The VLDB Journal manuscript No.
(will be inserted by the editor)

Shawn R. Jeffery · Michael J. Franklin · Minos Garofalakis

An Adaptive RFID Middleware for Supporting Metaphysical Data Independence

Received: date / Accepted: date

met·a·phys·ics: *a division of philosophy that is concerned with the fundamental nature of reality and being [2].*

Abstract Sensor devices produce data that are unreliable, low-level, and seldom able to be used directly by applications. In this paper, we propose *Metaphysical Data Independence* (*MDI*), a layer of independence that shields applications from the challenges that arise when interacting directly with sensor devices. The key philosophy behind MDI is that applications do not deal with any aspect of physical device data, but rather interface with a high-level reconstruction of the physical world created by a sensor infrastructure. As a concrete instantiation of MDI in such a sensor infrastructure, we detail MDI-SMURF, an RFID middleware system that alleviates issues associated with using RFID data through adaptive techniques based on a novel statistical framework.

Keywords Data cleaning · RFID technology · Statistical sampling · Sensor-based applications

1 Introduction

With the widespread deployment of physical sensing devices such as wireless sensor networks and RFID technology, the physical world is being brought ever closer to the digital world: RFID provides enterprises with up-to-the-second information on their supply chains [23]; wireless sensor networks enable unprecedented visibility into environmental and structural processes [60]; and

Shawn R. Jeffery
UC Berkeley
E-mail: jeffery@cs.berkeley.edu

Michael J. Franklin
UC Berkeley
E-mail: franklin@cs.berkeley.edu

Minos Garofalakis
Yahoo! Research and UC Berkeley
E-mail: minos@yahoo-inc.com
Work done while at Intel Research Berkeley

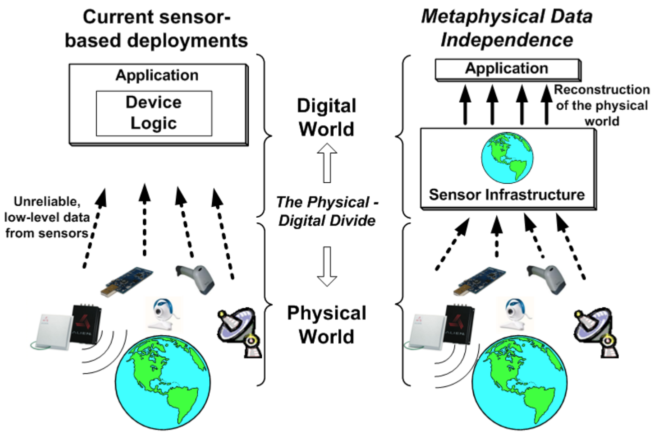

Current sensor-based deployments
Application
Device Logic
Unreliable, low-level data from sensors
Digital World
The Physical - Digital Divide
Physical World
Metaphysical Data Independence
Application
Reconstruction of the physical world
Sensor Infrastructure

Fig. 1 Today's sensor-based applications are dependent on the underlying sensors and thus complex and hard to manage. Metaphysical Data Independence allows applications to interact with a reconstruction of the physical world in the digital world, greatly simplifying application deployment.

ubiquitous computing technology is changing the way we interact with our surroundings [3].

The Physical-Digital Divide. Sensor-based applications use data about the physical world to make decisions or perform other tasks. There is a wide gulf, however, between the data produced by devices in the physical world and the needs and requirements of applications in the digital world. We term this rift the *physical-digital divide* [32].

The crux of the problem is that the raw data provided by a set of sensor devices do not adequately represent the physical world. Rather, the data are noisy due to outliers and mis-calibrated sensors, incomplete as a result of dropped messages, and coarse-grained in both time and space as devices cannot sample infinitely nor be deployed with complete coverage. Furthermore, in many cases the application desires data for which no sensing device exists (e.g., [12,52]). For instance, a digital home application is concerned about the locations of the house's inhabitants, but there is no device that provides exactly these data.

Metaphysical Data Independence. Sensor-based deployments typically include complex logic to map from low-level, dirty sensor data to high-level application concepts and maintain this mapping over time [11,60]. Any errors or fluctuations in the underlying devices must be handled directly by the application. As a result, current sensor-based applications tend to be complex, brittle, and hard to evolve.

To address this problem, we introduce a new layer of data independence, *Metaphysical Data Independence*[1](*MDI*), that shields sensor-based applications from the challenges associated with managing and accessing physical sensor devices. Just as traditional RDBMSs use *physical* and *logical* data independence to hide the complexity and changes of the physical data storage and base schema, respectively, MDI hides the complexity of the physical sensing devices from sensor-based applications.

The key philosophy behind Metaphysical Data Independence is that sensor data should be abstracted as data about the physical world; that is, applications should interact with a reconstruction of the physical world in the digital world, as if the physical-digital divide did not exist (as illustrated in Figure 1).

To represent this reconstruction, we need to model the physical world at a level appropriate for use across many applications. The challenge in defining this model is to balance the tension between hiding the complexity of sensor data and providing data at a semantic level that is appropriate for a large class of applications.

To this end, we outline a data model for sensor-based applications based on the concepts of *objects*, *attributes*, and *uncertainty estimates*. Objects in this model correspond to real-world entities such as people, products, and rooms. Each object has a set of associated attributes such as location, temperature, and velocity. Due to the uncertainty of sensor data, an integral part of this model is uncertainty estimates. The set of all objects, their attributes, and associated uncertainty estimates at a given time-step make up the "state of the world" at that time-step.

The fundamental aspect of this model and interface is that they expose no information about the underlying physical devices. Just as an application interacting with a traditional relational database using SQL does not have to know anything about the underlying storage models, disk usage, or data format, an application using MDI can be oblivious to details of the devices on which it is deployed.

[1] The term "metaphysical" is a loose reference to Plato's *Divided Line* philosophy concerning the real world and the perceived world [48]. He conjectured that the physical world is only accessible through a person's imperfect senses and thus knowledge and reason must be used to guide any perception of the real world.

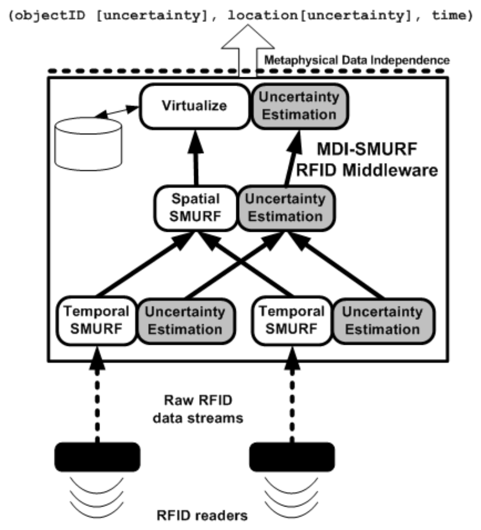


Fig. 2 The architecture of MDI-SMURF.

MDI-SMURF. In order to realize the MDI interface, many mechanisms need to work in concert to access, clean, process, and virtualize data from sensing devices to adequately reconstruct the physical world in the digital world. In this paper we present one such solution, MDI-SMURF,[2] an implementation of Metaphysical Data Independence for RFID deployments. The principle contribution of MDI-SMURF is a novel statistical framework that enables it to continually and adaptively correct for the temporal and spatial errors associated with RFID data and produce data corresponding to the MDI interface. We place the work presented in [35] in the broader context of supporting MDI for RFID deployments. Furthermore, we outline how MDI-SMURF extends the concepts presented in [35] with additional functionality to handle all aspects of the physical-digital divide.

MDI-SMURF is an RFID middleware platform organized as a pipeline of processing stages [33,34] with an associated uncertainty-tracking shadow pipeline [53] as shown in Figure 2. RFID data from readers flow into *Temporal-SMURF*, a *smoothing filter* that uses its statistical framework to correct for dropped readings common in RFID data streams. Additionally, Temporal-SMURF estimates the resulting uncertainty of the cleaned readings. These cleaned readings are then streamed into *Spatial-SMURF*, a module that extends Temporal-SMURF's statistical framework to address errors and semantic issues that arise from multiple RFID readers deployed in close proximity. A simple translation module (*Virtualize*) converts the temporally and spatially cleaned readings to MDI readings with the following schema: `(objectID[uncertainty], location[uncertainty], time)`.

This paper is organized as follows. We first provide background on Metaphysical Data Independence, the un-

[2] "SMURF" stands for *Statistical sMoothing of Unreliable RFid data*

derlying philosophy of MDI-SMURF (Section 2). In Section 3, we give an overview of RFID technology and its current shortcomings. We then detail MDI-SMURF in Sections 4 - 6. Finally, we review related work and conclude in Sections 7 and 8, respectively.

2 Background: The Physical-Digital Divide and MDI

In this section, we provide a high-level background on the challenges associated with sensor-based deployments and outline our philosophy, Metaphysical Data Independence, for addressing these issues.

2.1 The Physical-Digital Divide

We illustrate the shortcomings of current sensor technologies using examples from two of the more common types of sensing devices, RFID readers and wireless sensor networks. While each type of sensing device has specific challenges associated with processing the data it produces, there are many issues that are common across devices.

Sensor Data Unreliability. Applications typically need a reliable and complete picture of reality to operate correctly. Sensors, however, tend to produce incomplete and unreliable data.

Sensors often employ low cost, low power hardware and wireless communication, which lead to frequently dropped messages. For example, the observed *read rate*, or percentage of tags in a reader's vicinity that are actually reported, in real-world RFID deployments is often in the $60-70\%$ range [33,38]; in other words, over 30% of the tag readings are routinely dropped. Wireless sensors also demonstrate similar errors. For instance, in a wireless sensor network experiment at the Intel Research Lab in Berkeley, each sensor delivered, on average, only 42% of the data it was asked to report [31].

Not only are readings frequently dropped, but often individual sensor readings are unreliable. In a sensor network deployment in Sonoma County, CA, for example, 8 out of 33 temperature-sensing motes failed, but continued to report readings that slowly rose to above 100^o Celsius [59].

Thus, the data produced by sensors must be appropriately cleaned to compensate for these failures before they can be used by any application. Incorporating cleaning logic in applications greatly increases the complexity of the application.

Granularity Mismatch. Sensor-based applications tend to have specific notions of time and space that usually do not correspond to the sensing granularity of the underlying devices [19,34].

Temporally, the actual sensing granularity of the devices may be coarser than an application desires due to power or bandwidth limitations as in the case of wireless sensors, or it may finer such as with the high sample rate of RFID readers.

Similarly, the spatial sensing granularity of devices may not match an application's notion of space. One device may monitor multiple application-level spatial units, such as rooms or shelves. Conversely, there may be multiple devices that monitor the same spatial unit. For instance, RFID deployments usually deploy multiple readers in close proximity to ensure full coverage of the area of interest.

This mismatch in sensing granularity potentially causes semantic errors. For instance, redundant readings in space could lead to an application seeing an object in two places at the same time.

The Semantic Gap. Usually, sensor-based applications view the world as high-level concepts; sensing devices, on the other hand, produce low-level data that often has little meaning to the application.

In many cases, applications are interested in data for which no physical sensing device exists. Common examples of such data include people in pervasive applications [3], products in supply-chain management applications [23], or attributes of manufactured goods in industrial processes [52]. These high-level application concepts must be derived through the combination of data from multiple devices as well as other sources of data.

Translating from low-level device readings to application-level concepts involves intimate knowledge of the environment, devices, and the data they produce, thus complicating application development and deployment.

Variability in Sensor Deployments. Not only do sensor data exhibit the issues described above, but the nature of these issues changes over time and from deployment to deployment.

Wireless sensor motes, for instance, lose accuracy as their batteries wear [60]. RFID readers produce different quality data and their detection fields vary depending on the environment in which they are deployed [24]. For example, during a series of RFID reader tests in a variety of environments, the quality of readings from readers in two different rooms, next door to each other, varied greatly. (See Section 3 for a detailed description of this experiment.)

As a result of such variability, sensor deployments not built to adjust and adapt to varied environments tend to produce erroneous data as conditions change.

2.2 Metaphysical Data Independence

As illustrated by the issues above, there is significant complexity involved in converting data produced by sensing devices into data that can be used in an application. Motivated by these complexities, we propose a new layer of data independence, *Metaphysical Data Independence*

(*MDI*). MDI defines a separation of concerns between the application logic and the logic needed to access and manage the data from physical devices. The key philosophical statement behind MDI is that the specifics of the underlying devices are abstracted away behind a model of the physical world that represents a reconstruction of this world in the digital world.

Object Model. Our model consists of objects, attributes, and uncertainty estimates.

Objects: At the core of our model is the concept of an object. An object is loosely defined as any entity in the real world to which an application may uniquely refer. For instance, a digital home may have people objects, room objects, and pet objects. The actual objects defined for each deployment may vary dramatically, but experience shows that within an application the appropriate objects are evident (e.g., [41,57]).

Attributes: Associated with each object is a set of attributes that describe the state of that object. Attributes are attached directly to the object, and in general have little meaning without this association (e.g., temperature means nothing unless it is the temperature of some object). The primary attributes of any real-world object are time and space (location); these attributes are typically included for all objects. Examples of other attributes include temperature and velocity. An attribute may also be a complex multi-valued field instead of a single value. For instance, some attributes may describe the contour of a value in a certain area [27].

While an attribute is dependent on the object that it describes, it is independent of any physical sensing device. There may be multiple devices that observe the same value. For instance, wireless motes and some RFID tags [58] both sense light levels. Soft sensors [49], on the other hand, may be used to combine data from multiple types of sensors to derive an attribute. Alternatively, no device may be employed at all to sense a particular attribute; model-based sensing [18], cached values, or archived data may be used. In any case, such acquisition and processing is hidden by the model.

Uncertainty: Due to the limitations of sensing devices, an estimate of the uncertainty in the reported data is essential. Thus, throughout this model is the notion of uncertainty. Uncertainty serves as the unifying means by which device-dependent processing can be hidden.

We use uncertainty estimates at multiple levels. Attached to each object is a confidence of existence, used in a similar manner to that defined in [44,53]. Additionally, each attribute has some uncertainty value(s) with which it is associated. The exact representation of these values are dependent on the type of attribute. For instance, the uncertainty of a particular value for a temperature attribute may be represented as a range and confidence [18], while uncertainty in location could be described with a spatial probability distribution function. Regardless of the description method, uncertainty is expressed in a device-independent manner.

The set of all objects, their attributes, and uncertainty at a given time-step make up the "state of the world" at that time-step.

Interface. Data expressed in this model can be accessed as a stream of object states, ideally through declarative means using languages such as CQL [7] or those defined as part of the HiFi [51] or SASE [65] projects.

Sensor-based applications have differing needs in terms of data; the interface should allow applications to specify selection predicates on which objects and attributes are necessary. Given these predicates, the sensor infrastructure can optimize access to these data. Similarly, since different applications have different data quality requirements, the interface needs to allow applications to request the desired level of data quality through uncertainty predicates. Such predicates can help guide the infrastructure on which devices to use and by which means to produce the requested data. Further, some sensor data processing is inherently application-specific. Thus, the interface should allow user-defined code to be pushed down into the sensor infrastructure to assist in data processing.

As there are potentially many applications accessing data from the same set of devices, but with different requirements, the sensor infrastructure should support multiple streams of MDI data. Generating this output should be done in such a way as to minimize access to the devices themselves and to maximize sharing across streams [37]. Further, the interface should provide seamless access to present, past, and future MDI data. That is, access to streaming data (present), archival data (past), and future predictions based on models should all happen through the same interface in a unified manner.

Finally, we note that while there will always be uses of sensor devices that demand low-level access to the data and devices, such as advanced scientific monitoring applications or tight-loop sensor-actuator systems, we feel that there is a large class of emerging sensor-based applications that would benefit from the higher level of abstraction with which to interact with sensor data that MDI provides.

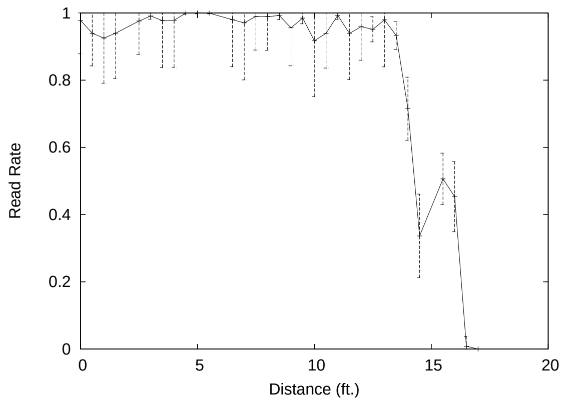
Having outlined our MDI approach for dealing with sensor data, in the remainder of the paper we focus on a specific MDI-based infrastructure for dealing with RFID data.

3 RFID Technology

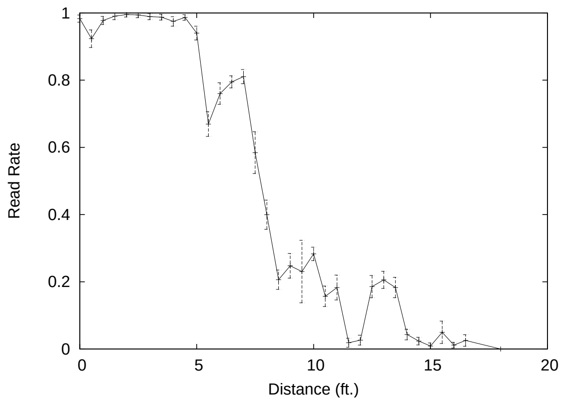
In this section give a brief background on RFID technology and the challenges facing current deployments.

3.1 RFID Background

RFID (Radio Frequency IDentification) is an electronic tagging and tracking technology designed to provide non-



(a)Alien reader with Alien Squiggle tag in a controlled environment.



(b)Sensormatic reader with Alien I2 tag in a noisy environment.

Fig. 3 Read rate of a single tag at varying distances from the reader in different environments. Error bars represent ± one standard deviation.

line-of-sight identification [63]. 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 tag 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 [22]). 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 typically grouped into what we term *epochs*.[3] The epoch size, which may be specified as a number of interrogation cycles or as a unit of time, is configured as part of the initial reader hardware setup. A typical epoch range is 0.2-0.25 seconds [1,56]. 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 last read. Readers store this information internally in a *tag list* (Table 1) which is periodically transferred to readers' clients.

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

Table 1 Example reader tag list.

RFID Reader and Tag Performance.

To better understand the properties 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 number of interrogations) for 100 epochs.

Our profiling experiments use two types of readers, the Alien ALR-9780 [4] and the Sensormatic Agile 2 [55], with three types of tags (Alien "I2", "M", and "Squiggle" [5]). 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 [24]. Our second profiling environment, a lab with metal objects such as desks and computer equipment, represents a noisy environment.

Figure 3 depicts the results from two different profiling experiments showing the read rate of the tag at distances ranging from 0 to 20 feet. The plots shown here are representative of the 8 different profiles we collected; all other experiments yielded a curve similar in shape to one of these two plots.

All of the profiles we collected 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* [29], 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 fluctuates as it drops 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 corre-

[3] In ALE terms, an epoch is a *read cycle* [6].

sponding 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 3(a), whereas it makes up only 25% of the range in the profile in Figure 3(b). All of our experiments showed similar behavior. Note that these findings are consistent with the results of in-depth commercial studies of the performance of many different tags and readers under highly-controlled conditions [17].

We also test the readers to determine how they respond to the presence of *multiple tags* in their detection ranges. For these tests, 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 are similar to the single tag case (there is still a separation between a major and minor detection region), the read rate in the major detection region typically drops to around 80%. Additional tests show that the read rate in the major detection region stays somewhat constant with increasing numbers of tags, at least up to 25 tags in the reader's detection range.

In the remainder of the paper, we use these observations in the design of MDI-SMURF's cleaning mechanisms and in the implementation of an RFID data generator for evaluating our techniques.

3.2 Smoothing Filters for RFID Data Cleaning

A primary factor limiting the widespread adoption of RFID technology is the *unreliability* of the data streams produced by RFID readers [38]. The standard data-cleaning mechanism for today's RFID middleware 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 [28,40]. The goal 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 continuous query (e.g., in CQL [7]) as shown in Query 1 (for a 5 second window). This query states that if the tag appears at least once in the window, it is considered to be present for the entire window.

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
```

Static Window Smoothing. 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 4): *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).

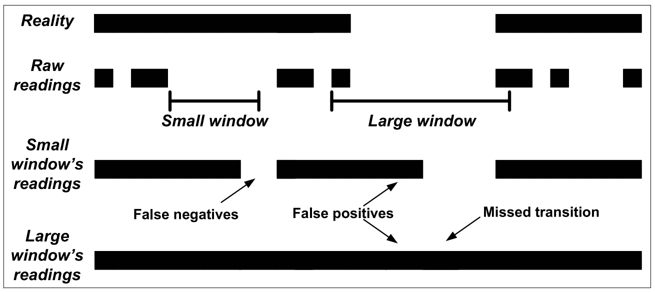


Fig. 4 Tension in setting the smoothing-window size for tracking a single tag (dark bars indicate the tag is present/read): small windows fail to fill in dropped readings (false negatives); large windows fail to capture tag movement (false positives).

– *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 the physical surroundings [16, 24].

– *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).

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 motionless on a shelf exhibits a different movement pattern from a tag on a conveyor belt.

As a result, a considerable challenge to deploying RFID-based applications is ascertaining the characteristics of the environment and configuring the middleware to take into account the above factors. Furthermore, no single window size is expected to be effective over the lifetime of a deployment as both the reader reliability and tag behavior may vary dynamically; thus, either the window size must be repeatedly reconfigured, or the quality of the data suffers.

A second major problem with fixed-window smoothing techniques is the use of a single window size for all tags in a deployment. Different subsets of tagged objects may behave very differently from others. For instance in a warehouse environment, some tagged items may be placed on a shelf while others are moved on forklifts. The best smoothing window size for each of these groups of tags is potentially different.

Adaptive Windowing for MDI. The key to masking the unreliability of RFID data in support of MDI is to hide the window size from the application and instead automatically determine the window size initially and then adapt it as the system runs. To this end, we have developed Temporal-SMURF, an adaptive smoothing filter that does not require the application to set the window size; instead, Temporal-SMURF uses statistical properties of the data to continually adjust the window size in response to the data to provide a reliable stream of RFID readings. Furthermore, Temporal-SMURF adapts its 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.

4 RFID Data Cleaning with Temporal-SMURF

In this section we present Temporal-SMURF, our adaptive smoothing filter for cleaning RFID data.

4.1 RFID Data: A Statistical Sampling Perspective

Temporal-SMURFcaptures 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.

Consider an epoch t. Recall from Section 3 that an epoch is a reader's unit of detection. Epochs represent our basic time units, many of which can be combined to make up a smoothing window [28,40]. 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") during that epoch. Temporal-SMURF views S_t as an *unequal probability random sample* of the tag population.

Temporal-SMURF uses a *per-epoch sampling probability* $p_{i,t}$ for each tag that represents the probability that tag i is detected in epoch t. While there are many possible means of deriving this value, in this work we utilize the response-count information stored in the reader's tag list (Table 1). Specifically, for each tag $i \in S_t$, Temporal-SMURF employs the response-count information for tag i in conjunction with the known number of interrogation cycles per epoch to derive $p_{i,t}$. 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 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.

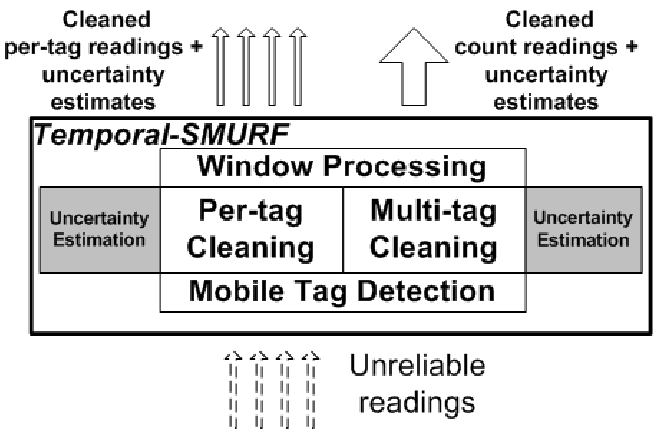


Fig. 5 Temporal-SMURF's internal architecture.

The key insight of viewing each RFID epoch as a "sampling trial" enables Temporal-SMURF's novel, statistical-driven perspective on adaptive RFID data cleaning. In a nutshell, Temporal-SMURF views the observed readings within a smoothing window 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, Temporal-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), Temporal-SMURF uses *sampling-based estimators* [15,54] to provide accurate, *unbiased* estimates for tag-population aggregates (e.g., counts), and thus avoids the systematic under-counting of conventional smoothing techniques. Thus, Temporal-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.

4.2 Temporal-SMURF

Temporal-SMURF's overall architecture is presented in Figure 5. Temporal-SMURF contains 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, Temporal-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*. Finally, Temporal-SMURF contains shadow modules for both per-tag and multi-tag smoothing to calculate uncertainty estimates.

Sliding Window Processing. As with any window-based cleaning scheme, Temporal-SMURF produces an output reading for a given tag ID if there exists at least one reading for that tag within the smoothing window [28,40]. Temporal-SMURF's sliding-window processor implements two basic modifications to conventional RFID smoothing filters: (1) partitioned RFID smoothing, and (2) epoch-based mid-window slide.

To handle subsets of tagged objects that behave differently from others, Temporal-SMURF's cleaning techniques adapt the smoothing-window size at a much finer granularity than traditional smoothing mechanisms. At one extreme, when tracking individual tag movements, Temporal-SMURF runs its adaptive sliding-window processing *per tag ID*. In general, the granularity of Temporal-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, such as through the `Partition By` clause in CQL.

As epochs are a sample cycle in Temporal-SMURF's sampling-based model of RFID data, Temporal-SMURF slides its windows by a single epoch (as opposed to a time period or by tuples). Furthermore, Temporal-SMURF produces readings with a timestamp corresponding to the midpoint of the window after the entire window has been seen. This behavior 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 this approach yields the most reliable readings.

4.3 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. Temporal-SMURF must set its window size such that it provides completeness for periods of dropped readings as well as accurately captures transitions for periods where the tag has left. To help differentiate between these two behaviors and to guide subsequent window adaptations, Temporal-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. Temporal-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 shown in Section 4.1. Note that we assume that within an appropriately-sized window, the $p_{i,t}$s will be relatively homogeneous and thus averaging is a valid estimate of the actual $p_{i,t}$.[4] 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:

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

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

Per-Tag Adaptive Window Size Adjustment. With our binomial sampling model in place, we first consider the problem of setting Temporal-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 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 $> 1 - \delta$. ∎

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

[4] 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 4.5) allows Temporal-SMURF to appropriately size its window to capture tag dynamics.

$(1-p_i^{avg})^{w_i}$. Setting this probability $\le \delta$ and taking logs gives $w_i \ln(1-p_i^{avg}) \le \ln \delta$. Combining this result with the inequality $-x \ge \ln(1-x)$ for $x \in (0,1)$, we see that it suffices to require that $-w_i p_i^{avg} \le \ln \delta$, or, equivalently, $w_i \ge \frac{\ln(1/\delta)}{p_i^{avg}}$. 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 settings for δ (i.e., less than 0.1) do not have a large effect on the overall window size.

While using a smoothing-window size as suggested by Lemma 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 4). In order to avoid smoothing over transitions and producing many false positives, Temporal-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 is crucial: readings *will* routinely be lost (e.g., for tags outside the reader's major detection region (Figure 3)), and thus an overly-sensitive transition detection mechanism can result in failing to compensate for lost readings. On the other hand, a coarse detection mechanism can miss true signal transitions, resulting, once again, in false positives. Temporal-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, Temporal-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[5]:

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

[5] More conservative, non-CLT-based probabilistic criteria, e.g., based on the Chebyshev or Chernoff bounds [42], can also be used here.

Estimating Data Quality for Cleaned Per-Tag Readings. Through its principled, statistical sampling framework, Temporal-SMURF can also compute and attach *uncertainty estimates* for each tag reading emitted to higher-level applications. More specifically, consider the value $|S_i|$ for tag i in the current window, and let $w_i p_i^{avg} - |S_i| = \eta$, where η does not satisfy the transition condition (1) above. Then, we can attach an uncertainty indicator with the emitted reading of tag i, indicating our level of confidence in the presence of tag i (during the window). This is done by estimating an upper bound on the probability

$$\begin{aligned}&\mathbf{Pr}[w_i p_i^{avg} - |S_i| = \eta \mid \text{tag } i \text{ is present}]\\ &\quad < \mathbf{Pr}[|w_i p_i^{avg} - |S_i|| \ge \eta \mid \text{tag } i \text{ is present}].\end{aligned}$$

Such upper bounds can be estimated based on either a CLT argument using the η-percentiles of the Normal distribution, or standard tail inequalities for the binomial distribution, such as Chernoff bounds [42]. For instance, using Chernoff bounds, we have

$$\begin{aligned}&\mathbf{Pr}[|w_i p_i^{avg} - |S_i|| \ge \eta \mid \text{tag } i \text{ is present}]\\ &\qquad \le 2e^{-\eta^2/(2w_i p_i^{avg})}.\end{aligned}$$

For our experimental evaluation (Section 5), we use the above equation based on Chernoff bounds to compute Temporal-SMURF's uncertainty estimates.

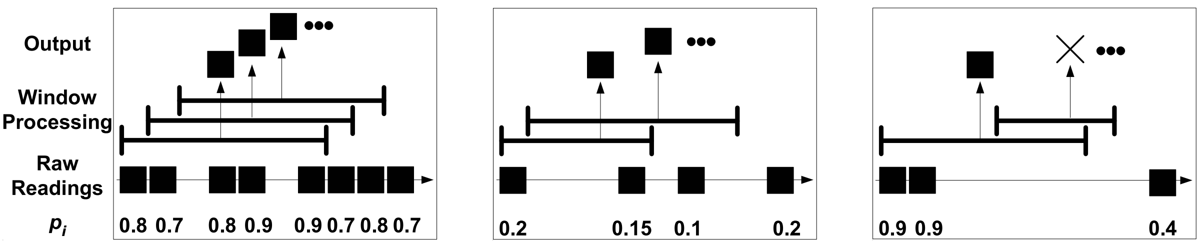
SMURF Per-Tag Cleaning Algorithm. A pseudocode description of Temporal-SMURF's adaptive per-tag cleaning algorithm is depicted in Algorithm 1. Temporal-SMURF employs the common Additive-Increase/Multiplicative-Decrease (AIMD) paradigm [13] to adjust its window size for each tag i, based on guidance from its binomial-sampling model as discussed above.[6]

Algorithm 1 Temporal-SMURF Adaptive Per-Tag Cleaning

```
Require: $T$ = set of all observed tag IDs
         $\delta$ = desired completeness confidence
  $\forall i \in T$, $w_i \leftarrow 1$
  while (getNextEpoch()) do
    for ($i$ in $T$) do
      processWindow($W_i$)
      $w_i^* \leftarrow$ completenessSize($p_i^{avg}$, $\delta$)        // Lemma 1
      if ($w_i^* > w_i$) then
        $w_i \leftarrow \max\{\min\{w_i + 1, w_i^*\}, 1\}$
      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
```

Temporal-SMURF runs its sliding-window smoothing 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

[6] Note that our algorithm uses only simple mathematical operations; thus, the overhead beyond traditional smoothing techniques is minimal.



(a)Normal sliding window processing for tag i in Temporal-SMURF. At each epoch, Temporal-SMURF emits a reading with a timestamp corresponding to the midpoint of the window.

(b)Ensuring completeness. In the left-most window, the 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.

Fig. 6 Graphical depiction of per-tag cleaning in Temporal-SMURF.

processing Temporal-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, Temporal-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, Temporal-SMURF consults its binomial-sampling model to determine the number of epochs necessary to ensure completeness with high probability (completenessSize(p_i^{avg}, δ)), based on Lemma 1. If the required size w_i^* exceeds the current window size $w_i = |W_i|$, Temporal-SMURF grows its current window size for tag i additively. This additive window growth rule allows Temporal-SMURF to incrementally monitor the tag's readings as the window grows and thus remain responsive to changes in the underlying signal.

If the current window size satisfies the completeness requirement, then Temporal-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, Temporal-SMURF multiplicatively decreases the size of its current smoothing window for tag i (i.e., divides it in half). By multiplicatively decreasing its window size, Temporal-SMURF can quickly react to detected transitions, while at the same time avoiding over-reaction in the unlikely event of an incorrect transition detection. Of course, if the completeness requirement is met and no transition is detected, Temporal-SMURF continues with its current window size for tag i.

To summarize, Figure 6 graphically depicts some example scenarios in Temporal-SMURF's basic per-tag cleaning scheme.

4.4 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 Temporal-SMURF's per-tag cleaning algorithms (Section 4.3) 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 overhead: Temporal-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 these problems, Temporal-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). Temporal-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 suffer from consistent underestimation errors.

Temporal-SMURF employs its unequal-probability random sampling model to correct for this underestimation bias through the use of *π-estimators* (also known as *Horvitz-Thompson estimators*) [54] to approximate population aggregates.[7] 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:

$$\hat{N_W} \ = \ \sum_{i\in S_W} \frac{1}{\pi_i}.$$

In other words, the count π-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 $\pi_i \cdot 1/\pi_i = 1$); thus, the single occurrence of tag 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, $E[\hat{N_W}] = N_W$ [54]. Thus, by weighting with sampling probabilities, Temporal-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 [54]:

$$\hat{\mathrm{Var}}[\hat{N_W}] \ = \ \sum_{i\in S_W} \frac{1-\pi_i}{\pi_i^2}. \tag{3}$$

Of course, even though Temporal-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, Temporal-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 Temporal-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 used in Lemma 1, we set the upper bound for Temporal-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 ($\geq 1-\delta$). Note that 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 Temporal-SMURF employs π-estimators to correct for missed readings, such a pessimistic window could result in over-estimation errors.

Temporal-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 Temporal-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 4.3, 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, Temporal-SMURF detects that a statistically-significant transition in population count has occurred in the second half of W if the following condition is satisfied:

$$|\hat{N_W} - \hat{N_{W'}}| \ > \ 2\left(\sqrt{\hat{\mathrm{Var}}[\hat{N_W}]} + \sqrt{\hat{\mathrm{Var}}[\hat{N_{W'}}]}\right). \tag{4}$$

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 (4) attempts to accurately capture such significant drop-offs within the current window, and allows Temporal-SMURF to adaptively shrink its smoothing window size. Second,

[7] Although our discussion here focuses primarily on tag counts, Temporal-SMURF's π-estimator scheme 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.

while Condition (4) 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 (4) 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 is that as the sub-range within W decreases, the variability of the $\hat{N}_{W'}$ estimate goes up, making it difficult to detect very short-lived transitions. Our empirical results demonstrate that using Condition (4) for just the second-half window $W' = (t - w/2, t]$ is sufficient to provide accurate population-count estimates to applications.

Estimating Data Quality for Cleaned Multi-Tag Readings. Similar to the single-tag case, Temporal-SMURF's statistical sampling foundation allows for uncertainty indicators to be attached to derived π-estimates and emitted to higher-level applications. In the multi-tag case, such indicators take the form of appropriate *confidence intervals* for the $\hat{N}_W$ π-estimators based on their unbiasedness and observed sample variances. Such intervals can be computed through standard probabilistic methods, e.g., using the Normal distribution based on CLT arguments, or using the (more conservative) Chebyshev bound [42]:

$$\mathbf{Pr}[|\hat{N}_W - N_W|] \geq \eta] \;\leq\; \frac{\hat{\text{Var}}[\hat{N}_W]}{\eta^2}.$$

For Temporal-SMURF's confidence interval, η, we use the following equation based on the Chebyshev bound above:

$$\eta = \sqrt{\frac{\hat{\text{Var}}[\hat{N}_W]}{\alpha}}, \tag{5}$$

where $1-\alpha$ is the desired confidence level; that is, for $(1-\alpha)$ percent of the readings reported by Temporal-SMURF, we expect the true value of the tag count aggregate to be in the range $\hat{N}_W \pm \eta$.

SMURF Multi-Tag Cleaning Algorithm. Algorithm 2 depicts the pseudo-code for Temporal-SMURF's multi-tag cleaning scheme that incorporates the above techniques. Similar to per-tag cleaning, Temporal-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.

For each epoch, Temporal-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.

Algorithm 2 Temporal-SMURF Adaptive Multi-Tag Cleaning

```
Require: $\delta$ = desired average completeness confidence
  $w \leftarrow 1$
  while (getNextEpoch()) do
    processWindow($W$)
    $W \leftarrow$ slideWindow($w$)
    $w^* \leftarrow$ completenessSize($p^{avg}$, $\delta$)        // Lemma 1
    if (detectTransition($\hat{N}_W$,$\hat{N}_{W'}$,$\hat{\text{Var}}[\hat{N}_W]$,$\hat{\text{Var}}[\hat{N}_{W'}]$))
    then
      $w_i \leftarrow \max\{\min\{w_i/2, w_i^*\}, 1\}$
    else if ($w^* > w$) then
      $w_i \leftarrow \max\{\min\{w_i+1, w_i^*\}, 1\}$
    end if
  end while
```

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 (4). If a change is detected, Temporal-SMURF halves its window size. Otherwise, Temporal-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 Temporal-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 Temporal-SMURF focuses primarily on capturing transitions in the aggregate and uses π-estimation to compensate for small windows in an unbiased manner.

4.5 Mobile Tag Detection

Here we present an enhancement to Temporal-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 Temporal-SMURF to use a large smoothing-window (based on Lemma 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 Temporal-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 out by Temporal-SMURF's mobile tag detector.

Temporal-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 $\frac{\Delta p_{i,t}}{epochs}$), Temporal-SMURF calculates a filter threshold as $filterThresh = \epsilon - slope \cdot w_{md}$. 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, Temporal-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 Temporal-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, Temporal-SMURF's cleaning techniques produce an accurate stream of readings (both individual tag IDs and counts) describing tags in the physical world.

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 the tests described in Section 3. We simplify a reader's detection field to derive a model of RFID readers as shown in Figure 7.

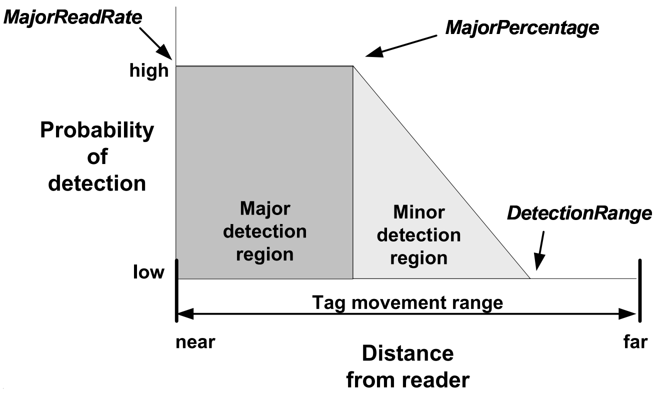


Fig. 7 Reader model and tag behavior for the RFID data generator.

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.
- *MajorReadRate*: the read rate (i.e., the probability of detection) of a tag within the major detection region. While our experimental studies show that the read rate in the minor detection region has high variance (Section 3), for the sake of simplicity we model the read rate in this region as a linear drop-off from the end of the major detection region to the end of the reader's detection range. We ran additional tests where we introduced variance into the read rate in the minor detection region; these experiments yielded similar results to those presented here.

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 3. 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 produced by a tag passing through an RFID-enabled door can be generated 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 grouped tags, 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 [61]. 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.[8] 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 synthetic RFID data traces. We manipulate the other parameters as part of these 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 3 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	0.8
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 Temporal-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 ≈ 0.2 seconds).

5.2 Per-Tag Cleaning Experiments

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.

[8] To eliminate 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.

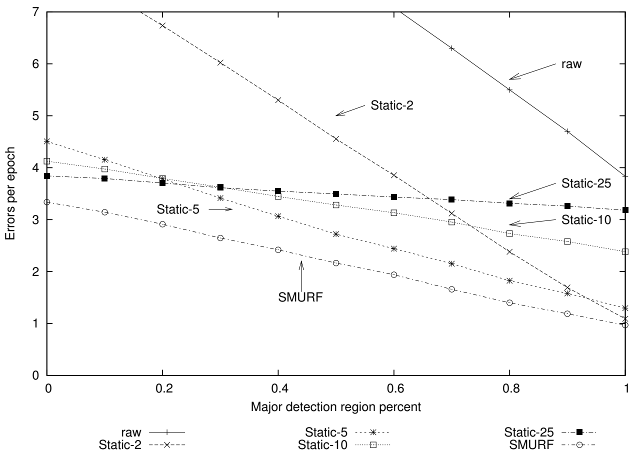


Fig. 8 Average errors per epoch as *MajorPercentage* varies from 0 to 1 with tags following *Fido* behavior.

Experiment 1: Varied Reader Reliability. In the first test, 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 8 shows the results of this experiment.

As can be seen, when the major detection region percentage is 0 (a 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 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, Temporal-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.

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 a noisy environment: the major detection region percentage is set to 0 (the left-most x-value in Figure 8). The readings produced by the reader in this scenario are particularly challenging to clean as the data are highly unreliable and the tag sporadically moves at

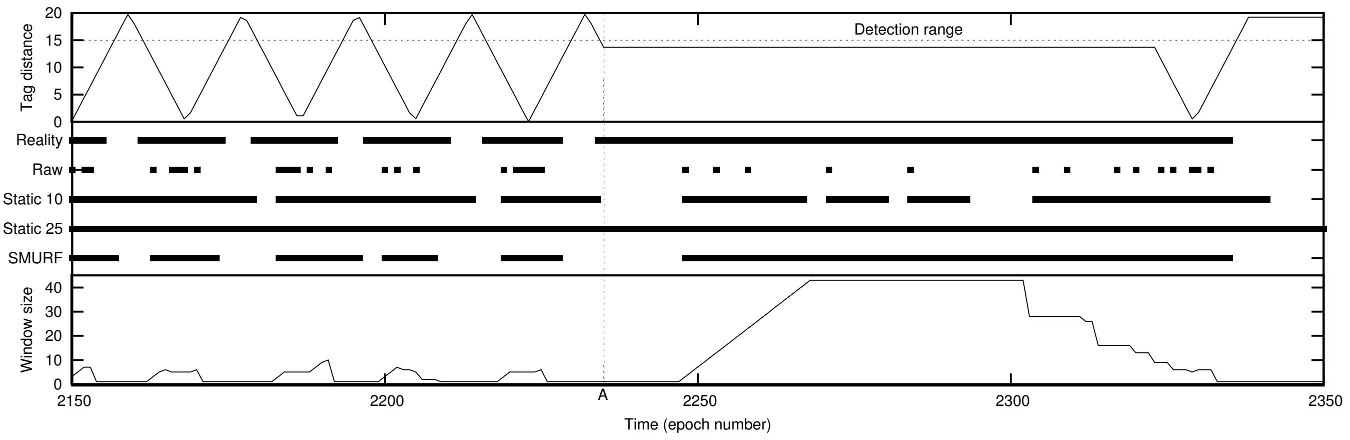


Fig. 9 A 200-epoch trace of different mechanisms cleaning the readings from a single tag moving with *Fido* behavior.

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 9 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 8), and the output of Temporal-SMURF. The bottom subsection shows Temporal-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. 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.

Temporal-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, Temporal-SMURF grows its window in reaction to the unreliable readings it receives during this period. Thus, Temporal-SMURF accurately reports the tag as present despite the severe lack of readings.[9]

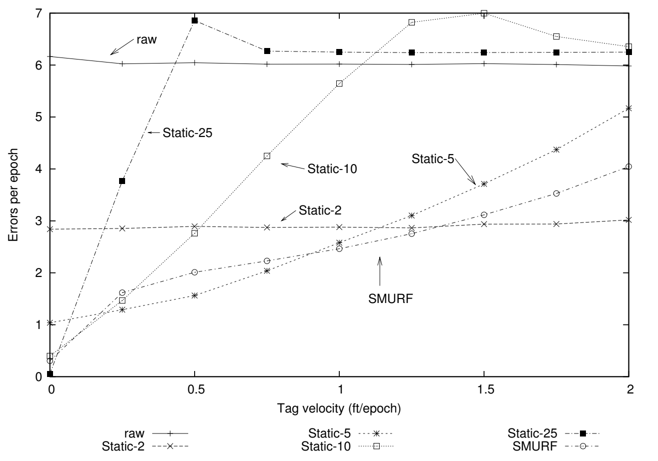
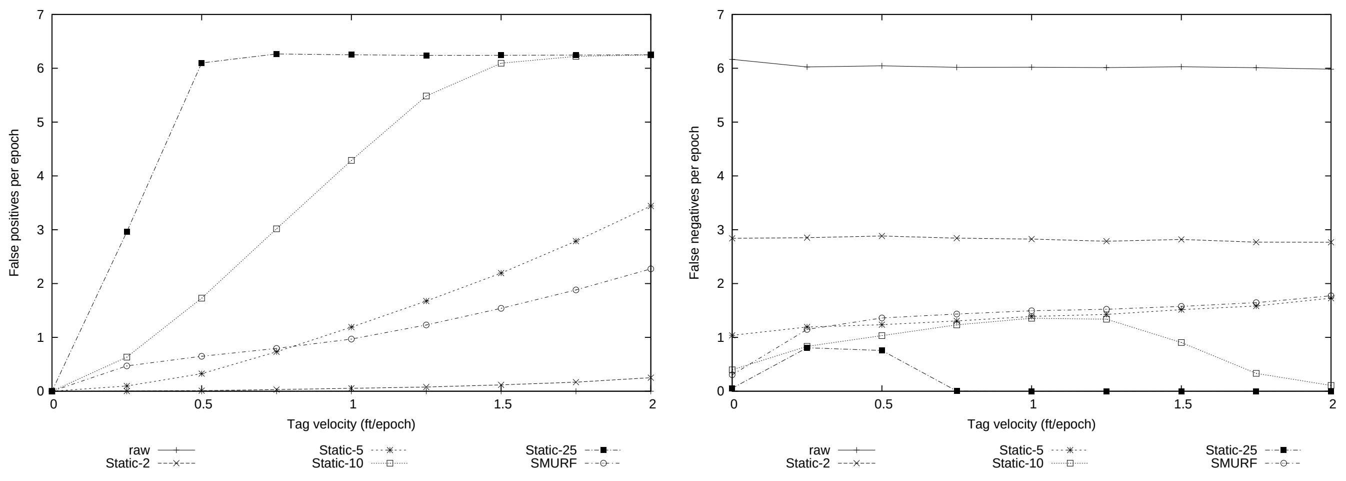


Fig. 10 Average errors per epoch as tag velocities vary from 0 to 2 feet/epoch following *Pallet* behavior.

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 10 shows the results of this experiment. Additionally, we measure separately the average false positives and average false negatives produced per epoch by each scheme, shown in Figure 11.

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 win-

[9] 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.



(a)Average false positives per epoch. (b)Average false negatives per epoch.

Fig. 11 Average errors per epoch as tag velocities vary from 0 to 2 feet/epoch following *Pallet* behavior, separated into false positives and false negatives.

dows drop at higher velocities is because at that point they continuously report all tags as present. Thus, while they produce a large number of false positives, they produce few or 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.

Temporal-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, Temporal-SMURF is able to generally track the best static window. At low velocities, Temporal-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, Temporal-SMURF's binomial sampling scheme occasionally sets its window too large: it produces roughly twice as many false positives as *Static-5* at lower velocities. As the tags speed up, however, mobile tag detection filters readings from tags that are exiting and thus reduces the false positives. As can be seen in Figure 11, from tag velocities of 1 to 1.25 feet/epoch both Temporal-SMURF and *Static-5* show similar increases in false negatives, but Temporal-SMURF produces only $2/3^{rds}$ the false positives as *Static-5*.

At the highest velocities, *Static-2* performs better than Temporal-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, $MajorPercentage = 0.7$), Temporal-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 at Temporal-SMURF's limit; transition detection and mobile tag detection prevent it from breaking down completely.

Experiment 3: δ as a Declarative Parameter. While the primary contribution of Temporal-SMURF is the removal of the imperative window size parameter from RFID data cleaning, Temporal-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 analyze how the value for this parameter affects Temporal-SMURF cleaning, we run Temporal-SMURF with different values for δ. For this experiment, we set $MajorPercentage$ to 0.7 and move tags with *Fido* behavior. Given that the dependence on δ of the binomial sampling approach used by Temporal-SMURF is weak (logarithmic), using a small δ is common practice. Thus, we vary δ between 0.1 (90% completeness) to 0.01 (99% completeness). The results are shown in Figure 12. We ran this experiment with multiple tag and reader characteristics; all experiments produced similar results.

First of all, notice that the value of δ has very little impact on the overall number of errors Temporal-SMURF produces. This means that an application that does not tune δ will not be adversely affected by choosing a standard value (e.g., 0.05).

Second, δ is a declarative parameter that allows an application to express *what* data it wants; Temporal-

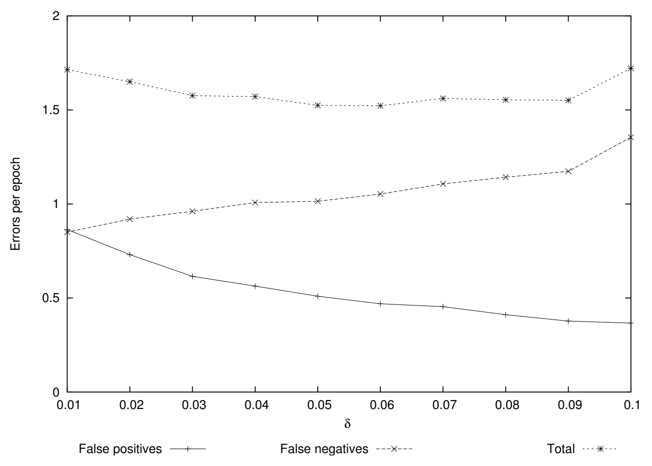


Fig. 12 Errors per epoch using different values of δ.

SMURF then determines *how* to produce that data. If an application, such as an RFID-enabled doorway, desires responsiveness (low false positives), it can set δ closer to 0.1, in which case Temporal-SMURF sets its window smaller. Conversely, an application (e.g., shelf-monitoring) that desires completeness (less false negatives) can set δ closer to 0.01 causing Temporal-SMURF to grow its window larger to meet the completeness requirement. In either case, the value of δ has little impact on overall error.

Experiment 4: Accuracy of Uncertainty Estimates. Here we measure the accuracy of Temporal-SMURF's uncertainty estimation. For this experiment, we use the same setup as in Experiment 2: we set $MajorPercentage$ at 0.7 and move tags with *Pallet* behavior at varying speeds. At each tag velocity, we measure the *accuracy* of the uncertainty estimate. Accuracy measures how close the estimate is to reality, where reality is 1 if the tag is in the reader's detection field for the given epoch, 0 if it is not. If we denote a reading's uncertainty estimate as $conf$, then for correct readings the accuracy is defined as $conf$, and for incorrect readings the accuracy is defined as $(1 - conf)$.

For Temporal-SMURF's estimation strategy, we assign an uncertainty estimate based on the Chernoff bound as described in Section 4. We denote this strategy as *Chernoff.* We compare the accuracy of Temporal-SMURF's uncertainty estimation to three other strategies. The first comparison strategy, *Percent*, assigns a confidence based on the percent of positive readings in a window: $conf = \frac{|S_i|}{w_i}$. Strategy *Average* simply assigns a confidence using the average of the detection probabilities: $conf = p_i^{avg}$. The third algorithm we compare, *Unit*, assigns unit uncertainty estimates to every output reading: $conf = 1$.

The results are shown in Table 3, reported as the average accuracy of the uncertainty estimates across all environments. Strategies *Average* and *Percent* perform poorly as they tend to assign quality estimates that are too low. Both *Chernoff* and *Unit* perform well. *Unit* performs well due to the fact that Temporal-SMURF is very successful at producing clean data. In a sense, the goal of Temporal-SMURF is to produce readings of confidence 1, so it is no surprise that *Unit* performs well. *Chernoff*, on the other hand, achieves the highest accuracy by assigning high confidence to true positive readings and lower confidence to false positive readings.

Strategy	Average accuracy
Percent	0.73 (0.04)
Average	0.59 (0.02)
Unit	0.94 (0.03)
Chernoff	0.95 (0.01)

Table 3 Average accuracy of different mechanisms for estimating uncertainty across a range of tag velocities. The standard deviation is in parenthesis.

Experiment 5: Experiences with Real RFID Data. The previous experiments were based on a generator that created RFID data based on a simple model. Real-world RFID data does not 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 Temporal-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 3 using an Alien reader [4] and a single Alien "I2" [5] 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, has 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; Temporal-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 Temporal-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 locations 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%.

Finally, these 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, Temporal-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, Temporal-SMURF performs the best: for instance, in the controlled environment, Temporal-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*.

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

5.3 Multi-Tag Aggregate Cleaning Experiments

As stated in Section 4.4, 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 Temporal-SMURF processing, we show two versions, as outlined in Section 4.4: Temporal-SMURF using per-tag cleaning with summation (σ-SMURF) and Temporal-SMURF using π-estimators (π-SMURF).

As π-SMURF cannot produce individual tag readings, we change our evaluation metric to root-mean-square error (RMS error) of the count of reported tags compared to reality.

Experiment 6: 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.

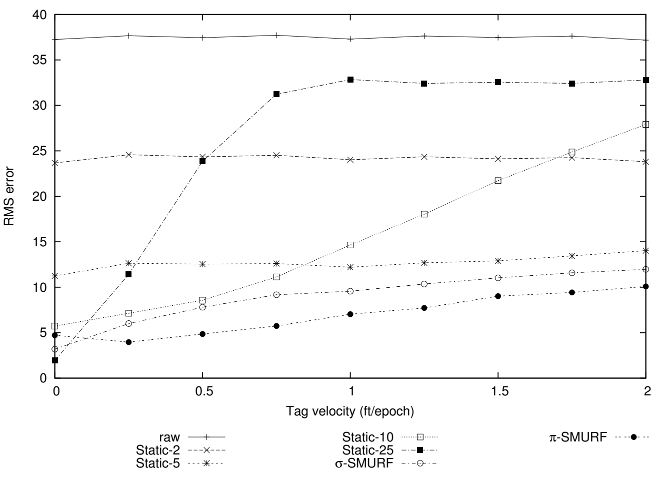
To determine the count accuracy of different schemes as the tag velocity increases, we run a similar test to Experiment 2. We set $MajorPercentage$ at 0.25 and vary the tag velocity between 0 and 2 feet/epoch using *Pallet* behavior. We show the results in Figure 13(a).

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: on expectation, π-SMURF provides accurate estimates.

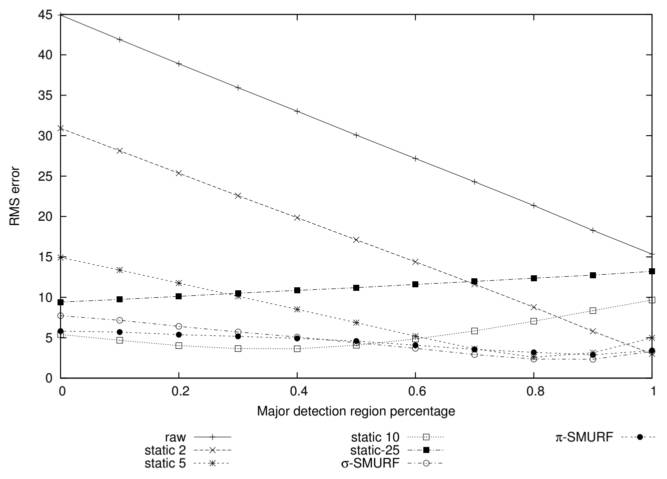
To determine how each scheme performs as the level of reliability changes, similar to Experiment 1, we move tags using *Fido* behavior and vary the major detection region percentage. At each value of $MajorPercentage$, we measure the error of each scheme. Here, both σ-SMURF and π-SMURF are competitive with the best static window.

The results are shown in Figure 13(b). Across a range of environments, both σ-SMURF and π-SMURF are competitive with the best static window. Note that this case represents the worst-case scenario for π-SMURF. As tags are moving independently and at random, on average there are an equal number of tags exiting and entering the detection field at each epoch. Since π-SMURF only looks at the count of tags, it is unable to recognize these changes. Despite this limitation, it remains close to the best scheme.

Experiment 7: Accuracy of Uncertainty Estimates. We also verify the accuracy of Temporal-SMURF's uncertainty estimates in the multi-tag case. For these tests, we fix the confidence bound at 95% and confirm that the true count of tags falls within the range supplied by Temporal-SMURF's confidence interval. For this uncertainty estimate, we use π-SMURF for the count estimate and derive a confidence interval based on the Chebyshev bound as described in Equation 5. We use the same setup as in the second half of Experiment 6: we move tags using *Fido* behavior and vary the major detection region percentage. For each value of $MajorPercentage$, we measure the *hit rate*, or percentage of epochs for which the true count of tags is within Temporal-SMURF's confidence interval.



(a)RMS error of each scheme as the tag velocity increases.



(b)RMS error of each scheme as the reader reliability increases.

Fig. 13 The RMS error of different cleaning schemes counting 100 tags.

Across most of the range of $MajorPercentage$, the true count of tags is within the confidence interval over 95% of the time; when $MajorPercentage$ is between 0 and 0.9, the average hit rate is 0.97. When the reader is highly reliable ($MajorPercentage > 0.9$), however, the hit rate suffers: when $MajorPercentage = 1$, the hit rate falls to 0.28. This low accuracy is due to the fact that the tags are mobile in a very reliable environment: within the smoothing window the p_i^{avg} is high (> 0.8), yielding a low variance for the estimator and consequently a tight confidence interval; at the same time, the tags are highly mobile, so some tags included in the estimate have left the detection range causing a less accurate count estimate. In general, Temporal-SMURF's uncertainty estimates will be less accurate when the reader is very reliable and tags are moving quickly. Since it is highly unlikely for a reader to have a $MajorPercentage$ greater than 0.9, these poor uncertainty estimates are likely not a problem in practice. Note that Temporal-SMURF's uncertainty estimates remain accurate when the reader is highly reliable, but the tags not mobile; when $MajorPercentage = 1$ and the tags are not mobile, the hit rate rises to 0.99.

Experiment 8: 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 14):

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 14. 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, Temporal-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 Temporal-SMURF techniques adjust their window sizes to balance unreliability and tag movement to outperform all static window schemes.

Finally, when the tags are moving very fast in a controlled environment, π-SMURF does particularly well as

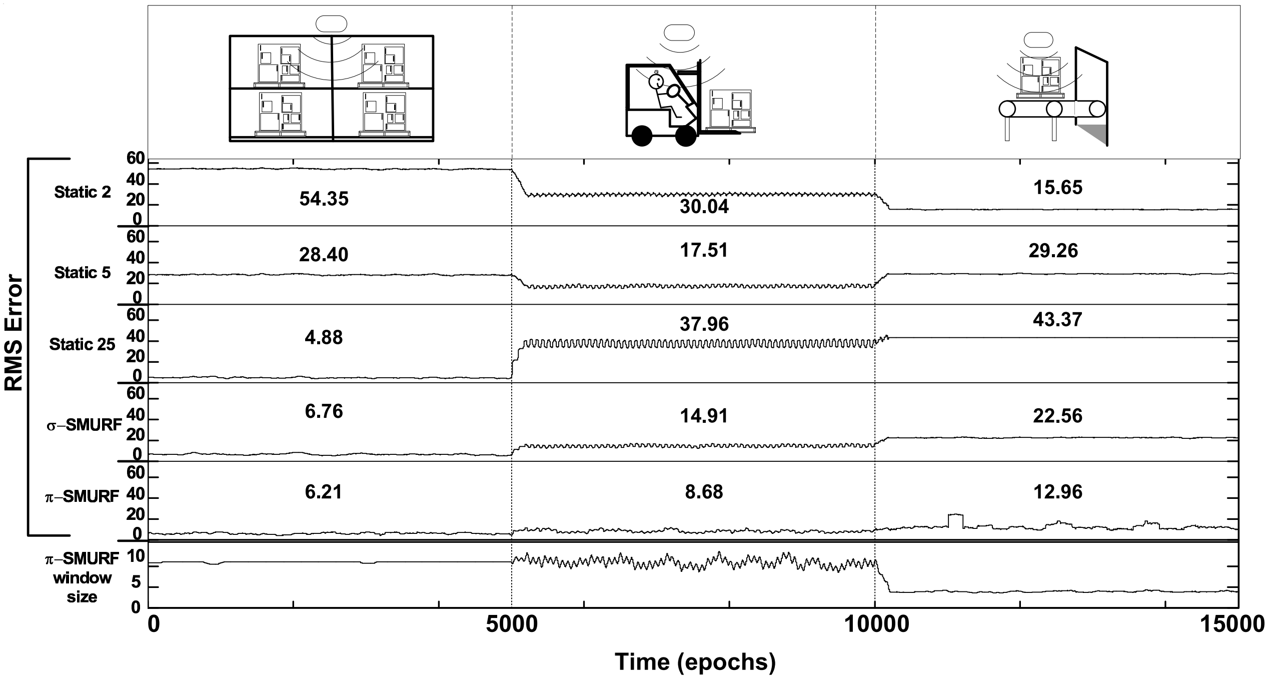


Fig. 14 A simulated pallet moving through three phases in a warehouse: shelf, forklift, and conveyor belt.

it drastically reduces its window size in reaction to the tags' movement while using π-estimators to avoid undercounting 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 Temporal-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 Providing MDI for RFID-based Applications

In the previous sections, we detailed how Temporal-SMURF effectively dealt with RFID's unreliable and frequently changing nature by adaptively cleaning the data streams. In this section, we give an overview on how MDI-SMURF handles the other two challenges associated with RFID data: RFID's spatially mismatched and low-level nature.

Spatial Mismatch. There is typically a spatial mismatch between readers' detection fields and application-level spatial units. That is, the area in which a reader is able to detect tags is usually not the same as the unit of space in which the application is interested. Furthermore, most RFID deployments contain multiple readers placed in close proximity to ensure full coverage of the area of interest and ensure completeness.

These two factors lead to overlapping detection fields, causing potential duplicate readings. Duplicates may be from readers in the same area (e.g., two readers in the same room of a digital home) and thus *reinforce* each other; in such a case, the middleware should boost the confidence of these readings. In other cases, the readings may conflict with one another (e.g., two readers on adjacent shelves in a retail scenario) and must be *arbitrated* [25,34,33]

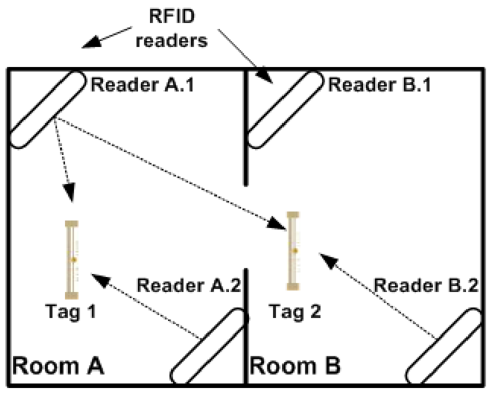


Fig. 15 Spatial issues in RFID deployments. Readings for tag 1 reported by readers A.1 and A.2 reinforce each other, while readings for tag 2 from readers A.1 and B.2 must be arbitrated.

Figure 15 illustrates these issues. Here, a digital home application is monitoring room occupancy for two rooms using four RFID readers (two in each room). Reinforcement is shown by the readings for tag 1 from readers A.1 and A.2. Arbitration is necessary for the readings for tag 2 from readers A.1 and B.2.

The opportunities of reinforcement and the challenges of arbitration cause problems for sensor-based applications. In the case of reinforcement, an application has to understand the effect on overall confidence, and perhaps collapse several readings into a single reading with appropriately-adjusted (higher) confidence. For arbitration, the application is presented with a tag that is reported in two or more locations and must somehow decide on an appropriate spatial location the tag. Dealing with both of these issues involves detailed knowledge of the detection fields of the readers involved and how these fields change over time, further complicating the processing of RFID readings.

Here, we briefly outline some ideas on how to apply our statistical-modeling techniques to handle such issues in the design of Spatial-SMURF, a *spatial* RFID filter designed to enable applications to effectively cope with reinforcement and arbitration in RFID readings.

In a nutshell, Spatial-SMURF approaches RFID reinforcement and arbitration issues using a *spatial window* abstraction. A spatial window is a grouping of readers across space that provides a multi-resolution probabilistic statement about a tag's actual location The output of Spatial-SMURF's spatial windowing is a combination of intersections and unions of reader detection fields, with attached uncertainty estimates indicating the probability of the tag being located inside the corresponding spatial window. Applications can map these spatial windows to application-level spatial units; as we discuss below, MDI-SMURF incorporates a module that enables each deployment to specify the appropriate spatial mappings.

The abstract goal of Spatial-SMURF, then, is to determine, for each tag, appropriate groupings of reader detection fields as well as corresponding uncertainty indicators (i.e., existence probabilities) for the tag in each grouping. This is obviously a very complex problem, and a full-fledged solution is beyond the scope of this paper; here, we simply outline a brief example showing how some of the above ideas can be applied in the context of Spatial-SMURF.

Example 1 Consider a simple scenario with two RFID readers A and B and let $r(A)$ and $r(B)$ denote their (possibly overlapping) detection fields. The output of each reader is fed into a Temporal-SMURF module that assigns an uncertainty estimate for the existence of any tag i in its detection field based on the sampling bounds outlined earlier in this paper; that is, we can estimate the probabilities $\rho_{r(A)} = \mathbf{Pr}[i \in r(A)]$ and $\rho_{r(B)} = \mathbf{Pr}[i \in r(B)]$.

Since $r(A)$ and $r(B)$ can overlap, in order to provide a multi-resolution spatial window to applications, Spatial-SMURF can also estimate the uncertainty indicators for other spatial regions of interest, such as $r(A) \cap r(B)$ and $r(A) \cup r(B)$. To compute $\rho_{r(A)\cap r(B)}$, the "naive" approach of assuming independence of readers (i.e., setting $\rho_{r(A)\cap r(B)} = \rho_{r(A)} \cdot \rho_{r(B)}$) is incorrect if the detection fields overlap. Instead, Spatial-SMURF can estimate the uncertainty of $r(A) \cap r(B)$ (i.e., the probability that tag i lies in the overlap region of readers A and B) using the formula $\rho_{r(A)\cap r(B)} = \mathbf{Pr}[i \in r(A) | i \in r(B)] \cdot \rho_{r(B)}$, where the conditional probability on the right-hand side of the equation can be estimated using a variety of techniques such as using the percentage of spatial overlap across detection ranges together with some spatial uniformity assumption, or from historical overlap data collected for the two readers. Similarly, Spatial-SMURF can estimate $\rho_{r(A)\cup r(B)} = \rho_{r(A)} + \rho_{r(B)} - \rho_{r(A)\cap r(B)}$.

Higher-level applications are then provided with a multi-resolution spatial-window probabilistic statement of the form: $\{r(A)[\rho_{r(A)}],\ r(B)[\rho_{r(B)}],\ r(A) \cap r(B)[\rho_{r(A)\cap r(B)}],\ r(A) \cup r(B)[\rho_{r(A)\cup r(B)}]\}$, and can determine the appropriate level of spatial localization for the tag based on the observed confidence levels. □

We leave a detailed study and evaluation of Spatial-SMURF to future work.

Semantic Gap. There is a semantic gap between the low-level data RFID readers provide and the needs of applications. The output of most RFID middleware is a stream of tag IDs and the reader at which they were read. Applications, on the other hand, typically want to know about high-level concepts such as products, people, rooms, and shelves.

The final stage in the MDI-SMURF pipeline is a *virtualize* module [33,34] that translates the smoothed and spatially processed streams from RFID-specific readings into MDI readings. Basically, the main goal of this module is to convert the data into a stream with the following schema: `(objectID[uncertainty], location[uncertainty], time)`. Producing the objectID involves a relatively simple mapping from tag ID to an application-specific object ID, such as the approach proposed for ONS [45]. For the location, Spatial-SMURF's spatial windows need to be mapped to an application-level spatial unit as mentioned above. This module can incorporate this logic internally so as to shield applications from such complexity. These mappings can be defined in a declarative manner to ease configuration [34].

Hiding these mappings in the middleware rather than exposing them to the application provides the final step for supporting MDI: all RFID-specific information is hidden by MDI-SMURF. Thus, tag IDs in a deployment can change (e.g., the deployment can switch from pallet-level to item-level tagging) or the actual readers may change; any applications using MDI-SMURF can be oblivious to such changes.

By successively correcting all challenges associated with the physical-digital divide facing RFID-based applications, MDI-SMURF provides Metaphysical Data Independence for a wide array of emerging applications. As a result, RFID applications are less complex and easier to manage and maintain.

7 Related Work

Metaphysical Data Independence provides a powerful paradigm under which to organize and unify much of the research in sensor data processing to date. In this section, we survey work related to sensor data processing in the context of MDI.

We organize this section from bottom (physical world) to top (digital world): data acquisition, data cleaning, temporal and spatial processing, virtualization, and application infrastructures.

Data Acquisition. The first task, obviously, in supporting MDI is to acquire data from the sensor devices.

Systems such as TinyDB [39], Cougar [8], and DSN [14] significantly reduce the complexity of accessing data from raw devices. Nonetheless, they produce low-level, dirty streams of readings that must be processed substantially before being sent to applications.

At a level above these systems is BBQ [18]. Many sensor data are highly correlated, both in time and space. This fact can be exploited by using model-based sensing, which only uses the physical devices when its internal model is unable to answer a query with sufficient confidence. While BBQ explores the idea of using probabilistic models for sensor measurements, our work is the first to apply statistical techniques for adaptive RFID data processing. Furthermore, this scheme relies on learning and maintaining many fairly heavyweight multi-dimensional Gaussian models; our techniques rely on simple, non-parametric sampling estimators. Exploring interactions between the two approaches is an interesting area for future work.

Sensor Data Cleaning. Much work has focused on techniques for low-level cleaning of sensor data [21,43]. Other work, on the other hand, has addressed high-level cleaning using data mining-based techniques [50]. Similarly, StreamClean takes a high-level approach to cleaning RFID data using global integrity constraints [36]. Such techniques straightforwardly fit within a sensor infrastructure providing MDI.

More specifically related to Temporal-SMURF, several projects have explored simple techniques to clean RFID data based on fixed-window smoothing. In one paper, the authors identify the trade-off between smoothing the data and capturing the temporal variation but provide no real solutions [24]. In previous work, we recognize the need to clean RFID data and use an approach based on declarative continuous queries [33,34]. We show smoothed RFID data using different sized windows, but do not address how to choose the best size.

Many commercial RFID middleware solutions contain configurable filters to process data produced by RFID readers [9,28,40,62]. Many of these platforms explicitly incorporate data smoothing as a solution to RFID unreliability. None of these systems, however, provide any guidance for how to configure the smoothing filters, nor do they address the granularity mismatch and low-level semantics of RFID data. Furthermore, they do not estimate the uncertainty of the readings they produce.

Adaptive filtering has been studied in digital signal processing in wide-ranging contexts such as image analysis and speech processing [47]. Especially applicable are nonlinear digital filters, which are designed to capture transitions in the signal. For instance, AWED [47] adapts the size of a smoothing window for cleaning noisy images using a multi-phase approach involving smoothing and edge detection that inspired the basic Temporal-SMURF design.

Related to data cleaning is uncertainty estimation for cleaned sensor data. In [53], we proposed the shadow quality-tracking pipeline used in the MDI-SMURF architecture. The uncertainty estimates presented here improve upon the techniques presented in that paper.

Temporal and Spatial Processing. Removing the dependence on the spatial and temporal specifics of sensing devices has been the focus of recent work in the context of both wireless sensor networks and RFID data.

MauveDB [19] provides temporal and spatial independence from the underlying sensors through the use of model-based views that are updated as the sensor data changes. Similarly, ESP [33,34] proposes *temporal* and *spatial granules* to capture the application-level notion of time and space. While the focus of Distributed Regression [27] is efficient in-network processing of wireless sensor data to extract more information than traditional aggregation schemes, by fitting a regression function to the sensor data it produces spatial independence. Finally, localization [29,30] using multiple sensor readings provides a means by which the data can be fused into high-level spatial information that is independent of the underlying devices.

While all of the above techniques can be used to remove the dependence on the spatial and temporal sensing granularity of physical devices, they do not fully address all of the challenges associated with the physical-digital divide.

Virtualization. Many projects have looked at providing an abstraction layer between the application and devices in the physical world.

Closest to our work is Semantic Streams [64], which proposes a programming framework using inference modules for deriving various characteristics of the physical world declaratively. Similarly, the Context Toolkit [20] advocates an architectural approach to hiding the details of sensor devices. These projects, however, do not fully address many of the issues with crossing the physical-digital divide, such as data cleaning, temporal and spatial processing, and variability.

SensorML [46] and ALE [6] define high-level interfaces to sensor and RFID devices, respectively. They ease the development of applications by providing a standard interface with which to interact with their respective type of device, but they are device-specific and thus do not completely separate the devices and the application with a layer of independence.

The *Virtual Device* architecture [32] presented as part of the HiFi [25,51] project was designed as a general architecture for supporting MDI and influenced much of the work presented here; MDI-SMURF can be thought of as a specific instance of a Virtual Device for RFID deployments.

Application Infrastructures. With the rise in popularity of sensing devices, there are many systems in the digital world designed to understand or make use of sensor data.

Trio [44], MYSTIQ[10], and HeisenData [26] are systems designed to manage uncertain or probabilistic data. These systems exist in the digital world, but understand the uncertain nature of the physical world; thus, they provide applications built to use data from the real world the means of understanding the uncertainty inherent in the physical world. Further, sensor infrastructures designed to provide MDI could benefit from the techniques for processing and tracking uncertain data being developed by these projects.

Complex event processing infrastructures developed as part of the HiFi [51] and SASE [65] projects provide languages and execution environments designed explicitly for applications that utilize sensor data. Such systems and the applications using them would greatly benefit from Metaphysical Data Independence: applications could be built using event languages on top of an MDI-based interface without being concerned with any of the issues in dealing with sensor devices.

8 Conclusions

While sensor-based applications hold much promise to revolutionize the way in which we interact with the physical world, current sensor-based applications are brittle, ad-hoc, and hard to deploy and maintain due to their dependence on the devices over which they are built.

At the core of this issue is the physical-digital divide: sensor data is unreliable, mismatched with application needs in time and space, semantically low-level, and frequently changing. Incorporating logic in the application to bridge this divide cause applications to become complex and hard to manage and scale.

Toward this end, Metaphysical Data Independence draws a separation of concerns between applications and the devices on which they are built. Through a high-level object-based abstraction interface, applications interact with a reconstruction of the physical world in the digital world as if the physical-digital divide did not exist.

As a concrete example of how to create this reconstruction, MDI-SMURF is an RFID middleware system that addresses all issues associated with RFID data and provides an MDI-based interface to applications that use RFID data.

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

In order to realize the promise of sensor-based applications, it is critical that these applications be separated from the issues associated with crossing the physical-digital divide by Metaphysical Data Independence. As a result of the data independence provided by sensor infrastructures such as MDI-SMURF, sensor-based applications are substantially simpler and more robust.

9 Acknowledgments

The authors would like to thank the following people for their contributions. Gustavo Alonso, Wei Hong, and Jennifer Widom contributed to the original MDI vision and shaped much of the resulting work. Nathan Burkhart, Eugene Wu, and Kai Xia assisted in the sensor experiments. Matt Denny, Ryan Huebsch, David Liu, Alexandra Meliou, and other members of the Berkeley Database Group provided feedback on early drafts. Ryan Aipperspach, Kurt Brown, Joe Hellerstein, Eirinaios Michelakis, and Daisy Wang of the Data Furnace/HeisenData project provided additional insight from the point of view of data quality and pervasive applications. Anish Das Sarma helped with quality tracking for sensor data. Finally, the anonymous reviewers provided many useful comments. This work was funded in part by NSF under ITR grants IIS-0086057 and SI-0122599, and by research funds from Intel and the UC MICRO program.

References

1. Alien Technology. Nanoscanner Reader User Guide
2. Merriam-Webster Online Dictionary. http://m-w.com
3. Abowd, G.D., Mynatt, E.D.: Charting past, present, and future research in ubiquitous computing. ACM Transactions on Computer-Human Interaction **7**(1), 29–58 (2000)
4. Alien ALR-9780 915 MHz RFID Reader. http://www.alientechnology.com/products/rfid-readers/alr9780.php
5. Alien RFID tags. http://www.alientechnology.com/products/rfid-tags
6. Application Level Event (ALE) Specification Version 1.0. Http://www.epcglobalinc.org/standards_technology/EPCglobal_ApplicationALE_Specification_v112-2005.pdf
7. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. The VLDB Journal **15**(2), 121–142 (2006)
8. Bonnet, P., Gehrke, J., Seshadri, P.: Towards Sensor Database Systems. In: Proc. Mobile Data Management, *Lecture Notes in Computer Science*, vol. 1987. Springer, Hong Kong (2001)
9. Bornhövd, C., Lin, T., Haller, S., Schaper, J.: Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure. In: VLDB (2004)

10. Boulos, J., Dalvi, N., Mandhani, B., Mathur, S., Re, C., Suciu, D.: MYSTIQ: a system for finding more answers by using probabilities. In: SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data (2005)
11. Buonadonna, P., Gay, D., Hellerstein, J.M., Hong, W., Madden, S.: TASK: Sensor Network in a Box. In: EWSN (2005)
12. Chen, J., Kam, A.H., Zhang, J., Liu, N., Shue, L.: Bathroom Activity Monitoring Based on Sound. In: Pervasive (2005)
13. Chiu, D.M., Jain, R.: Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. Comput. Netw. ISDN Syst. **17**(1) (1989)
14. Chu, D.C., Popa, L., Tavakoli, A., Hellerstein, J.M., Levis, P., Shenker, S., Stoica, I.: The design and implementation of a declarative sensor network system. Tech. Rep. UCB/EECS-2006-132, EECS Department, University of California, Berkeley (2006). URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-132.html
15. Cochran, W.G.: "Sampling Techniques". John Wiley & Sons (1977)
16. Daniel Dobkin and Steven Weigand: Tags vs. the World: HF and UHF Tags in non-ideal environments. WCA RFID SIG (2005)
17. Deavours, D.D.: Performance Analysis of Commercially Available UHF RFID Tags Based on EPCglobal's Class 0 and Class 1 Specifications. RFID Alliance Lab (2004)
18. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J.M., Hong, W.: Model-Driven Data Acquisition in Sensor Networks. In: VLDB Conference (2004)
19. Deshpande, A., Madden, S.: MauveDB: supporting model-based user views in database systems. In: SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data (2006)
20. Dey, A.K.: Providing architectural support for building context-aware applications. Ph.D. thesis, Georgia Institute of Technology (2000)
21. Elnahrawy, E., Nath, B.: Cleaning and querying noisy sensors. In: WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications (2003)
22. EPC Tag Data Specification Version 1.1. http://www.epcglobalinc.org/standards_technology/EPCTagDataSpecification11rev124.pdf
23. EPCGlobal, Inc. http://www.epcglobalinc.org/
24. Fishkin, K.P., Jiang, B., Philipose, M., Roy, S.: I Sense a Disturbance in the Force: Unobtrusive Detection of Interactions with RFID-tagged Objects. In: Ubicomp (2004)
25. Franklin, M.J., Jeffery, S.R., Krishnamurthy, S., Reiss, F., Rizvi, S., Wu, E., Cooper, O., Edakkunni, A., Hong, W.: Design Considerations for High Fan-In Systems: The HiFi Approach. In: CIDR (2005)
26. Garofalakis, M.N., Brown, K.P., Franklin, M.J., Hellerstein, J.M., Wang, D.Z., Michelakis, E., Tancau, L., Wu, E., Jeffery, S.R., Aipperspach, R.: Probabilistic Data Management for Pervasive Computing: The Data Furnace Project. IEEE Data Eng. Bull. **29**(1), 57–63 (2006)
27. Guestrin, C., Bodi, P., Thibau, R., Paski, M., Madden, S.: Distributed regression: an efficient framework for modeling sensor network data. In: IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks (2004)
28. Gupta, A., Srivastava, M.: Developing Auto-ID Solutions using Sun Java System RFID Software (2004). DOI http://java.sun.com/developer/technicalArticles/Ecommerce/rfid/sjsrfid/RFID.html
29. Hahnel, D., Burgard, W., Fox, D., Fishkin, K., Philipose, M.: Mapping and Localization with RFID Technology. In: ICRA (2004)
30. Hightower, J., Brumitt, B., Borriello, G.: The Location Stack: A Layered Model for Location in Ubiquitous Computing. In: Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002), pp. 22–28. IEEE Computer Society Press, Callicoon, NY (2002)
31. Intel Lab Data. http://berkeley.intel-research.net/labdata/
32. Jeffery, S.R., Alonso, G., Franklin, M.J., Hong, W., Widom, J.: Virtual Devices: An Extensible Architecture for Bridging the Physical-Digital Divide. Tech. Rep. UCB-CS-05-1375, UC Berkeley CS Division (2005)
33. Jeffery, S.R., Alonso, G., Franklin, M.J., Hong, W., Widom, J.: A Pipelined Framework for Online Cleaning of Sensor Data Streams. In: ICDE (2006)
34. Jeffery, S.R., Alonso, G., Franklin, M.J., Hong, W., Widom, J.: Declarative Support for Sensor Data Cleaning. In: Pervasive (2006)
35. Jeffery, S.R., Garofalakis, M., Franklin, M.J.: Adaptive Cleaning for RFID Data Streams. In: VLDB'2006: Proceedings of the 32nd international conference on Very large data bases, pp. 163–174 (2006)
36. Khoussainova, N., Balazinska, M., Suciu, D.: Towards correcting input data errors probabilistically using integrity constraints. In: MobiDE '06: Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access (2006)
37. Krishnamurthy, S.: Shared query processing in data streaming systems. Ph.D. thesis, University of California, Berkeley (2006)
38. Laurie Sullivan: RFID Implementation Challenges Persist, All This Time Later. Information Week (2005)
39. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong:, W.: The Design of an Acquisitional Query Processor For Sensor Networks. In: SIGMOD (2003)
40. Manage Data Successfully with RFID Anywhere Edge Processing. http://www.ianywhere.com/developer/rfid_anywhere/rfidanywhere_edgeprocessing.pdf
41. Martonosi, M.: Embedded systems in the wild: Zebranet software, hardware, and deployment experiences. In: LCTES (2006)
42. Motwani, R., Raghavan, P.: "Randomized Algorithms". Cambridge (1995)
43. Mukhopadhyay, S., Panigrahi, D., Dey, S.: Data aware, Low cost Error correction for Wireless Sensor Networks. In: WCNC (2004)
44. Mutsuzaki, M., Theobald, M., de Keijzer, A., Widom, J., Agrawal, P., Benjelloun, O., Sarma, A.D., Murthy, R., Sugihara, T.: Trio-One: Layering Uncertainty and Lineage on a Conventional DBMS. In: Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR '07) (2007)
45. Object Naming Service (ONS) Standard, Version 1.0. http://www.epcglobalinc.org/standards/Object_Naming_Service_ONS_Standard_Version_1.0.pdf
46. OpenGIS Sensor Model Language (SensorML) (05-086r2). http://portal.opengeospatial.org/files/?artifact_id=13879
47. Pitas, I., Venetsanopoulos, A.N.: "Nonlinear digital filters: principles and applications". Kluwer, Boston, MA (1990)
48. Plato: "Republic"
49. Qin, S.: Neural networks for intelligent sensors and control — practical issues and some solutions. In Neural Networks for Control, D. Elliott, Ed. Academic Press, 1996. Chapter 8.
50. Rao, J., Doraiswamy, S., Thakkar, H., Colby, L.S.: A deferred cleansing method for rfid data analytics. In: VLDB'2006: Proceedings of the 32nd international conference on Very large data bases (2006)

51. Rizvi, S., Jeffery, S.R., Krishnamurthy, S., Franklin, M.J., Burkhart, N., Edakkunni, A., Liang, L.: Events on the edge. In: SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data (2005)
52. Rousu, J., Elomaa, T., Aarts, R.J.: Predicting the Speed of Beer Fermentation in Laboratory and Industrial Scale. In: IWANN (2), pp. 893–901 (1999)
53. Sarma, A.D., Jeffery, S.R., Franklin, M.J., Widom, J.: Estimating Data Stream Quality for Object-Detection Applications. In: IQIS (2006)
54. Särndal, C.E., Swensson, B., Wretman, J.: "Model Assisted Survey Sampling". Springer-Verlag New York, Inc. (Springer Series in Statistics) (1992)
55. Senosrmatic Agile 2 915Hz RFID Reader. http://www.sensormatic.com/RFID/stationary/
56. SensorID Series 2 Agile Reader Query Protocol (2004)
57. Sharp, C., Schaffert, S., Woo, A., Sastry, N., Karlof, C., Sastry, S., Culler, D.: Design and Implementation of a Sensor Network System for Vehicle Tracking and Autonomous Interception (2005)
58. Smith, J.R., Sample, A.P., Powledge, P.S., Roy, S., Mamishev, A.: A Wirelessly-Powered Platform for Sensing and Computation. In: Ubicomp, pp. 495–506 (2006)
59. Sonoma Redwood Sensor Network Deployment. http://www.cs.berkeley.edu/get/sonoma/
60. Tolle, G., Polastre, J., Szewczyk, R., Culler, D.E., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., Hong, W.: A Macroscope in the Redwoods. In: SenSys, pp. 51–63 (2005)
61. UW RFID Lab http://www.uwrfidlab.org/
62. Wang, F., Liu, P.: Temporal Management of RFID Data. In: VLDB, pp. 1128–1139 (2005)
63. Want, R.: The Magic of RFID. ACM Queue **2**(7), 40–48 (2004)
64. Whitehouse, K., Zhao, F., Liu, J.: Automatic programming with semantic streams. In: SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems (2005)
65. Wu, E., Diao, Y., Rizvi, S.: High-performance Complex Event Processing Over Streams. In: SIGMOD Conference (2006)