()) **do**
  for (i in T) **do**
    `processWindow`(W_i)
    $w_i^* \leftarrow$ `completeSize`(p_i^{avg}, δ) // *Lemma 4.1*
    if ($w_i^* > w_i$) **then**
      $w_i \leftarrow \max\{\min\{w_i + 2, w_i^*\}, 1\}$
    else if (`detectTransition`($|S_i|$, w_i, p_i^{avg})) **then**
      $w_i \leftarrow \max\{\min\{w_i/2, w_i^*\}, 1\}$
    end if
  end for
end while

SMURF runs a sliding-window aggregate for each observed tag i. The window size is initially set to one epoch for each tag, and then adjusted dynamically based on observed readings. (If at any point during processing SMURF sees an empty window for a tag, it resets its window size to one epoch.)

During each new epoch, and for each tag i, SMURF starts by processing the readings of tag i inside the window W_i (`processWindow`(W_i)). This processing includes estimating the required model parameters for tag i (e.g., p_i^{avg}, $|S_i|$) using tag-list information as well as emitting an output reading for tag i if there exists at least one reading within the window. Then, SMURF consults its binomial-sampling model to determine the number of epochs necessary to ensure completeness with high probability (`completeSize`(p_i^{avg}, δ)), based on Lemma 4.1. If the required size w_i^* exceeds the current window size $w_i = |W_i|$, SMURF grows its current window size for i additively.[4] This "additive window growth" rule allows SMURF to incrementally monitor the tag's readings as the window grows and thus remain responsive to changes in the underlying signal.

[4]In order to advance the slide point, which is set to the middle of the window, by one epoch, the window must be grown by 2 epochs.

If the current window size satisfies the completeness requirement, then SMURF tries to detect if a transition occurred during W_i (`detectTransition`($|S_i|$, w_i, p_i^{avg})), based on Condition (1). If a transition is flagged, SMURF multiplicatively decreases the size of its current smoothing window for i (i.e., divides it in half). By multiplicatively decreasing its window size, SMURF can quickly react to detected transitions and, at the same time, avoid over-reaction in the unlikely event of an incorrect transition detection. Of course, if the completeness requirement is met and no transition is detected, SMURF continues with its current window size for tag i.

To summarize, Figure 3 graphically depicts some example scenarios under SMURF's basic per-tag cleaning scheme.

4.3 Adaptive Multi-Tag Aggregate Cleaning

In many real-world RFID scenarios, applications need to track large populations of tags, typically in the several hundreds or thousands. In addition, applications often do not require information for each individual tag, and only need to track simple *aggregates* (e.g., counts or averages) over the entire tag population. For instance, a retail-store monitoring application may only need to know when the *count* of items on a shelf drops below a certain threshold.

An "obvious" cleaning approach in such scenarios is to apply SMURF's per-tag cleaning algorithms (Section 4.2) for each individual tag in the population and then aggregate the results across individual smoothing filters for each epoch. Such a solution, however, potentially suffers from underestimation bias: tags not read *at all* in a window will not be counted. Additionally, this approach incurs significant overhead: SMURF needs to continuously track and dynamically adapt the window for each individual tag; furthermore, many window adjustments can happen (e.g., with mobile tags) even though the underlying aggregate signal (e.g., population count) remains stable. To avoid such overheads, SMURF employs statistical-estimation techniques to accurately estimate the population count without cleaning on a per-tag basis.

Random-Sampling Model and Estimators for Multi-Tag Aggregates. Consider the problem of estimating the *count* of the tag population over a window of size w epochs (say, $W = (t - w, t]$). As earlier, we use p_i^{avg} to denote the average empirical sampling probability for tag i during W (i.e., the average read rate over all observations of i in W derived from the reader's tag list information). SMURF

views each epoch as an independent "sampling experiment" (i.e., Bernoulli trial) with success probability p_i^{avg}; thus, the overall probability of reading tag i *at least once* during W is estimated as:

$$\pi_i \;=\; 1-(1-p_i^{avg})^w \tag{2}$$

Again, the size w of the smoothing window plays a critical role in capturing the underlying aggregate signal: a large w ensures completeness (i.e., all π_i's are close to 1), but a small w is often needed to ensure that the variability in the population count is adequately captured. Unfortunately, compromising on completeness implies that RFID smoothing algorithms that simply report the observed readings count can result in consistent underestimation errors.

SMURF employs its unequal-probability random sampling model to correct for this underestimation bias through the use of π- (or, *Horvitz-Thompson*) *estimators* [28] to approximate population aggregates.[5] Specifically, let $S_W \subseteq \{1,\ldots,N_W\}$ denote the subset of observed (i.e., sampled) RFID tags over the window W (N_W denotes the true count), with sampling probabilities determined by Equation (2). The π-estimator for the population count based on the sample S_W is defined as:

$$\hat{N_W} \;=\; \sum_{i\in S_W}\frac{1}{\pi_i} \tag{3}$$

In other words, the π count estimator weights each sample point i with its sampling probability π_i. The reason for this is fairly intuitive: If tag i, which is observed with probability π_i, appears once in the sample, then, on average, we expect to have $1/\pi_i$ tags with similar probabilities in the full population (since $\pi_i \cdot \frac{1}{\pi_i} = 1$); thus, the single occurrence of i in the sample is essentially a "representative" of $1/\pi_i$ tags in the full population.

The $\hat{N_W}$ π-estimator is *unbiased* (correct on expectation); that is, $E[\hat{N_W}] = N_W$ [28]. Thus, by weighting with sampling probabilities, SMURF's π-estimator techniques correct for the underestimation bias of conventional smoothing schemes in a principled, statistical manner (even for small smoothing window sizes). Similar calculations show that, assuming independence across different tags, the variance of $\hat{N_W}$ is estimated by [28]:

$$\hat{\mathrm{Var}}[\hat{N_W}] \;=\; \sum_{i\in S_W}\frac{1-\pi_i}{\pi_i^2} \tag{4}$$

Of course, even though SMURF guarantees unbiasedness, as the window shrinks, the observed sample size and corresponding π_i's also drop, resulting in possibly lower-quality (high-variance) π-estimators. As our experimental results demonstrate, SMURF's π-estimation algorithms still significantly outperform conventional smoothing algorithms in such "difficult" settings.

Adaptive Window Size Adjustment for Multi-Tag Aggregates. As in the single-tag case, we first consider the problem of upper-bounding SMURF's smoothing window in a manner that results in reasonably complete readings over the reader's detection range. Let S_W denote the sample of (distinct) tags read over the current smoothing window W, and let $p^{avg} = \sum_{i\in S_W} p_i^{avg}/|S_W|$ denote the average per-epoch sampling probability over all observed tags. Following a rationale similar to that in Lemma 4.1, we set the upper bound for SMURF's smoothing window size for multi-tag aggregate cleaning at $w = \lceil \frac{\ln(1/\delta)}{p^{avg}} \rceil$; in other words, for completeness, we require that the "average tag" in the underlying population is read with high probability ($\geq 1-\delta$). (A more pessimistic window-size estimate would use the *minimum* of the p_i^{avg}'s in the above calculation to ensure that the "worst" tag is read — however, since SMURF employs π-estimators to correct for missed readings, such a pessimistic window could result in overestimation errors.)

SMURF also employs its random-sampling model and π-estimator calculations in order to dynamically adapt its smoothing window size to accurately capture the temporal variation in the population count (analogous to transition detection in the per-tag case). The key observation here is that SMURF can detect transitions in the underlying aggregate signal as *statistically-significant changes* in its aggregate estimates over sub-ranges of its current smoothing window. Specifically, assume $W = (t-w, t]$ is the current window, and let $W' = (t-w/2, t]$ denote the second half of W. Also, let $\hat{N}_W$ and $\hat{N}_{W'}$ denote the π-estimators for the tag population counts during W and W', respectively. Under similar CLT-like assumptions as in Section 4.2, we have that the corresponding true population counts (N_W and $N_{W'}$) satisfy $N_W \in \hat{N}_W \pm 2\sqrt{\hat{\mathrm{Var}}[\hat{N}_W]}$ and $N_{W'} \in \hat{N}_{W'} \pm 2\sqrt{\hat{\mathrm{Var}}[\hat{N}_{W'}]}$ with high probability. Based on these observations, SMURF detects that a statistically-significant transition in population count has occurred in the second half of W if the following condition is satisfied:

$$|\hat{N}_W - \hat{N}_{W'}| \;>\; 2\left(\sqrt{\hat{\mathrm{Var}}[\hat{N}_W]} + \sqrt{\hat{\mathrm{Var}}[\hat{N}_{W'}]}\right) \tag{5}$$

The above condition essentially asserts that the difference $|N_W - N_{W'}|$ of true counts is non-zero with high probability.

There are two important points to note here. First, remember that the key problem with adaptive smoothing-window sizing is to correct for *false-positive readings* due to a large window W and a drop-off in the true number of tags in the detection range over W. (An increase in the tag count over W is always "caught", regardless of the current window size, since the observed new readings are by default interpolated throughout the smoothing window.) Condition (5) attempts to accurately capture such significant drop-offs within the current window, and allows SMURF to adaptively shrink its smoothing window size. Second, while Condition (5) with $W' = (t-w/2, t]$ is sufficient to identify count changes that persist for at least $w/2$ epochs within the smoothing window, it may still miss transitions that last for $< w/2$ epochs. A more general solution here is to check Condition (5) for a series of dyadic-size windows $W' = (t-w/2^i, t]$ ($i = 1, 2, \ldots$) at the tail end of W, and signal a transition whenever one of these conditions is satisfied. (Note that, as we slide W across time, any transition is initially located at the tail end of W and, thus, can be discovered by the above technique.) The caveat here, of course, is that, as the sub-range within W decreases, the variability of

[5] Although our discussion here focuses primarily on tag counts, SMURF's π-estimator schemes for adaptive multi-tag cleaning can be easily extended to other aggregates. For instance, if our goal is to estimate the sum of some measure (e.g., temperature) over the underlying tag population, then the contribution of tag i to our π-estimator formula becomes $\frac{y_i}{\pi_i}$, where y_i is the measured quantity of interest.

the $\hat{N}_{W'}$ estimate goes up, making it difficult to detect very short-lived transitions. Our empirical results demonstrate that using Condition (5) for just the second-half window $W' = (t - w/2, t]$ is sufficient to provide accurate, adaptive population-count estimates to applications.

SMURF Multi-Tag Cleaning Algorithm. Algorithm 2 depicts the pseudo-code for SMURF's multi-tag cleaning scheme, which incorporates the above techniques. Similar to per-tag cleaning, SMURF uses AIMD to adjust its smoothing window size; however, in contrast to the per-tag case, only a single window W is maintained (and adapted) for all observed tags.

Algorithm 2 SMURF Adaptive Multi-Tag Cleaning

```
Require: δ = desired average completeness confidence
  w ← 1
  while (getNextEpoch()) do
    processWindow(W)
    W ← slideWindow(w)
    w* ← completeSize(p^avg, δ)                    // Lemma 4.1
    if (detectTransition(N̂_W, N̂_W', V̂ar[N̂_W], V̂ar[N̂_W'])) then
      w_i ← max{min{w_i/2, w_i*}, 1}
    else if (w* > w) then
      w_i ← max{min{w_i + 2, w_i*}, 1}
    end if
  end while
```

For each epoch, SMURF starts by processing the readings in the window W (`processWindow`(W)). This involves computing key window parameters (e.g., p^{avg}, $\hat{N}_{W'}$, $\hat{\text{Var}}[\hat{N}_{W'}]$), determining the aggregate contribution from each tag ($1/\pi_i$), and calculating (and subsequently emitting) the estimated tag count ($\hat{N}_W$) using π-estimation.

The window is then checked for a statistically-significant change in the count estimate in its second half (`detectTransition` ($\hat{N}_W$, $\hat{N}_{W'}$, $\hat{\text{Var}}[\hat{N}_W]$, $\hat{\text{Var}}[\hat{N}_{W'}]$)) based on Condition (5). If a change is detected, SMURF halves its window size. Otherwise, SMURF checks if the current window meets the completeness requirement based on the average tag detection probability p^{avg} and grows its window additively, if necessary.

Note that the ordering of the increasing and decreasing phases in Algorithm 2 is reversed from the per-tag case. Since SMURF's π-estimation scales-up readings in a window to estimate the underlying tag population, the completeness requirement (i.e., a large window) is not as crucial for accurate estimation as in the single-tag case (where a missed reading causes a 100% error). Thus, multi-tag processing in SMURF focuses primarily on capturing transitions in the aggregate and uses π-estimation to compensate for small windows in an unbiased manner.

4.4 Mobile Tag Detection

Here we present an enhancement to SMURF processing that applies to both per-tag and multi-tag cleaning.

Tags that are detected far away from the reader with a low probability can force SMURF to use a large smoothing-window (based on Lemma 4.1). While large windows are necessary to accurately detect *static tags* placed far from the reader, they can cause problems in environments where tags are *mobile*. For per-tag cleaning, a mobile tag detected with a low $p_{i,t}$ just before it leaves the reader's detection range causes a large number of false positives since it forces an abnormally large window. In the multi-tag case, a similar reading results in an overly large contribution to the overall count estimate, and thus a large over-estimation error.

To alleviate the effects of low $p_{i,t}$s produced by mobile tags, we enhance SMURF with a pre-processing stage that recognizes mobile tags that are exiting the detection range and reacts accordingly. This stage, termed *mobile tag detection*, monitors individual tag $p_{i,t}$s, and attempts to determine when low detection probabilities are caused by an exiting mobile tag (as opposed to a static remote tag, which should force a large window). Mobile tag detection uses a simple heuristic: tags that are read with consistently falling $p_{i,t}$s are likely to be moving away from the reader and, thus, may be exiting the detection range soon. Such readings with low $p_{i,t}$ values are filtered away by SMURF's mobile tag detector.

SMURF's mobile tag detection algorithm forms a best-fit line using least squares fitting with the observed $p_{i,t}$s in the window. Using the slope of this line (in units of $\frac{\Delta p_{i,t}}{epochs}$), SMURF calculates a filter threshold as $filterThresh = \epsilon - slope * w_{md}$. This threshold is a value of $p_{i,t}$ for which it is estimated that the $p_{i,t}$ for the tag will drop below some value ϵ in the next w_{md} epochs, where w_{md} is w_i in the per-tag case and w in the multi-tag case. The reason the algorithm looks ahead w_{md} epochs is intuitive: the larger the window the greater the potential for false positives if the tag exits; thus, SMURF more aggressively filters readings when the window size is large. Using $\epsilon = 0$ yields a good indication of whether the tag will be exiting the detection range soon. Mobile tag detection filters all readings for mobile tags whose $p_{i,t}$s fall below this threshold, thus preventing such readings from adversely influencing the window size calculation or count estimation.

5. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate SMURF's data cleaning techniques. For both per-tag and multi-tag cleaning, we illustrate two key points: (1) there is no single static window that works well in the face of fluctuating tag movement, reader unreliability, or both; and, (2) across a range of environments with different levels of tag movement and reader unreliability, SMURF cleaning techniques produce an accurate stream of readings (both individual tag IDs and counts) describing tags in the physical world.

Additionally, we illustrate SMURF's declarative nature in two ways. First, it successfully adapts its window size as the environment changes, freeing the application from specifying an imperative window size parameter. Second, it exposes a tuning knob, δ, that allows an application to trade-off false positives for false negatives without compromising overall cleaning accuracy.

Before showing experiments for both per-tag and multi-tag cleaning, we first describe our experimental setup.

5.1 Experimental Setup

In order to run experiments across a wide variety of scenarios, we built a data generator to produce synthetic RFID streams given realistic configurations of tags and readers.

Reader Detection Model. The data generator is based on RFID reader detection regions as observed in our tests described in Section 2. We simplify a reader's detection field

to derive a model of RFID readers as shown in Figure 4.

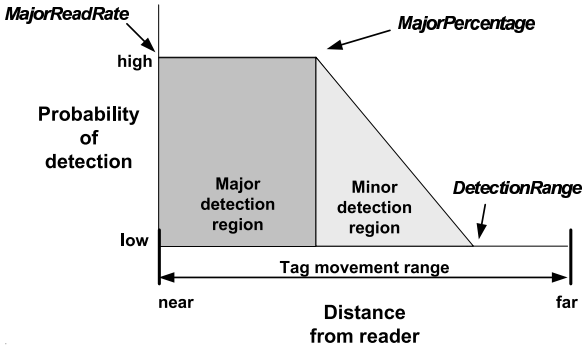


Figure 4: Reader model and tag behavior for the RFID data generator

The model uses the following parameters to capture a wide variety of reader behavior under different conditions:

- *DetectionRange*: the distance in feet from the reader to the edge of the reader's detection range.
- *MajorPercentage*: the percent of the reader's overall detection range that is the major detection region. The edge of the major detection region can be calculated by multiplying this number by the total detection range *DetectionRange*.
- *MajorReadRate*: the read rate (i.e., the probability of detection) of a tag within the major detection region. The read rate in the minor detection region drops off linearly to the end of the reader's detection range.

Tag Behavior. We randomly place $NumTags$ tags uniformly between 0 and 20 feet from the reader along its central axis. Here we have detailed data describing the read rate of the readers along this axis as described in Section 2. By moving the tags along this axis, we can generate readings with $p_{i,t}$s corresponding to many types of movement. For instance, the $p_{i,t}$s of readings generated by a tag passing through an RFID-enabled door can be produced by moving a tag from outside *DetectionRange* to directly in front of the reader, and then back to outside *DetectionRange*.

Tags move between 0 and 20 feet following one of two behaviors representative of a range of RFID applications:

1. *Pallet*: All tags have the same velocity. This simulates tags that are grouped together, such as tagged items on a pallet.
2. *Fido*: Each tag chooses a random initial velocity (uniform between 1 and 3 feet/epoch). Note that the average velocity, 2 feet/epoch, is roughly equivalent to conveyor-belt speed [27]. Every 100 epochs, on average, each tag switches from a moving state to a resting state (and vice versa). When a tag resumes movement, it chooses another random velocity between 1 and 3 feet/epoch. This behavior simulates tracking environments such as a digital home, where each tag displays independent random behavior.

Data Generation. We run the generator for $NumEpochs$ epochs.[6] At each epoch, the generator determines which tags are detected based on the read rate at each tag's location relative to the reader. It then produces a set of readings containing a tag ID, epoch number, and the tag's $p_{i,t}$ (the read rate at which the reader read the tag). Additionally, the generator produces the set of all tags within the reader's detection range at each epoch to serve as the reality against which we compare the output of each cleaning mechanism.

Table 2 summarizes the experimental parameters we use to produce our synthetic RFID data traces. We manipulate the other parameters as part of our experiments. The settings for the RFID detection model were chosen as they represent the average of the reader/tag combinations we profiled. Recall from Section 2 the average read rate drops to around 0.8 with multiple tags in the reader's detection field; we set $MajorReadRate$ to reflect this behavior.

Parameter	Value
$DetectionRange$	15 feet
$MajorReadRate$	.8
$MajorPercentage$	varied
$NumTags$	25 (per-tag), 100 (multi-tag)
$Velocity$	varied
$NumEpochs$	5000 epochs

Table 2: Experimental parameters

Smoothing Schemes. We clean the data produced by the generator using SMURF as well as various sized static smoothing window schemes. We denote each fixed-window scheme as *Static-x*, where x is the size of the window in epochs (1 epoch $\approx$ 0.2 seconds).

5.2 Per-Tag Cleaning

The first set of experiments examine cleaning techniques that report individual tag ID readings. We analyze the performance of different cleaning schemes as the environment changes in terms of tag movement and reader reliability.

Our evaluation metric for per-tag cleaning is average errors per epoch. An error is a reading that indicates a tag exists when it does not (a false positive), or a (lack of) reading where a tag exists, but is not reported (a false negative). The average errors per epoch is calculated as $\sum_{j=1}^{NumEpochs}(FalsePositives_j + FalseNegatives_j)/NumEpochs$. This metric captures both types of errors in one metric that allows us to easily compare the effectiveness of each scheme.

Experiment 1: Varied Reader Reliability. In the first tests, we determine how each technique reacts to different levels of reader unreliability. We move tags using *Fido* behavior and vary the major detection region percentage. At each value for $MajorPercentage$ between 0 and 1, we measure the average errors per epoch produced by each scheme (recall that a lower value for $MajorPercentage$ corresponds to a more unreliable environment). Figure 5 shows the results of this experiment.

As can be seen, when the major detection region percentage is at 0 (a very noisy environment), the large windows do comparatively well, producing around 4 errors per epoch (i.e., misreporting about 4 tags out of 25 per epoch, on average). We truncate the traces for *raw* and *Static-2* due to their poor performance. As $MajorPercentage$ increases, the accuracy of all schemes improves due to more reliable raw data. When the major detection region makes up the entire

[6]To eliminate any effects caused by the start or end of the trace, we run the generator for an additional 300 epochs and omit the first and last 150 epochs from our measurements.

detection field ($MajorPercentage = 1$), the small windows are competitive; *Static-2* misreports slightly more than 1 tag out of 25 per epoch, on average.

In this experiment, SMURF cleaning has the lowest errors per epoch across the entire range of environments. Its relative performance is particularly good in this case because of its partitioned smoothing: it adapts, on a per-tag basis, to each tag's independent random behavior. Static windowing schemes that use a single window for all tags cannot capture this variation.

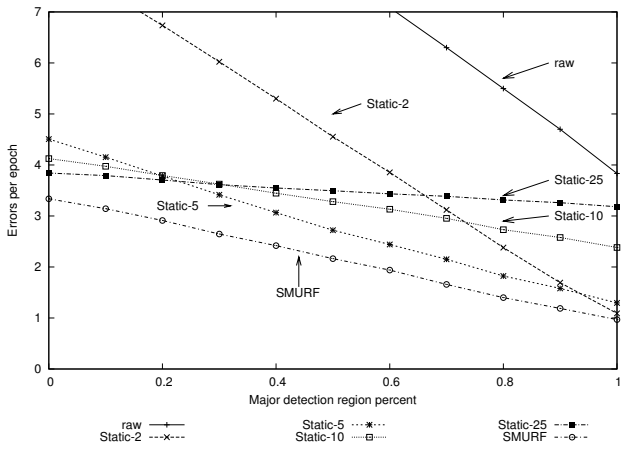


Figure 5: Average errors per epoch as $MajorPercentage$ varies from 0 to 1 with tags following *Fido* behavior.

To further investigate the mechanisms behind each smoothing scheme, we drill-down on a 200 epoch trace of this experiment. We focus on readings produced from a single tag ID in an very noisy environment: the major detection region percentage is set to 0 (the left-most x-value in Figure 5). The readings produced by the tag in this scenario are particularly challenging to clean as the data are highly unreliable and the tag sporadically moves at a high velocity: a smoothing scheme must be able to discern between periods of dropped readings and periods when the tag is transiently absent.

Figure 6 shows this time-line. The top subsection of the figure shows the tag's distance relative to the reader: the tag moves with a high velocity for a period, stops (at point A) for a period at the edge of the detection field, and then resumes movement. The middle subsection of the graph shows reality (e.g., the readings that would have been produced by a perfect reader), readings produced by the best two static window smoothing schemes (according the Figure 5), and the output of SMURF. The bottom subsection shows SMURF's window size over the course of the trace.

During the first period, the tag rapidly moves in and out of the detection field; the challenge for any smoothing scheme is to accurately capture this movement. Both static windows, however, fail to capture all of the tag's transitions. In the worst case, *Static-25* continuously reports the tag as present. Of course, smaller windows would catch these transitions, but would perform worse during the second phase of this trace.

At point A, the tag stops at the edge of the detection range, causing the reader to infrequently report the tag. *Static-10* fails to report the tag's behavior due to lack of readings: according to *Static-10*, the tag is still moving. *Static-25* accurately reports the tag's presence only because it reports the tag's existence continuously.

SMURF, in comparison, captures the high-level behavior of the tag during the entire trace. During the first phase of tag movement, it keeps its window size small, as can be seen at the bottom of the figure, and accurately reports that the tag is moving; it succeeds at catching all transitions. Once the tag stops, SMURF grows its window in reaction to the unreliable readings it receives during this period. Thus, SMURF accurately reports the tag as present despite the severe lack of readings.

Note that there is a short period just after point A where all schemes fail to report the tag while it exists. During this period, the reader produces no readings; no scheme without foreknowledge of the tag's motion can report the tag before it is read.

Experiment 2: Varied Tag Velocity. Next, we measure each scheme's effectiveness as the tag velocity changes. We fix the $MajorPercentage$ at 0.7 (representing a controlled environment) and move tags with *Pallet* behavior. At each velocity from 0 and 2 feet/epoch, we measure the average errors per epoch produced by each scheme. Figure 7 shows the results of this experiment.

The results illustrate the challenge in setting a static smoothing window. As we increase the tag velocity, there is no single static window that does consistently well. *Static-25* and *Static-10* do well when the tags are motionless by eliminating many of the dropped readings (they miss less than 1 tag out of 25 every other epoch, on average). As the tags speed up, however, the performance of the large windows degrade due to many false positives. The reason the errors for the two large windows drop at higher velocities is because at that point they continuously reporting that all tags are present. Thus, while they produce a large number of false positives, they produce no false negatives.

On the other hand, the smaller windows (*Static-2* and *Static-5*), aren't able to fully compensate for lost readings. As the tag velocity increases, these schemes become comparatively better by filling in some of the missed readings without producing many false positives. *Static-5*, however, performs poorly at high tag speeds due to false positives. In a deployment where tags move with different velocities or change velocities over the course of time, an application cannot set a single static smoothing window that captures the variation in tag movement to provide accurate data.

SMURF, in contrast, consistently performs well as the tags increase speed. When the tags are motionless, it removes many of the false negatives and is competitive with the large window schemes.

As the tags increase velocity, SMURF is able to generally track the best static window. At low velocities, SMURF does well, but not as well as *Static-5*. Here, tags are not moving fast and thus mobile tag detection has little effect. As a result, SMURF's binomial sampling scheme occasionally sets its window too large: it produces roughly twice as many false positives as *Static-5*. As the tags speed up, however, mobile tag detection filters readings from tags that are exiting and thus reduces the false positives. At a tag velocity of 1 foot/second, both SMURF and *Static-5* show similar increases in false negatives, but SMURF now produces only $2/3^{rds}$ the false positives as *Static-5*.

At the highest velocities, *Static-2* performs better than

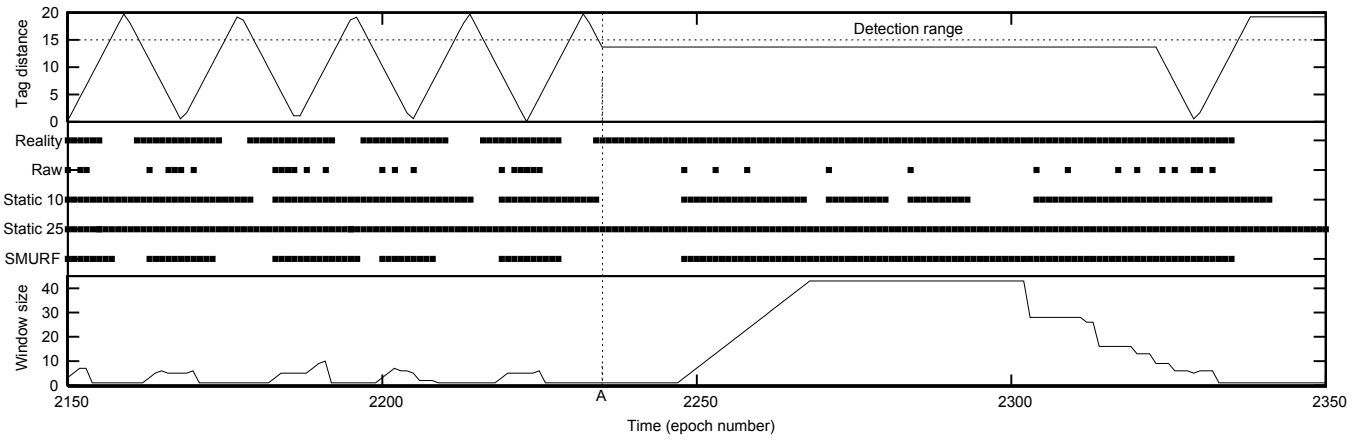


Figure 6: A 200 epoch trace of different cleaning mechanisms cleaning the readings from a single tag moving with *Fido* behavior. The top subsection shows the tag's distance from the reader (recall that the reader's detection range is set at 15 feet). The middle subsection shows the readings produced by each scheme. The bottom subsection illustrates SMURF's window size.

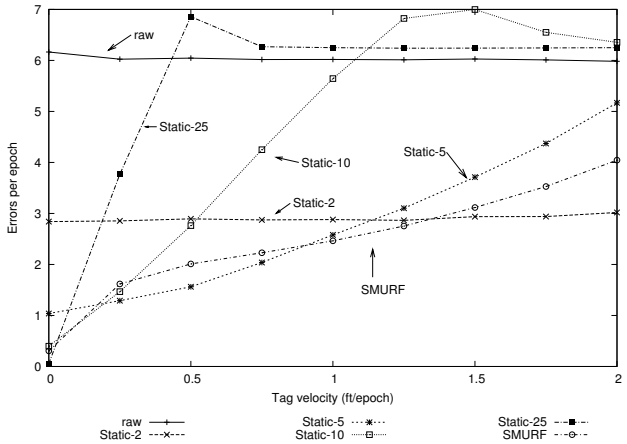


Figure 7: Average errors per epoch as tag velocities vary from 0 to 2 feet/epoch following *Pallet* behavior.

SMURF. Here, the tag velocity is approaching a fundamental limitation for any detection scheme: if the time between transitions is smaller than the window size, then the transition will be lost. In our setup, at 2 feet/epoch the time between transitions is 5 epochs. Thus, for a smoothing scheme to be able to detect a transition, the window size must be set smaller than 5 epochs. In this experiment ($MajorReadRate$ = 0.8, $MajorPercentage$ = 0.7), SMURF uses an average window size (without transition detection or mobile tag detection) of $\lceil \frac{\ln(1/\delta)}{p_i^{avg}} \rceil = \lceil \frac{\ln(1/0.05)}{0.68} \rceil = 5$. Thus, the tag velocity in this case is approaching SMURF's limit; transition detection and mobile tag detection prevent it from breaking down completely.

Experiment 3: Experiences with Real RFID Data. The previous experiments were based on a generator that created RFID data based on a simple model. Of course, real-world RFID data won't follow this model exactly. Here we describe our experiences with real RFID data and the performance of cleaning mechanisms on this data. First, we collect real RFID data under varying circumstances and examine how it differs from the model used in our generator. Second, we show that SMURF's cleaning techniques are robust to any discrepancies.

For these experiments, we recreate the conditions used in Experiment 2 through an RFID testbed deployed in the controlled environment from Section 2 using an Alien reader [3] and a single Alien "I2" [4] tag suspended in the same plane as the antenna. We gather data using tag velocities ranging from 0 to 2 feet/epoch. For the motionless tag test, we average results from data collected every 0.5 feet from 0 to 15 feet (the reader's detection range is approximately 15 feet). For the mobile tag tests, we move the tag back and forth between 0 and 20 feet from the reader. For tag velocities we are unable to produce in our testbed (1.5 and 2 feet/epoch) we collect data at lower velocities and then speed up the data traces. All runs are performed for 2000 epochs ($\approx$ 400 seconds). Additionally, we collect limited traces from two reader positions in the noisy environment, differing by $\approx$ 5 feet.

During the course of these experiments, we discovered that real RFID data differ from our model in two main ways. First, if the reader is deployed near obstacles (e.g., walls), its detection field does not follow the same shape as seen in all other positions: it is much more irregular. The detection field for a reader deployed close to a wall and metal desks, for instance, had multiple high and low detection regions. Such behavior argues for an adaptive approach to data cleaning: very small changes in the environment can cause dramatic changes in RFID reader and thus necessitates changes to any static windowing scheme.

Real RFID data differ from our model in another important way: the reader occasionally produces many more or many less readings than expected based on the reported $p_{i,t}$. For instance, the reader occasionally produces many readings with a very low $p_{i,t}$ (e.g., 0.1) in a window; SMURF is robust to such cases. In rare cases, a tag statically placed at very specific distances relative to the reader (e.g., $\approx$ 12 feet $\pm$ 2 inches for one of the reader positions) will cause the reader to occasionally produce only one reading in 5-10 epochs but report the $p_{i,t}$ of the reading as greater than 0.8. Based on this $p_{i,t}$, it is expected to see roughly 8 readings in a window of 10 epochs. In such cases, the SMURF algorithm mistakenly signals a transition and shrinks the window, causing many false negatives (e.g., 12% dropped readings versus 10% for *Static-10* and 2% for *Static-25*). As such behavior occurs rarely and only in very specific loca-

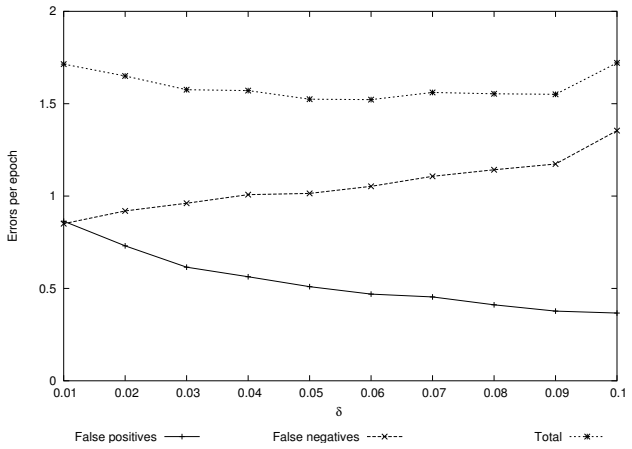


Figure 8: Errors per epoch using different values of δ.

tions with static tags, we do not expect this to be a problem in practice. If necessary, the δ parameter can be used to help alleviate the effects of these types of readings: by setting δ to 0.01, the dropped readings are reduced to 6%.

Our tests confirm our two key points. Across the different speeds and environments, there is no single static window that works uniformly well. At high speeds in the controlled environment, *Static-2* works very well, while it falters at slow speeds and in the noisy environment. On the other hand, *Static-25* works very well with a motionless tag, but performs poorly when the tag starts moving. In contrast, SMURF handles all of these cases well. When the tag moves fast in the controlled environment, it closely follows *Static-2* while at the same time competing with *Static-25* when the tag is motionless. On average, SMURF performs the best: for instance, in the controlled environment, SMURF averages 0.05 errors per epoch, compared to 0.06 for *Static-2* and *Static-5*, 0.14 for *Static-10*, and 0.18 for *Static-25*.

Finally, we compare the results (average errors per epoch) of cleaning the real data versus a similar setup in our data generator. The trends and ordering of performance across cleaning schemes was almost exactly the same between the two results. The absolute values differed slightly, but in many cases the relative difference between the errors per epoch when using the real data and the generated data was within 10%. Thus, the results derived from cleaning our generated data are comparable to the results that would be produced by cleaning real data.

Due to the difficulty in running controlled experiments with RFID technology, the remainder of the experiments we use synthetic data streams.

Experiment 4: δ as a Declarative Parameter. While the primary contribution of SMURF is the removal of the imperative window size parameter from RFID data cleaning, SMURF provides a parameter δ, where $(1 - \delta)$ is the probability of reading a tag if it exists, that allows the application to declare a preference for reduced false positives or reduced false negatives.

To illustrate the effect of this parameter, we show in Figure 8 false positives, false negatives and total errors per epoch with different values for δ using the same setup as in Experiment 1. As can be seen, the value of δ determines the relative proportion of false positives to false negatives, but has little impact on overall error.

5.3 Multi-tag Aggregate Cleaning

As stated in Section 4.3, many applications only need a count of the tagged items in the area. Here we compare techniques for accurately counting the number of tags in a reader's detection field.

We show the same static windowing schemes as the previous experiments (*Static-2*, *Static-5*, *Static-10*, *Static-25*). For count aggregates, these schemes use the equivalent of a windowed count distinct operation. For SMURF processing, we show two versions, as outlined in Section 4.3: SMURF with per-tag cleaning with summation (σ-SMURF) and SMURF using π-estimators (π-SMURF).

As π-SMURF is not capable of producing individual tag readings, we change our primary evaluation metric for multi-tag cleaning to root-mean-square error (RMS error) of the count of reported tags compared to reality.

Experiment 5: Varied Reliability and Tag Velocity. We test the accuracy of the counts produced by each scheme as either the level of tag movement or unreliability increases. We run the same tests as Experiments 1 and 2, but with more tags (100), and measure the RMS error of each scheme's output compared to reality.

To determine the count accuracy of different schemes as the tag velocity increases, we run a similar test to Experiment 2. We set $MajorPercentage$ at 0.25 and vary the tag velocity between 0 and 2 feet/epoch using *Pallet* behavior. We show the results in Figure 9.

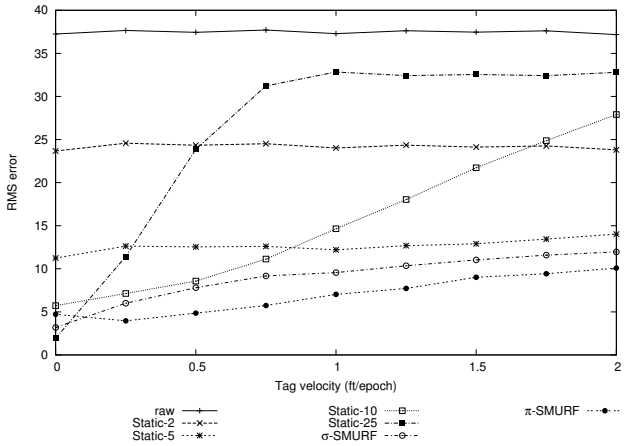


Figure 9: The RMS error of different cleaning schemes counting 100 tags moving at increasing velocities.

In most cases, both σ-SMURF and π-SMURF are more accurate than any static window. π-SMURF does particularly well here due to its unbiased nature. σ-SMURF, however, suffers from under-counting. To illustrate, we also measure the mean error of the count estimates (a measure of the bias of an estimator). At a tag velocity of 1 foot/epoch, for example, σ-SMURF has an mean error of -6.5, indicating an under-count of 6.5 items, on average. π-SMURF, in comparison, only has a mean error of -0.3. Thus, on expectation, π-SMURF provides accurate estimates.

To determine how each scheme performs as the level of reliability changes, similar to Experiment 1, we move tags using *Fido* behavior and vary the major detection region percentage. At each value of $MajorPercentage$, we measure the error of each scheme. Here, both σ-SMURF and π-SMURF are competitive with the best static window (not

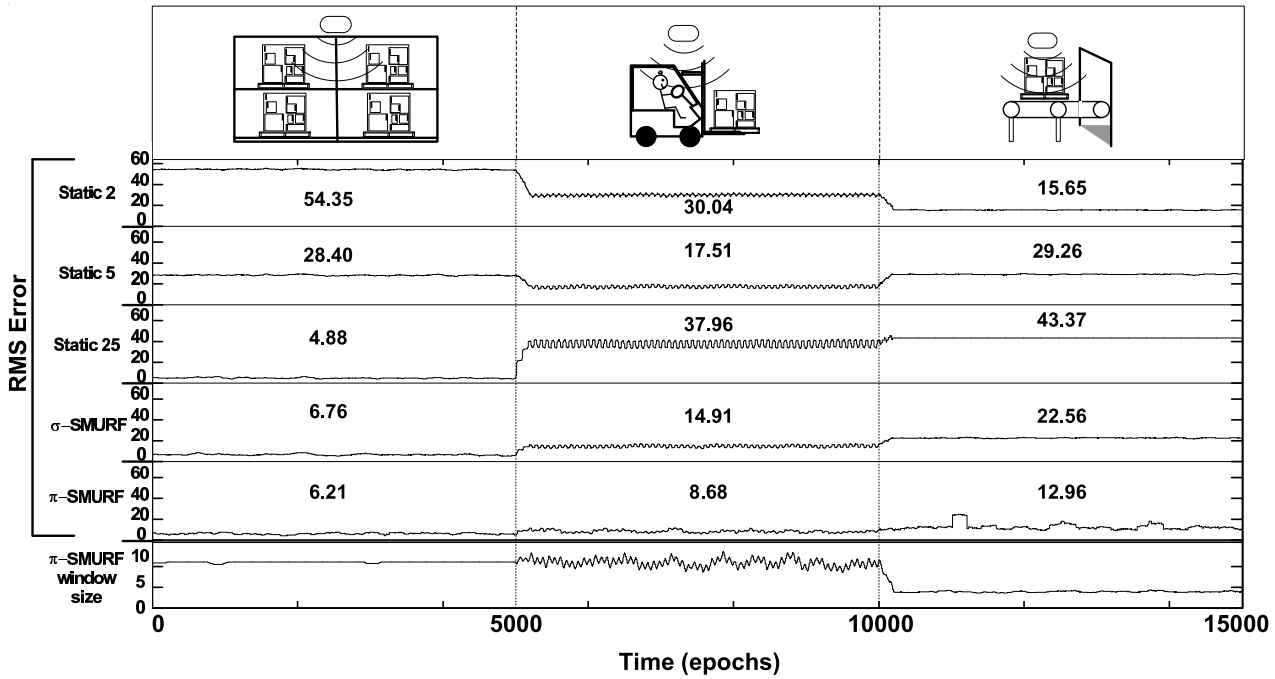


Figure 10: A simulated pallet moving through three phases in a warehouse: shelf, forklift, conveyor belt. The top traces show a 100 epoch sliding window of the RMS error for each scheme. The bottom subsection shows π-SMURF's window size (100 epoch moving window average)

shown).

Experiment 6: Tracking Counts in a Dynamic Environment. Here we illustrate how different multi-tag cleaning schemes react as conditions change over time. We simulate tag movement and reader characteristics typical of a warehouse scenario over the course of 15000 epochs. In this scenario, an application monitors the count of 100 tags placed together on a pallet as it travels through the warehouse in three phases (as depicted at the top of Figure 10):

1, *Shelf*: In the first phase, the pallet is motionless on a shelf. Due to interference from the shelf and other tags in the vicinity, the read rate is low: we set $MajorReadRate$ to 0.5 and $MajorPercentage$ to 0.5.

2, *Forklift*: After 5000 epochs, a forklift picks up the pallet and begins moving. Here, there is less reader interference due to other tags or obstructions, but the forklift reduces the major detection region ($MajorReadRate = 0.8$, $MajorPercentage = 0.25$). We simulate the forklift's motion by moving the tags at 0.5 feet/epoch.

3, *Conveyor Belt*: In the final phase, we simulate the pallet traveling on a conveyor belt. Here, the reader environment is controlled to reduce unreliability ($MajorReadRate = 0.8$, $MajorPercentage = 0.7$). The tags, however, move very fast (2 feet/epoch).

These three phases simulate realistic conditions in terms of tag and reader behavior. Any cleaning scheme should be able to handle all of these conditions to produce accurate readings describing the count of items on the pallet as it moves through the warehouse.

We clean the data produced by the tags on the pallet using different schemes and measure the RMS error during each phase as shown in the middle subsection of Figure 10. Additionally, we include a trace of a 100-epoch sliding window of the RMS error for each scheme to illustrate how accuracy changes over time.

When the pallet is on the shelf, the raw data (not shown) is very poor (reporting less than 20 tags out of 100 per epoch on average). To clean this data, a large window must be used: with either counting technique, SMURF provides a stream of count readings that are competitive with *Static-25*, the largest static window (of course, larger windows would do better here, but we omit them due to poor performance during the remainder of the experiment). The bottom portion of the figure shows the trace of a 100 epoch moving average of the window size set by π-SMURF. During the period when the tags are motionless, π-SMURF sets its window large to compensate for the unreliability of the reader.

Once the tags start moving, both SMURF techniques adjust their window sizes to balance unreliability and tag movement to outperform all static window schemes.

Finally, when the tags are moved very fast in a controlled environment, π-SMURF does particularly well as it drastically reduces its window size in reaction to the tags' movement while using π-estimators to avoid under-counting with such a small window.

As can be seen, there is no single static window that the warehouse monitoring application case use to provide accurate counts in this scenario. Using SMURF, in contrast, the application can get accurate readings throughout the pallet's lifetime without setting the smoothing window size. π-SMURF further refines its accuracy by providing an unbiased estimate.

6. RELATED WORK

Many commercial RFID middleware solutions contain configurable filters to process data produced by RFID readers [9, 19, 23, 32]. Many of these platforms explicitly incorporate data smoothing as a solution to RFID unreliability. None of these systems provide any guidance for setting the size of the smoothing window. SMURF

is designed to be incorporated into an RFID middleware system to provide self-tuning smoothing without requiring the application to set this parameter. As a result, these systems become simpler to deploy and produce more reliable data.

Several projects have explored simple techniques to clean RFID data, typically based on fixed-window smoothing [18, 25]. In one paper, the authors identify the trade-off between smoothing the data and capturing the temporal variation but provide no real solutions [18]. In previous work, we recognize the need to clean RFID data and use an approach based on declarative continuous queries [20, 21]. We show smoothed RFID data using different sized windows, but do not address how to choose the best size.

A related project explores techniques for stream data processing when the window size is not known *a priori*. The authors present efficient indexing techniques to support stream processing at multiple temporal resolutions. Such indexing techniques may assist SMURF in smoothing RFID data in multiple windows at once. We leave this exploration to future work.

The idea of using probabilistic models for sensor measurements has been explored in earlier work [15]; still, our work is the first to apply statistical techniques for adaptive RFID data cleaning. Furthermore, this scheme relies on learning and maintaining many fairly heavyweight multidimensional Gaussian models; our techniques rely on simple, non-parametric sampling estimators, specifically tuned for RFID data. Exploring interactions between the two approaches is an interesting area for future work.

Adaptive filtering has been studied in digital signal processing in wide-ranging contexts such as image analysis and speech processing [26]. Especially applicable are nonlinear digital filters, which are designed to capture transitions in the signal. For instance, AWED [26] adapts the size of a smoothing window for cleaning noisy images using a multiphase approach involving smoothing and edge detection that inspired the basic SMURF design.

7. CONCLUSIONS

While RFID technology holds much promise, the unreliability of the data produced by RFID readers is a major factor hindering large-scale deployment. Specifically, RFID readers suffer from low read rates, frequently failing to read tags that are present.

Current solutions to correct for missed readings using static smoothing filters are not adequate. Such filters require the application to set a static window size, incurring overhead for initial configuration: the window size must be set considering complex factors such as environmental conditions that affect RF signals and the range of expected tag behaviors. A more serious issue, however, is that a single smoothing window size cannot both compensate for missed readings while capturing the dynamics of tag motion. Thus, readings produced by smoothing filters using static windows do not accurately represent physical reality.

SMURF, in contrast, is a declarative, adaptive smoothing filter for RFID data. It does not require the application to set a smoothing window size: it automatically adapts its window size based on the characteristics of the underlying data stream. SMURF produces more reliable data streams by successfully balancing the tension between compensating for missed readings and capturing tag motion.

The key insight behind SMURF is its view of RFID data streams as a random sample of the tags in the physical world. Using this insight, SMURF incorporates techniques from sampling theory, such as binomial sampling and π-estimators, to guide cleaning operations in a principled, statistical manner.

In order for RFID technology to become feasible, RFID middleware must be able to produce reliable streams describing the physical world without incurring high overhead in terms of configuration and maintenance. SMURF is a significant step in this direction: RFID middleware incorporating SMURF are substantially easier to deploy and maintain and provide more reliable data.

8. REFERENCES

[1] Alien Technology. Nanoscanner Reader User Guide.

[2] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik. Aurora: a data stream management system. In *ACM SIGMOD*, 2003.

[3] Alien ALR-9780 915 MHz RFID Reader. http://www.alientechnology.com/products/rfid-readers/alr9780.php.

[4] Alien RFID tags. http://www.alientechnology.com/products/rfid-tags.

[5] Application Level Event (ALE) Specification Version 1.0. http://www.epcglobalinc.org/standards_technology/EPCglobal_ApplicationALE_Specification_v112-2005.pdf.

[6] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. *VLDB Journal*, (To appear).

[7] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Principles of Database Systems*, 2002.

[8] Barnaby J. Feder. Despite Wal-Mart's Edict, Radio Tags Will Take Time. *New York Times*, Dec 2004.

[9] C. Bornhövd, T. Lin, S. Haller, and J. Schaper. Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure. In *VLDB*, 2004.

[10] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, 2003.

[11] D.-M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Comput. Netw. ISDN Syst.*, 17(1), 1989.

[12] W. G. Cochran. *"Sampling Techniques"*. John Wiley & Sons, 1977.

[13] Daniel Dobkin and Steven Weigand. Tags vs. the World: HF and UHF Tags in non-ideal environments. *WCA RFID SIG*, June 2005.

[14] D. D. Deavours. Performance analysis of commercially available uhf rfid tags based on epcglobal's class 0 and class 1 specifications. *RFID Alliance Lab*, 2004.

[15] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB Conference*, 2004.

[16] EPC Tag Data Specification Version 1.1. http://www.epcglobalinc.org/standards_technology/EPCTagDataSpecification11rev124.pdf.

[17] EPCGlobal, Inc. http://www.epcglobalinc.org/.

[18] K. P. Fishkin, B. Jiang, M. Philipose, and S. Roy. I sense a disturbance in the force: Unobtrusive detection of interactions with rfid-tagged objects. In *Ubicomp*, 2004.

[19] A. Gupta and M. Srivastava. Developing auto-id solutions using sun java system rfid software. Oct 2004.

[20] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. A Pipelined Framework for Online Cleaning of Sensor Data Streams. In *ICDE*, 2006.

[21] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative Support for Sensor Data Cleaning. In *Pervasive*, 2006.

[22] Laurie Sullivan. RFID Implementation Challenges Persist, All This Time Later. *Information Week*, Oct 2005.

[23] Manage Data Successfully with RFID Anywhere Edge Processing. http://www.ianywhere.com/developer/rfid_anywhere/rfidanywhere_edgeprocessing.pdf.

[24] R. Motwani and P. Raghavan. *"Randomized Algorithms"*. Cambridge, 1995.

[25] M. Philipose, K. Fishkin, D. Fox, and D. Hahnel. Mapping and Localization with RFID Technology. Technical Report IRS-TR-03-014, Intel Research, Dec 2003.

[26] I. Pitas and A. N. Venetsanopoulos. *Nonlinear digital filters: principles and applications*. Kluwer, Boston, MA, 1990.

[27] Ryan J. Foley. New UW lab helps with product ID. *The Capital Times*, Aug 2005.

[28] C.-E. Särndal, B. Swensson, and J. Wretman. *"Model Assisted Survey Sampling"*. Springer-Verlag New York, Inc. (Springer Series in Statistics), 1992.

[29] Senosrmatic Agile 2 915Hz RFID Reader. http://www.sensormatic.com/RFID/stationary/.

[30] SensorID Series 2 Agile Reader Query Protocol, Apr 2004.

[31] L. Sirico. The Cost of Compliance II: The Details. http://mrrfid.com/index.php?itemid=50.

[32] F. Wang and P. Liu. Temporal Management of RFID Data. In *VLDB*, pages 1128–1139, 2005.

[33] R. Want. The Magic of RFID. *ACM Queue*, 2(7):40–48, 2004.