# Estimating Data Stream Quality for Object-Detection Applications

*Anish Das Sarma*
*Shawn Ryan Jeffery*
*Michael Franklin*
*Jennifer Widom*

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2005-23
http://www.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-23.html

December 6, 2005

# Estimating Data Stream Quality for Object-Detection Applications[*]

Anish Das Sarma[1] Shawn R. Jeffery[2] Michael J. Franklin[2] Jennifer Widom[1]

[1]Stanford University
{anish,widom}@cs.stanford.edu

[2]UC Berkeley
{jeffery,franklin}@cs.berkeley.edu

## Abstract

Object-detection applications rely on streams of data gathered from sensors, RFID readers, and image recognition systems, among others. These raw data streams tend to be noisy, including both false positives (erroneous readings) and false negatives (missed readings). Techniques exist for general-purpose *cleaning* of these types of data streams, based on temporal and/or spatial correlations, as well as properties of the physical world. Cleaning is effective at improving the quality of the data, however no cleaning procedures can eliminate all errors. In this paper we identify and address the problem of *quality estimation* as object-detection data streams are cleaned. We provide techniques for estimating both *confidence* and *coverage* as streams are processed by cleaning modules. Detailed experimental results based on an RFID application demonstrate the accuracy and effectiveness of our approach.

## 1 Introduction

Continuous detection of real-world objects forms an important component of a wide variety of systems today. Applications range from military tracking and digital homes to monitoring inventory and tracking wildlife [10, 17, 18, 22, 28]. Such Object-Detection Applications (ODAs) typically collect and process streams of data from sensors, RFID (Radio Frequency Identification) readers, image recognition systems, and other real-world monitoring devices.

[*]This work was supported by the National Science Foundation under ITR grants IIS-0324431, IIS-0086057, and SI-0122599, and by research funds from Intel, Microsoft, and the UC MICRO program

The quality of the raw data streams produced by these devices is notoriously poor [6, 9, 26, 20]. Intuitively, the quality of an ODA stream measures how well the stream of data values (hereafter *readings*) matches the objects actually present in reality. The errors in raw data streams generally consist of (1) *false positives* (erroneous readings): reporting objects that are not actually present; and (2) *false negatives* (missed readings): not reporting objects that actually are present.

Typically, these raw data streams are preprocessed, or *cleaned*, to improve their quality before decisions are taken based on the data. Cleaning techniques tend to be based on temporal and/or spatial correlations, as well as other known properties of the objects and their interaction with the physical world [7, 9, 16, 29, 20, 21]. For example, since animals move continuously in space and time, we don't expect to see an abrupt jump with respect to either of these coordinates in the reported animal-tracking data stream. As another example, two readings of the same object at two different locations at the same time cannot both be correct. In cleaning tools, multiple different cleaning modules may be arranged in a pipeline to successively improve the data streams.

While cleaning of object-detection data streams can substantially improve the quality of reported readings, no technique can reliably eliminate all errors. We argue that in many cases, it is essential for an application to be made aware of the level of reliability of its input streams, so that it can make appropriate decisions. For instance, a digital home application acting on unreliable data produced by a mediocre cleaning tool may sporadically turn on and off the house's lights. With an accurate estimate of the quality of the data stream, such an application could threshold its decision-making on the quality of the data, thus "calming" the application. Our goal is to develop techniques that can be used to provide applications with useful information about the quality of the cleaned data streams.

In this paper, we identify and address the problem of *quality estimation* as ODA data streams are cleaned. We introduce a *quality tracking pipeline* that consists of a set of quality estimation modules that shadow the main data cleaning pipeline. Each module in this shadow pipeline is associated with a particular cleaning module and tracks the quality of the data as the ODA streams are cleaned. We derive online algorithms for the quality tracking modules that provide accurate measures of quality as the ODA streams flow through the cleaning process. Our modules track two measures of quality: *confidence* and *coverage* [31]. Confidence, a value associated with individual readings, tracks the estimated probability of a reading in the data stream being correct, i.e., of the reported object actually being present at the given location and time. Coverage, a value associated with a window of readings, tracks the estimated fraction of present objects that are accounted for within a window of the data stream.

Applications can make direct use of confidence or coverage. For example, in the above digital home scenario, the application may want to turn on the lights only when the estimated confidence that someone is in the room exceeds a threshold. In an animal tracking application reporting numbers of animals present in specific regions, counts derived from the data stream may be scaled up based on estimated coverage.

It is important to note the clear distinction between how good an ODA stream is (its quality), and how good its quality estimates are. For example, there could be an excellent-quality data stream (very few false positives or negatives) yet poor estimates of confidence and coverage; conversely, there could be a poor-quality data stream yet accurate confidence and coverage estimates. Our goal in this paper is not to further improve the quality of the data streams, but rather to augment existing cleaning techniques with accurate estimates of confidence and coverage. In some cases, improving and assessing quality can be interrelated: in Section 8.2, we show how our quality estimation techniques can be used to help improve quality.

There is an existing body of ongoing work on improving the quality of ODA streams [9, 13, 25, 20, 21], but we are not aware of any prior work that attempts to assess or quantify the resulting quality. In our work, we look at existing cleaning techniques and demonstrate that quality estimates can be propagated along with the data stream using simple and intuitive mathematical algorithms. We validate our approach experimentally on a synthetic RFID-based application. Results show that in this domain our techniques for several types of cleaning modules produce accurate quality estimates and remain accurate as the stream progresses through a pipeline of multiple cleaning modules.

To summarize, the main contributions of this paper are:

- Formulation of the *quality estimation* problem for cleaned data streams in Object-Detection Applications, and the development of a *quality tracking pipeline* consisting of modules that estimate data quality as ODA streams are cleaned (Section 3).
- Specific algorithmic and mathematical techniques for quality estimation as data is processed by several types of cleaning modules. We consider cleaning based on temporal and spatial smoothing in Section 4, cleaning based on object location in Section 5, and filter-based cleaning in Section 6.
- A case study of our techniques on a synthetic RFID application (Section 7), with detailed experimental results validating our approach (Section 8).

Section 2 provides preliminary material on ODA and cleaning modules. Related work is covered in Section 9, and we conclude with future work in Section 10.

## 2 Data Model and Cleaning Techniques

We consider ODAs in which the data stream can be abstracted for cleaning and quality estimation purposes to the schema `<oid,time,loc>`. `oid`'s identify the *objects* in the application (such as an animal or ship), whose presence are reported by *sensors* (such as RFID readers, environmental sensors, or image analysis). Each sensor is associated with a *logical area*, such as a room in a digital home or a shelf in a retail home scenario, in which it detects objects; `loc` identifies this logical area (and not the actual spatial coordinates) of the object. We assume `time` refers to some discrete logical *time-unit* for the reading, which could be application-specific and need

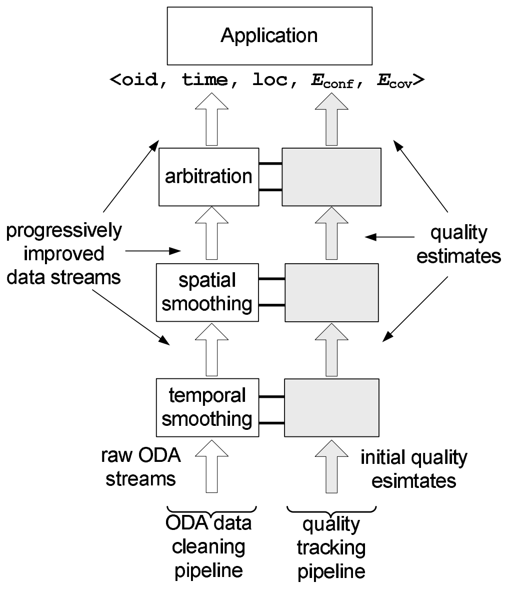

Application
<oid, time, loc, $E_{\text{conf}}$, $E_{\text{cov}}$>
arbitration
spatial smoothing
temporal smoothing
progressively improved data streams
quality estimates
raw ODA streams
initial quality esimtates
ODA data cleaning pipeline
quality tracking pipeline

Figure 1: Example cleaning pipeline including the quality tracking pipeline.

not correspond to wall-clock or system time. Since there could be multiple sensors associated with a logical area, there could be duplicate `<oid,time,loc>` values in the stream.

Cleaning occurs in one or more *modules* that can be categorized based on the following principles [20].

- Multiple readings complementing each other:
    - *Smoothing in time*: objects reported at one instant of time $t$ are likely to also be reported in an interval of time near $t$.
    - *Smoothing in space*: objects reported in one location by a sensor should also be reported by other sensors associated with the same location.
- Multiple readings contradicting each other:
    - *Arbitration:* objects cannot be present at two or more different locations at the same time instant, so such readings would be contradictory.

Typically, one or more of these cleaning modules are assembled in a pipeline to successively clean the ODA data streams, as shown in Figure 1. The figure also shows the shadow pipeline of quality estimation modules.

Given the simple schema of the data stream consisting of object, location, and time, we argue that most cleaning techniques must be based on the primary properties above. One could, of course, use specialized application knowledge—for example lions and deer are unlikely to be seen in the same logical area—and devise more sophisticated cleaning techniques. In this paper, however, we restrict ourselves to general-purpose, application-knowledge oblivious techniques.

Next we give details of the various cleaning modules considered in this paper. A graphical depiction of these modules is shown in Figure 2.

## 2.1 Temporal Smoothing

Temporal smoothing, as the name suggests, "smooths" the data stream in the time dimension. The idea is to decide whether an object $o$ was present at location $l$ and time $t$ not only based on the reading at time $t$, but also based on readings of this object in a window of size $W$ adjacent to $t$. There are many variants of temporal smoothing that vary how the window is positioned (e.g., starting, ending, or centered at $t$) and how readings within the window are processed (e.g., emitting an output reading if there is at least one reading in the window, or if the number of readings exceeds a threshold). We focus in this work on a basic temporal smoothing algorithm used in many ODA deployments [9, 16, 24, 20] that emits an output reading at time $t$ if there is at least one input reading in a window of size $W$ ending at $t$. Formally, a reading `<o,t,l>` is output after temporal smoothing if there exists a reading `<o,t',l>` in the input data stream with $(t - W) \leq t' \leq t$. Notice that temporal smoothing does not eliminate any readings in the input stream, but only adds additional readings. In our experimental section (Section 8) we show that our quality estimation techniques work for other variations of temporal smoothing algorithms.

## 2.2 Spatial Smoothing

Spatial smoothing aggregates data in the space dimension, combining readings from multiple sensors monitoring the same logical area. Here, we focus on a variant of spatial smoothing that merges multiple streams: the output of spatial smoothing contains a reading for an object if

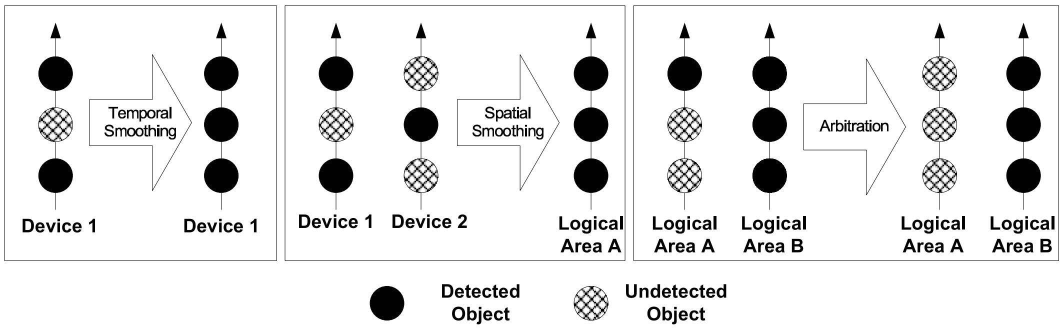


Figure 2: Typical general-purpose cleaning modules for object-detection data streams.

any one of the sensors reports it. Formally, a reading `<o,t,l>` is output after spatial smoothing if there exists a reading `<o,t,l>` in the input of at least one of the sensors in the logical area. (We assume the presence of sensor-id's if needed.)

Note that temporal and spatial smoothing when used together can go in either order, however since temporal smoothing can operate on data from an individual sensor, it may be advantageous to do it first.

## 2.3 Arbitration

Arbitration reconciles conflicting readings from sensors monitoring different logical areas. The objective is to identify readings of the form `<o,t,l1>`, `<o,t,l2>` that report the same object at multiple different locations (i.e., logical areas) at the same time. Since these conflicts could possibly be resolved better by using application-level logic, this cleaning module simply identifies such conflicting readings and marks them for the application to resolve. Another option for this cleaning module is to eliminate all but one (or a few) readings for every object-time pair based on each reading's confidence; we show how our techniques extend to this case as well.

# 3 Quality of ODA Data Streams

In this section, we define the quality of an ODA data stream as the difference between the reported data and reality.

Conceptually, the *reality stream* is a virtual stream consisting of readings at each time interval for all objects present in each of the application's logical areas. This stream represents physical reality: it is the stream that the ODA sensors would produce if the technology were perfect (i.e., if sensors reported all and only the objects present in their logical area at each time instant). At a high level, the *quality* of an ODA stream measures the deviation (in terms of false positives and negatives) of the data stream from the reality stream.

We use two components for our intuitive notion of quality —*confidence* and *coverage*—to capture how closely the data stream resembles the reality stream. Confidence accounts for objects that are not present in reality but are reported, and coverage accounts for objects that are present in reality but are not reported. We first define confidence and coverage formally, then discuss estimating these values in an online ODA cleaning pipeline.

## 3.1 Definitions of Confidence and Coverage

Consider the reality stream *R*, and an ODA data stream $D$ whose quality we want to measure after the application of zero or more cleaning modules. Given $R$ and $D$, we define: (1) True Positives, *TP*, as the set of all readings that are present in $R$ as well as $D$; (2) False Positives, *FP*, as the set of all readings that are present in $D$ but not in $R$; (3) False Negatives, *FN*, as the set of all readings present in $R$ but not in $D$. Now we define confidence and coverage as follows:

- **Confidence:** Confidence, a per-reading value, gives the probability that the reading exists in the reality stream (i.e., that the object exists in reality at the given time and location). Ideally, confidence is 1 for all readings in *TP* and 0 for all readings in *FP*. This metric is naturally extended to the entire data stream $D$, or any subset of readings: The confidence for a set of readings is the average of the confidences of the individual readings.

- **Coverage:** Coverage is a window-level value assigned to a set of readings for a given time period $T$. It gives the fraction of readings from $R$ in the time period $T$ that are present in $D$. Hence, for the entire stream, or for any time-window having associated values of *TP*, *FP* and *FN*, $Coverage = \frac{|TP|}{|TP \cup FN|}$. Note that the coverage time window may be different than the temporal smoothing time window.

## 3.2 Quality Estimation

Since the exact values for *TP*, *FP* and *FN* are not known, we cannot determine the exact confidence and coverage of a cleaned data stream. Our goal is to provide *estimates* of confidence and coverage that are close to their actual values. We denote the estimates of confidence and coverage by $E_{conf}$ and $E_{cov}$, respectively. Though the ideal confidence of each reading is either 0 or 1, $E_{conf}$ estimates the probability of a reading being correct and ranges between 0 and 1.

We augment the schema of a reading in an ODA stream with confidence and coverage. Quality estimation for each cleaning module is based on the incoming data stream of readings of schema <oid, time, loc, $E_{conf}$, $E_{cov}$>, the output data stream with the same schema, and knowledge of the cleaning technique.

Due to our modular approach, before any cleaning has been performed we need some initial estimates for $E_{conf}$ and $E_{cov}$. Confidence and coverage for a raw data stream of readings usually depends on several properties. These properties are heavily dependent on the sensors, objects, and the particular ODA deployment, so there are no general techniques for initial quality estimation of an ODA stream. There are, however, a wide variety of techniques that are applicable. For instance, work on building models of sensor error characteristics and detection fields [12, 14] can be used to infer initial estimates. In Section 7, we show how initial quality estimation is done for a specific RFID-based application.

## 3.3 Quality Tracking Pipeline

Given the pipelined, modular design of most ODA cleaning mechanisms, we develop a parallel *quality tracking pipeline* for ODA data streams. Each module in this pipeline is associated with a particular cleaning module (e.g., temporal smoothing, spatial smoothing, arbitration) and propagates the confidence and coverage of the stream alongside the data each cleaning module produces.

In the next sections, we introduce specific algorithmic techniques for implementing these modules. While these techniques are intuitive and are shown to produce accurate estimates of quality, there are many possible implementations for each module. A key contribution, beyond the specific algorithms we present, is in the conceptual framework of a quality tracking pipeline. This pipeline delivers information that provides applications with accurate estimates of the quality of the ODA data on which they are operating.

# 4 Smoothing

Recall that temporal smoothing aggregates expected readings across time, taking advantage of the redundancy of readings in time. Spatial smoothing, on the other hand, spatially aggregates streams from multiple sensors that are monitoring the same logical area; it combines readings from individual unreliable sensors to produce a single data stream with higher quality.

Both types of smoothing operate over a window of readings of width $W$. We assume windows slide by 1 time-unit, although our approach applies directly to other slide granularities. Temporal smoothing uses a time window of $W_t$ time-units; spatial smoothing uses a *spatial window* consisting of the readings from $W_s$ sensors in a logical area for the same time-unit. The output of either type of smoothing is a reading for each window (i.e., a reading at each time-unit) if there exists at least one reading within the window.

Both temporal and spatial smoothing are designed to increase the coverage of the data stream by filling in for possible missed readings. Confidences may however increase or decrease. We now show how these estimates are recalculated during both types of smoothing.

## 4.1 Confidence Calculation

Consider a particular object $O$ in a temporal (for temporal smoothing) or spatial (for spatial smoothing) window of size $W$ (we use $W$ to represent either $W_t$ or $W_s$). Let the number of readings of object $O$ in the window be $r$, and let their confidences be $c_1, c_2, \ldots, c_r$.

Since our aim is to have accurate confidence estimates, we want to assign a high confidence to an output reading if the corresponding object was actually present in reality, and a low confidence if it was not. We estimate confidence based on this intuition: the greater the evidence we have of an object being present in a logical area, the greater is the output confidence for that reading. Intuitively, two factors contribute to the evidence of the presence of the object within a window:

- The greater the individual confidences of the input, and hence greater their sum, the greater is the evidence of a present object.
- The greater the number of input readings, the greater the evidence of a present object.

While there are potentially many ways to account for the above factors in the output confidence, we use a linear combination of these factors. In Section 8, we show empirically that this approach produces accurate confidence estimates. After smoothing readings with confidences $c_1, c_2, \ldots, c_r$, both types of smoothing emit an output reading for object $O$ with confidence $E_{conf}$ given by Equation 1:

$$E_{conf} = \frac{1}{W}[\alpha S + (1-\alpha)r] \tag{1}$$

where $S = \sum_{i=1}^{r} c_i$ and $W$ is either $W_t$ or $W_s$

Note that we use the same equation for both temporal smoothing and spatial smoothing. The weighting factor, $\alpha$, could be different in the two cases. In Section 8, we experimentally determine choices for $\alpha$.

## 4.2 Coverage Calculation

We revise the coverage estimates after smoothing based on the number and confidences of the readings introduced in the coverage time window. (Recall that coverage is defined over a time window that may be different than the time window used in temporal smoothing.) Consider a particular such window over which the coverage is defined, and let the incoming coverage of this window be $E_{cov}^{old}$. Let the incoming data stream have $r$ readings in this window, with confidences $(c_1, c_2, \ldots, c_r)$. Note coverage applies to the entire window of readings, not to individual objects.

For coverage calculation, we distinguish between output readings for which there was an input reading with the same oid, time, and location, and output readings that have no such input reading (i.e., readings that were "smoothed in"). These newly added readings boost coverage. Let there be $s$ such newly introduced readings, with confidences $c_{i_1}, c_{i_2}, \ldots, c_{i_s}$. We find the new coverage by calculating the increase in the expected number of readings. Since the confidence gives an estimate of the probability of a reading being present in reality, the sum of confidences gives the expected number of readings in reality. The coverage calculation is given in Equation 2.

$$E_{cov}^{new} = E_{cov}^{old} + \frac{(\sum_{k=1}^{s} c_{i_k})}{N_{real}} \tag{2}$$

Here, $N_{real}$ is the expected number of readings in reality, given by $(\sum_{k=1}^{r} c_k)/E_{cov}^{old}$. The first term is the incoming coverage estimate and the second term gives the estimated increase in the coverage.

# 5 Arbitration

Arbitration is designed to detect objects that were reported by sensors monitoring a different (but possibly adjacent) logical area to the one the object was actually in. Arbitration attempts to determine which of the conflicting readings are correct, and which are false positives. (Note that false positives due to spurious readings would be caught by smoothing, not by arbitration.)

We assume that arbitration does not remove any readings, but only adjusts the confidences. For example, if sensors for two logical areas both report an object with an equal confidence, it would be desirable to retain both of the readings with lower confidence since they are in conflict with each other. We assume that conflicts are better resolved at a higher-level such as by a human or application-level logic. Alternately, the arbitration module could eliminate all but one (or few) readings per object by applying a filter to remove low confidence readings after arbitration. We discuss such a filtering approach in Section 6.

### 5.1 Confidence Calculation

We now show how to revise the confidences of readings in arbitration. The input to arbitration is a set of $n$ data streams, one from each of the application's logical areas, $L_1, L_2, \ldots, L_n$. Let the input streams contain $r$ ($r \leq n$) conflicting readings for an object and time-unit in different logical areas. We denote the confidences of these $r$ readings by $c_1, c_2, \ldots, c_r$ respectively.

Since each conflicting reading reduces the likelihood of any other conflicting reading being correct, we reduce the confidences of each conflicting reading. We estimate the revised confidence $c_i^{new}$ for each reading based on the following considerations:

- Each additional logical area reporting an object reduces the new confidences of each of the outgoing readings of the same object in that time-unit.
- The higher the confidence of conflicting readings, the greater should be the decrease in their updated confidence values. This is because the higher the confidence of a particular reading, the lower the chance that the object is present in some other logical area.

We update the confidence of outgoing readings by scaling down each confidence by using the probability that the object is not present in the other logical areas. Equation 3 shows the formula for the new confidence.

$$c_i^{new} = c_i * \Pi_{j=1, j \neq i}^{j=r} (1 - c_j) \tag{3}$$

Since $c_i$ estimates the probability of the object being present in logical area $L_i$, $(1 - c_i)$ estimates the probability of it not being in $L_i$.

### 5.2 Coverage Calculation

Arbitration does not change coverage since it does not remove any readings. If readings are removed in arbitration, it is equivalent to applying a filter on the low confidences after arbitration, which we discuss next.

## 6 Filtering Based on Confidence

With confidences associated with each reading in the ODA data stream, one technique for cleaning the stream could be to eliminate all readings with a confidence below some threshold. Removing readings that have a low probability of existing in the sensor's logical area should eliminate errant readings and lower false positives.

When readings with confidence below a threshold are dropped, the confidence of the remaining readings are unchanged. We next show how the coverage of the resulting data stream is updated.

### 6.1 Coverage Calculation

The coverage of the data stream after filtering is computed in a manner similar to that of coverage after smoothing. Consider a window with $N$ readings in the reality stream over which the coverage needs to be re-estimated. Let its original coverage estimate be $E_{cov}^{old}$. Let the ODA data stream have $r$ readings in this window, with confidences $c_1, c_2, \ldots, c_r$. If $s$ readings with confidences $c_{i_1}, c_{i_2}, \ldots, c_{i_s}$ are eliminated after applying the threshold, we revise the coverage based on the expected drop in the number of readings from these $s$ readings, as show in Equation 4.

$$E_{cov}^{new} = E_{cov}^{old} - (\sum_{k=1}^{s} c_{i_k})/N_{real} \tag{4}$$

Here $N_{real}$ is the expected number of readings in reality, given by $(\sum_{k=1}^{r} c_k)/E_{cov}^{old}$.

In our experiments in Section 8, we consider other variations of filtering, such as applying thresholds for high as well as low confidences, and rounding confidences to 1 and 0 respectively.

## 7 Example: RFID Data

For the remainder of the paper, we ground our discussion in a concrete object-detection application: detecting tagged objects using RFID technology. Here we give a brief background on RFID technology followed by the presentation of a simple RFID deployment to illustrate the data processing and cleaning described in this paper. We

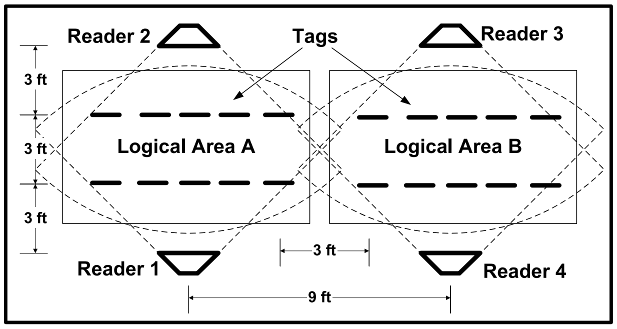


Figure 3: RFID Setup.

use this deployment in the next section for our experimental evaluation.

## 7.1 Background: RFID Technology

A typical RFID deployment consists of the following components:

- *Readers:* Readers use antenna to communicate with tags in the vicinity. In many RFID installations, multiple readers are grouped into units that are responsible for monitoring the same logical area [3].
- *Tags:* RFID tags have unique identifiers, and are placed on objects to be tracked. These tags respond to the readers, informing them of the object's ID.
- *Middleware:* RFID middleware serves as a bridge between RFID hardware and applications using RFID data by cleaning and processing the data streams before passing them on to the application.

Figure 3 shows an example deployment with two adjacent logical areas, each containing 10 tags. A total of four readers are present, two each for monitoring a logical area.

## Reader Operation

Readers *interrogate* nearby tags which respond with a unique identifier code. Interrogations are grouped into *epochs*, which correspond to our logical time-units. Readers keep track of the tags seen as well as metadata such as the number of interrogations to which each tag responded in the last epoch in a *tag list*. RFID middleware retrieves the tag list each epoch either by polling the reader, or the reader itself pushes the information after each epoch.

For more information on RFID technology, a detailed primer is available in [30].

## 7.2 Middleware Data Processing and Cleaning

RFID middleware systems typically process the data streams using a pipeline of cleaning modules similar to those discussed in this paper [7, 16, 29, 20].

The raw data stream in our setup has a schema as follows: `<tag_id,epoch,reader_id>`. The `reader_id` determines the location. In our deployment, temporal smoothing is applied for each tag_id and reader_id separately. Spatial smoothing aggregates across the readers monitoring the same location. Arbitration occurs across readings for the same tag detected in multiple logical areas in the same epoch. In our experiments, we do not filter based on confidence during the pipeline, but explore various types of thresholding at different points in the pipeline. These options are discussed in Section 8.2.

## 7.3 Initial Quality Estimation

Here we show how to initially estimate the quality of the raw RFID data stream.

### 7.3.1 Confidence Calculation

Initially, an RFID middleware system receives a tag list from an RFID reader. As previously mentioned, each tag_id in the tag list contains the number of times it has been read in the epoch, which can be used to calculate the *read rate*. The read rate for each tag is the ratio of responses to total interrogations, $\frac{R}{I}$.

We use the read rate to derive the initial confidence for each reading. For each tag $t$ the reader reports the number of times it was detected $R_t$ out of a total of $I$ interrogations. The initial confidence for this reading of $t$ is assigned to be $\frac{R_t}{I}$.

### 7.3.2 Coverage Calculation

We estimate the initial coverage using a simplified model of an RFID reader's detection field based on empirical

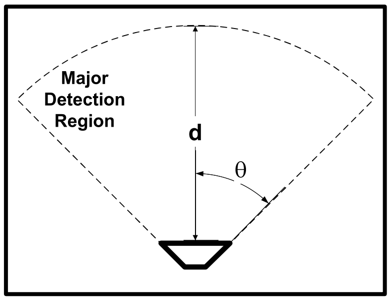


Figure 4: Simplified RFID detection model. The probability of detection within the major detection region is $p$; outside it is $s$.

studies (as shown in Figure 4) [26]. In this model, the reader has a *major detection range* where there is a probability $p$ of detecting a tag within a distance of $d$ at an angle of within $\theta$ of the reader. Any tag outside of the major detection range has a low probability $s$ of detection.

Ideally, the major detection range would constitute the logical area for the reader. Thus, the probability of detection for any tag in the logical area is $p$, and hence the expected fraction of correct readings present in the RFID stream is $p$. We therefore use $p$ as the initial coverage for the data stream.

## 8 Experimental Evaluation

In this section, we experimentally evaluate our algorithms for tracking quality using the RFID scenario discussed in the previous section. We validate our approach by measuring the accuracy of our confidence and coverage estimates, $E_{conf}$ and $E_{cov}$. We denote the accuracy of these estimates by $A_{conf}$ and $A_{cov}$ respectively. Intuitively, $A_{conf}$ and $A_{cov}$ give the closeness of the estimates $E_{conf}$ and $E_{cov}$ to the actual confidence and coverage. This notion is formalized below.

We look at each reading in the data stream. For an incorrect reading, a lower $E_{conf}$ implies higher accuracy and for a correct reading, a higher $E_{conf}$ implies a higher accuracy. Therefore, we define $A_{conf}$ to be $(1 - E_{conf})$ for incorrect readings and $E_{conf}$ for correct readings.

Coverage is by definition at the window level, and the accuracy of its estimate is also defined at a window level. If the coverage estimate of the data stream for a particular time window is $E_{cov}$, its accuracy $A_{cov}$ is given by $(1 - |E_{cov} - c|)$, where $c$ is the actual coverage of the data stream for that window in reality. We now define the estimation accuracy for an entire data stream $D$.

**Definition 8.1 (Estimation Accuracy of $D$)**

- *The accuracy of confidence estimates of $D$ is the average of $A_{\text{conf}}$ for all readings in $D$.*
- *The accuracy of coverage estimates of $D$ is the average of $A_{\text{cov}}$ for all readings in $D$.* □

Before moving to the details of the experimental setup and results, we summarize our findings:

- Our online algorithms accurately estimate the quality of the stream; in most cases, the accuracy of the quality estimates is above 80%.
- The estimates of quality remain accurate after propagating the data through multiple pipelined cleaning modules.
- The estimates are far better than having no quality information, i.e., assuming confidence and coverage of 1.
- Our techniques are robust with respect to the cleaning techniques and "dirtiness" of the data.
- Accurate assessment of quality can also help in improving the quality by filtering appropriately.

### 8.1 Experimental Setup

We use a simulated RFID deployment depicted in Figure 3 for our experiments. This deployment consists of two logical areas, each containing 10 tags arranged in two rows separated by 3 feet. Each logical area is monitored by 2 RFID readers, placed 3 feet from the first row of tags on either side.

Based on the RFID detection model illustrated in Figure 4 and this RFID deployment, we generate synthetic RFID data streams for use in our experiments. Here, we set $p$ to 0.9, $d$ to 8, $\theta$ to 45, and $s$ to 0.1. During each epoch, the simulated readers attempt to detect each tag using a biased coin-flip based on either $p$ or $s$ (depending on where each tag is located relative to the reader).

It is important to note that our techniques are robust to the actual parameters of the RFID detection model. For instance, we show later that as we vary either $p$ or $s$, our estimates remain accurate. Furthermore, we emphasize that RFID data is merely an example of our quality estimation schemes. Our techniques can track the quality of any type of ODA data.

This deployment demands all three types of cleaning modules: temporal smoothing, spatial smoothing, and arbitration. Temporal smoothing is needed to account for false negatives produced by the unreliable readers. Spatial smoothing is used to further alleviate the effects of false negatives using multiple readers associated with the same logical area. Finally, due to the proximity of the two logical areas, arbitration is necessary to remove duplicate readings.

We give experimental results for confidence in Section 8.2 and then for coverage in Section 8.3.

## 8.2 Experimental Results: Confidence

For our experimental evaluation of confidence estimation, we first test all cleaning modules individually, then test different pipelined configurations of two or more modules. We then test optimizations based on thresholding and rounding. We show our results are far better than absence of quality estimation, and finally show robustness to the quality of data.

### 8.2.1 Temporal Smoothing

We first test the accuracy of our quality estimation for a single temporal smoothing module. Here, we run temporal smoothing over each stream from the four readers and measure the accuracy of our confidence estimates. We use the basic temporal smoothing strategy as described in Section 2.1 and set $\alpha$ to 0.6. (In this experiment we tried varying $\alpha$ with little effect. Tuning $\alpha$ in pipelined cleaning is discussed later.)

The overall accuracy of our confidence estimation, given by Definition 8.1, is $0.85$. This shows that our confidence estimates are very close to the actual confidence. Hence, low confidences are assigned most often for wrong readings, and high confidences for correct readings.

To further explore how our confidence estimation performs, we plot the distributions for our confidence estimations for both correct readings in the cleaned stream (true positives) and incorrect readings (false positives). Figure 5 shows these distributions.

The graphs show that, as desired, our algorithm for temporal smoothing confidence estimation does indeed assign a high confidence for correct readings and very low confidence to false positives. For most readings in the temporally smoothed data stream that are correct (i.e., the corresponding object exists in the reported location at the reported time), our estimates assign a high confidence between 0.8 and 1 (Figure 5(a)). For incorrect readings in the data stream, our estimates assign a low confidence; most false positives are given a confidence less than 0.4 (Figure 5(b)).

To show that our estimation algorithm can handle different types of smoothing, we ran the same experiment with other variants of temporal smoothing. We tested schemes with the sliding window starting, ending, and centered at $t$, the time of the reading being emitted. We obtained similar results for all these cases: the confidence accuracy was $0.90$, $0.89$, and $0.89$ for the three cases respectively. Therefore, our approach appears robust to these variants of temporal smoothing.

### 8.2.2 Spatial Smoothing and Arbitration

We also test the accuracy of our quality estimates after only spatial smoothing. We run spatial smoothing over the streams from the four readers, where the readings from each of the two logical areas are smoothed together. The accuracy of our confidence estimation in this case is $0.84$.

We test our estimation algorithm when only running an arbitration module over the data streams of the four readers. Using only arbitration, we obtain a confidence accuracy of $0.90$.

Detailed results for these two experiments are omitted as they are similar to only applying temporal smoothing.

### 8.2.3 Pipelined Cleaning

In many ODA cleaning tools, two or more cleaning modules are assembled together in a pipeline to produce higher quality data. Next we show experimentally that our quality estimation algorithms can track confidence as the streams pass through multiple modules.

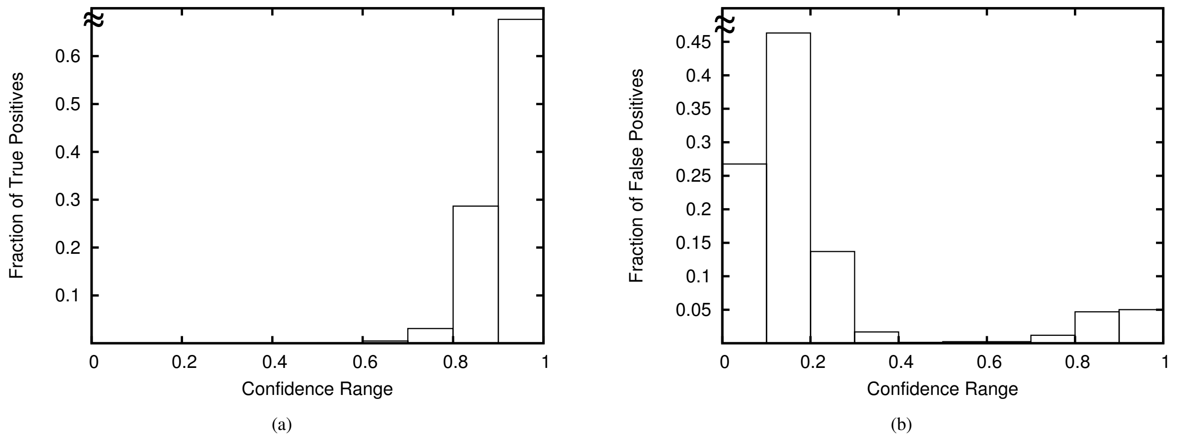


Figure 5: Confidence distributions for true/false positives after temporal smoothing.

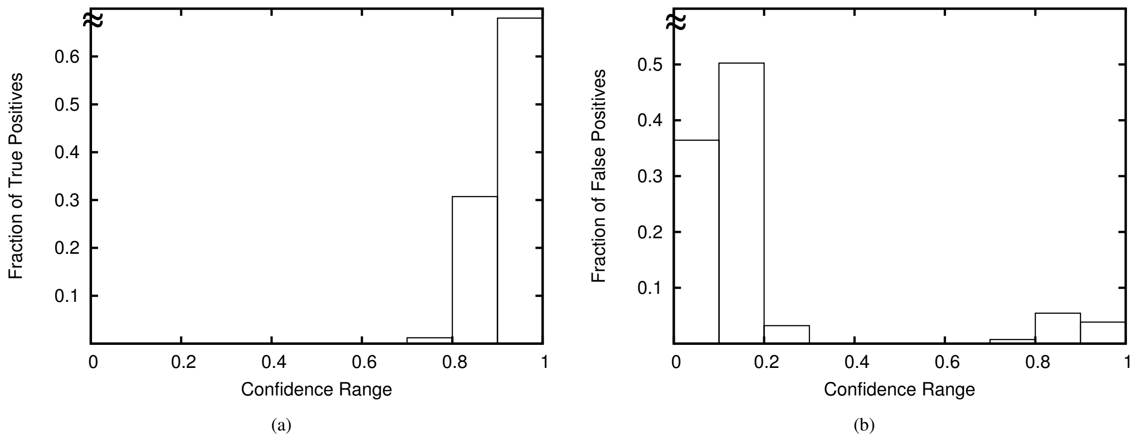


Figure 6: Confidence distributions for true/false positives after temporal + spatial smoothing.

**Temporal + Spatial Smoothing:** The first test involves a two-module pipeline containing temporal smoothing followed by spatial smoothing. The streams from each reader are temporally smoothed, then the streams from each logical area are spatially smoothed. The accuracy of our confidence estimation algorithms for the final output stream is $0.86$.

Again, we further analyze our algorithms by plotting the distribution of confidences for both true positives and false positives, shown in Figure 6. Once again, our algorithm does indeed assign a high confidence for correct readings and very low confidence to false positives. These distributions very closely resemble the distribution for only temporal smoothing, demonstrating that our algorithms are capable of tracking the data stream's quality as it passes through multiple modules, without a degradation in estimation accuracy.

**Tuning the Spatial Smoothing Parameter $\alpha$:** Recall the algorithm for confidence estimation involves the weighting factor $\alpha$: lower values of $\alpha$ weight towards the number of readings in a window, while high values favor the sum of the input confidences. Temporal smoothing is not sensitive to $\alpha$. Now we study the variation of accuracy with spatial smoothing's $\alpha$ for the two-module pipeline as above (temporal + spatial smooth).

Interestingly, with spatial smoothing following temporal smoothing, we see a steady increase in the accuracy of our estimates as we increase $\alpha$. That is, giving more weight to the actual confidences and not to the count of tags improves accuracy. The reason for this improvement is that during temporal smoothing, if a tag from an adjacent region is read even once in a particular time window, it gets added to the result with a low confidence (i.e., one false positive reading in the underlying raw stream produces multiple false positives, but with low confidence, in temporal smoothing). Hence, the confidence of readings are likely to be more reliable than just their presence or absence. We see that $\alpha = 1$ yields the best estimation accuracy, and thus we set $\alpha = 1$ for the remainder of the experiments involving this pipeline. Note that for other module assemblies, $\alpha$ should be similarly tuned and will not always be 1.

**Pipeline of Temporal Smoothing, Spatial Smoothing, and Arbitration:** We now give end-to-end accuracy results for a pipeline with all three types of cleaning modules. We clean the streams produced by the four readers first using temporal smoothing in each reader's stream, then spatially smoothing to produce a stream from each logical area, and finally using arbitration to account for object readings that appear in both areas at the same time.

Our quality estimates on the output after these three successive cleaning modules have an accuracy of $0.86$. As we shall see later, this accuracy is further improved to $0.95$ by the rounding and thresholding algorithms.

Once again we analyze the confidence distributions for true and false positive readings, shown in Figure 7. Similar to the other cleaning module configurations, in general correct readings are assigned high confidences and incorrect readings low confidences.

There is, however, an interesting difference between the distributions here versus those in Figures 5 and 6. Before arbitration, some false positives were assigned too high a confidence: some false positives received confidences higher than 0.8 and many received a confidence of 0.1-0.2. These high confidence estimates are all reduced after arbitration, where all false positives receive a confidence of 0-0.1. This effect is observed in the small peaks for the high confidence range in Figures 5(b) and 6(b) that have vanished in Figure 7(b). Conversely, arbitration incorrectly lowers the confidence of some correct readings, resulting in a small peak for the low confidence range in Figure 7(a) that is absent in Figures 5(b) and 6(b). This effect is due to the fact that arbitration lowers the confidence of readings, overall thus shifting the entire distribution of confidences to the left.

**Other Cleaning Module Pipelines:** Next we experiment with different pipelines and orderings of the cleaning modules. All variations we tested produced similar estimation accuracies as those shown above.

For instance, using spatial smoothing followed by temporal smoothing gives equally accurate estimates as the reverse ordering. The overall confidence accuracy was $0.90$, and the confidence distributions for correct and wrong readings are shown in Figures 8(a) and 8(b). As another example pipeline, spatial smoothing followed by arbitration yields an accuracy of $0.85$.

We omit results for all possible pipelines and orderings because of space constraints, but in summary, our techniques appear indifferent to the manner in which the cleaning modules are composed.

#### 8.2.4 Rounding and Threshold Modules

Next we evaluate two different post-processing modules that further improve the accuracy of our estimates: *rounding* and *thresholding*.

The *rounding module* modifies confidences of readings based on a *rounding distance* $rd$, $0 \leq rd \leq 0.5$: readings with confidence below $rd$ are reduced to confidence 0, and readings with confidence between $(1 - rd)$ and 1 are pushed up to confidence 1. Confidence of readings between $rd$ and $(1 - rd)$ remain unchanged. This form of rounding is motivated by the fact that if our estimates of confidence are reasonably high, we are almost sure that the tag is indeed present, and conversely, if the confidence is very low, we are very sure the tag is not present.

The *threshold module* is more extreme: it assigns a confidence of either $1$ or $0$ to every reading in the cleaned data stream. The value at or above which every confidence is set to $1$ is the *threshold* $t$, and every confidence below $t$ is set to $0$. This algorithm is important because for some applications, the application must either believe or dis-

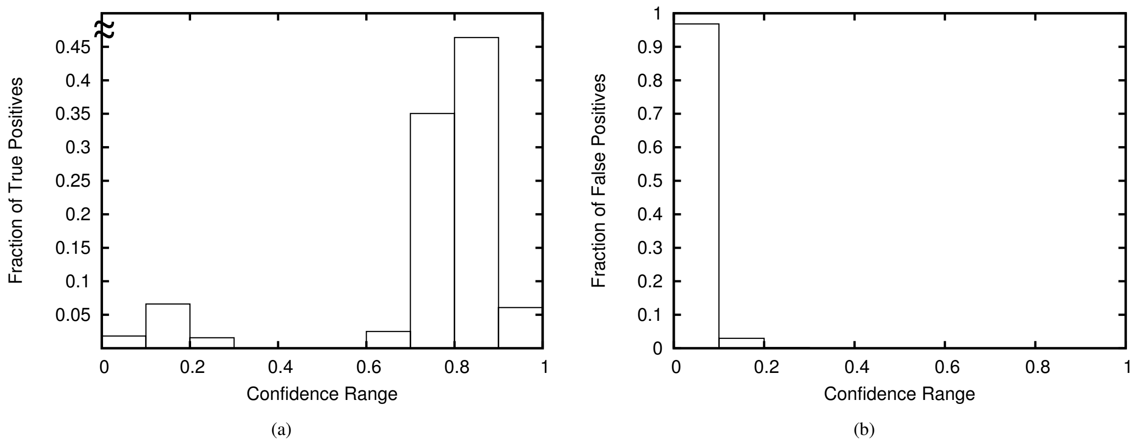


Figure 7: Confidence distributions for true/false positives after temporal + spatial smoothing + arbitration.

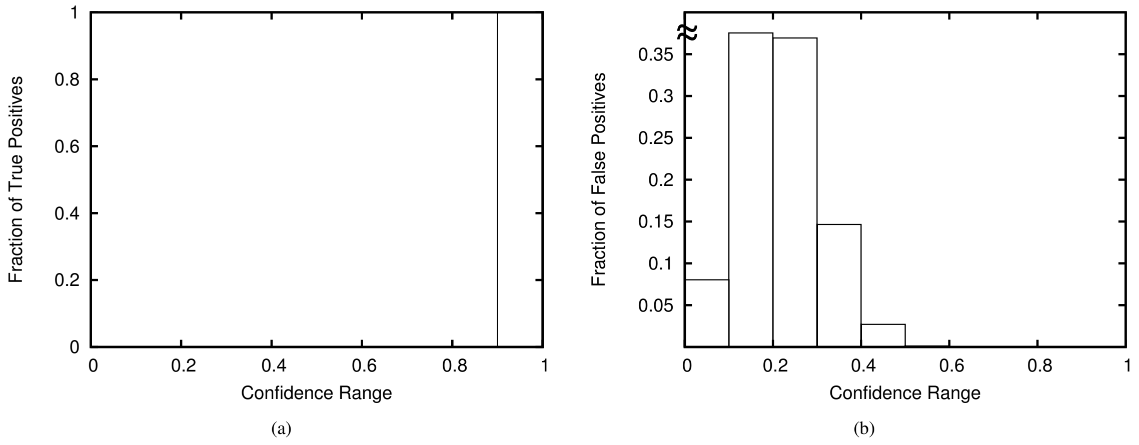


Figure 8: Confidence distributions for true/false positives after spatial + temporal smoothing.

believe every reading; for instance, RFID tags are either present or are not present.

Both of these post-processing modules improve accuracy of our estimates, so are useful in that regard. Furthermore, if we want to improve the overall data stream quality, we can use the threshold module and then drop all readings with confidence $0$.

**Rounding Module:** We evaluate the rounding module for different cleaning module configurations. For rounding used after three different configurations (temporal smoothing; temporal + spatial smoothing; temporal smoothing + spatial smoothing + arbitration), we vary the round distance $rd$ and measure the accuracy of the confidence estimate. Figure 9 shows the results. Since the differences are small, we have truncated the y-axis to highlight the results.

As can be seen, the accuracy improves with an increase in the round distance. The increase of accuracy with $rd$ indicates good performance of the confidence estimates as the algorithm separates the correct readings from the incorrect ones. This effect is expected, given the confidence distributions of the correct/incorrect readings produced by our algorithms as shown in Figures 7(a) and 7(b). Most

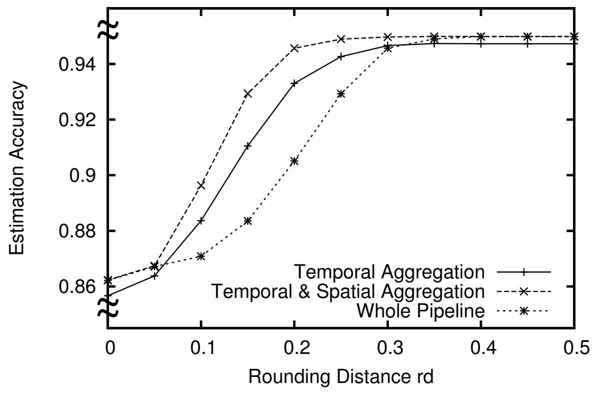


Figure 9: Rounding algorithm accuracy.

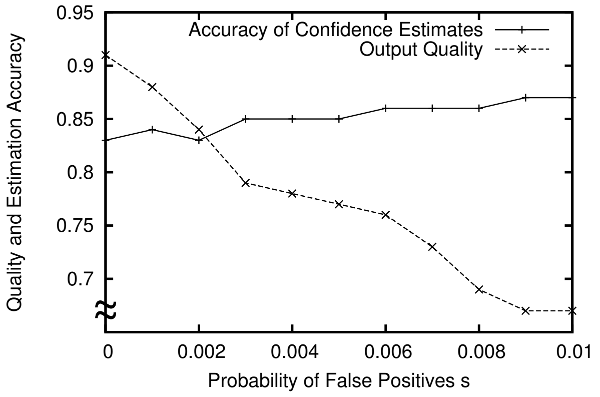


Figure 11: Confidence estimation accuracy versus quality.

correct readings receive a high confidence, so rounding them up to 1 increases the estimation accuracy, and similarly rounding down low confidences to 0 also increases the estimation accuracy. These results suggest that for this scenario we use a round distance of $0.5$, and thus effectively make all confidences 0 or 1 based on whether they are lower or higher than $0.5$ respectively.

**Threshold Module:** Here we analyze the performance of the threshold module applied after each pipeline configuration, similar to above. We vary the confidence threshold $t$ and measure the resulting accuracy of our estimates. Figure 10 shows the variation of accuracy with $t$ when applying the threshold module after various configurations of the pipeline.

For each cleaning module configuration, thresholding substantially increases accuracy. For the first two pipelines, a threshold of $t = 0.5$ gives the highest accuracy, and for the full pipeline the optimal threshold shifts down to $0.1$. These results indicate that false positives do end up getting a very low confidence. An equally important observation is that for all of the pipeline configurations, the accuracy of the threshold algorithm is very high for a wide range of threshold values. This result shows that the confidence estimates successfully separate the correct and incorrect readings.

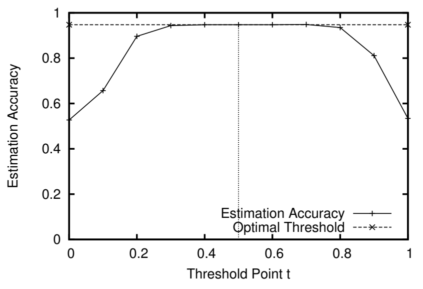
### 8.2.5 Comparison with Random and Unit Confidences

We now compare the accuracy of our estimation algorithm against the accuracy of naive algorithms that (1) assign random confidences to every reading, or (2) assign unit confidences (confidence 1) to every reading. Assigning unit confidences is equivalent to not having any quality estimation information, and hence "believing" every reading. We choose the optimal values of $rd$ and $t$ for the cases where we apply a rounding or threshold module after the indicated pipeline. Table 1 shows the estimation accuracy for various algorithms and for different stages of the pipeline. Clearly, our estimates perform far better than the naive random or unit confidence approaches, justifying the usefulness of our estimates over naive assumptions.

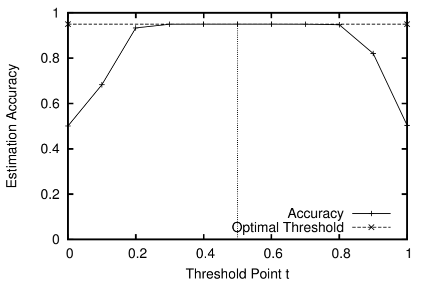
Finally, as noted earlier the rounding and threshold modules further improve the accuracy of our estimates. Therefore, to improve overall quality of data delivered to an application, we could choose to drop readings below a certain threshold confidence, and set the confidence of the remaining readings to 1.

### 8.2.6 Robustness to the Quality of Data

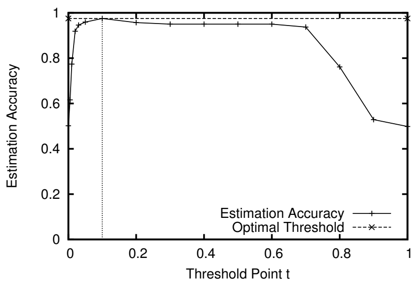
Next we show that our algorithms are robust to the "dirtiness" or actual quality of the underlying data. That is, our quality estimates are accurate irrespective of the actual quality of the data streams being cleaned. Figure 11



(a) Temporal smoothing



(b) Temporal + spatial smoothing



(c) Full pipeline

Figure 10: Accuracy of the thresholding algorithm applied after different cleaning pipeline configurations.

| **Confidence Estimation Accuracy** | **Random** | **Unit** | **Standard** | **Rounding** | **Threshold** |
|---|---|---|---|---|---|
| Temporal Smoothing | 0.50 | 0.53 | 0.86 | 0.95 | 0.95 |
| Temporal + Spatial Smoothing | 0.50 | 0.50 | 0.86 | 0.95 | 0.95 |
| Whole Pipeline | 0.50 | 0.50 | 0.86 | 0.95 | 0.98 |

Table 1: Comparison of Confidence Estimation Algorithms

shows the variation of estimation accuracy with the actual quality for the whole pipeline of temporal and spatial smoothing followed by arbitration. We vary the quality of the output by increasing the number of false positives in the input, shown on the x-axis. The actual confidence of the stream is given by the fraction of correct readings in the stream. As can be seen from the graph, though the actual quality of the stream drops on increasing the false positives, the accuracy of our confidence estimates remains stable.

## 8.3 Experimental Results: Coverage

**Coverage Estimate Accuracy:** We first give results for the accuracy of our coverage estimates. As mentioned earlier, we set the initial coverage of every reading from a reader to $p$, the probability of detection in the reader's logical area. Since arbitration does not change coverage, we study estimation accuracy for temporal smoothing and/or spatial smoothing. Figure 12 gives the estimation accuracy for: (1) initial quality estimates; (2) estimates after only temporal smoothing; (3) estimates after only spatial smoothing; and (4) estimates after temporal smoothing followed by spatial smoothing. We show the accuracy as we vary the probability of detection $p$.

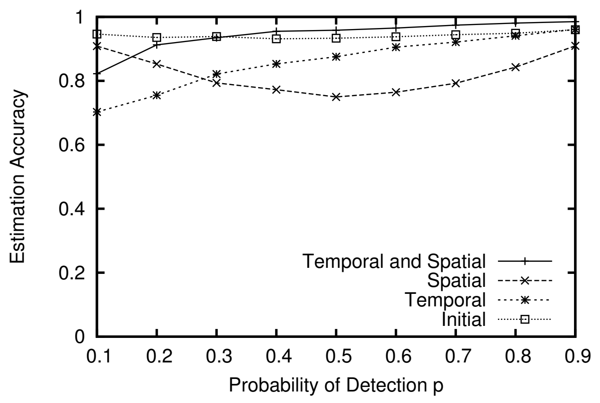


Figure 12: Coverage accuracy using pipelines consisting of temporal and spatial smoothing.

First, notice that the accuracy of our estimates across all values of $p$ are above 0.7, and in most cases above 0.9. This shows that our estimates of the coverage are very close to the actual coverage for most windows. Moreover, the accuracy remains high as the stream is cleaned through the pipeline. In fact, after both temporal and spatial smoothing, our coverage estimates are well over 0.9 for most values of $p$.

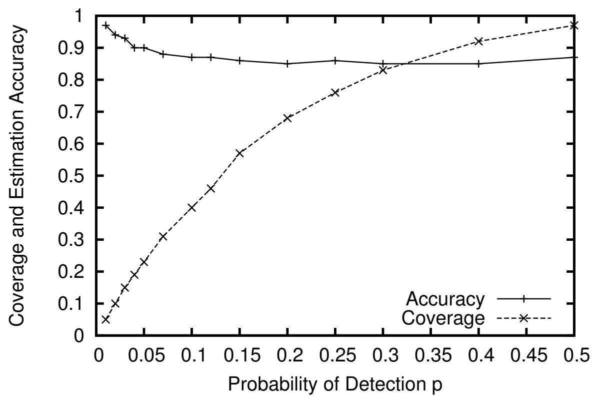


Figure 13: Estimate accuracy with varying data stream coverage quality after temporal smoothing.

**Robustness to Actual Coverage:** We demonstrate how our algorithms estimating coverage respond to different levels of coverage in the data stream.

We first vary the coverage of the cleaned stream by running only temporal smoothing with a window size of 5 epochs over streams produced with different values of $p$. By varying $p$ from 0.01 to 0.5 with a fixed window, the coverage varies almost from 0 to 1. Figure 13 shows both the coverage of the data stream and the accuracy of our quality estimates as we vary the value for $p$.

When the probability $p$ of detection is very low, the temporal smoothing module is unable to adequately smooth false negatives and thus it produces a data stream with a very low coverage. Our techniques are able to recognize this poor coverage and accurately estimate it as such. As $p$ increases, the coverage of the stream increases, and our algorithms detect this change. Despite the large range in coverage, the accuracy of our coverage estimations remain relatively high.

Finally, we test the accuracy of our coverage estimates as we change the size of the temporal smoothing window $w$. We fix $p = 0.3$ to produce somewhat unreliable data, so the cleaning modules can increase coverage by a measurable degree. (The accuracy of our estimates are similar for other values of $p$.) Figure 14 shows

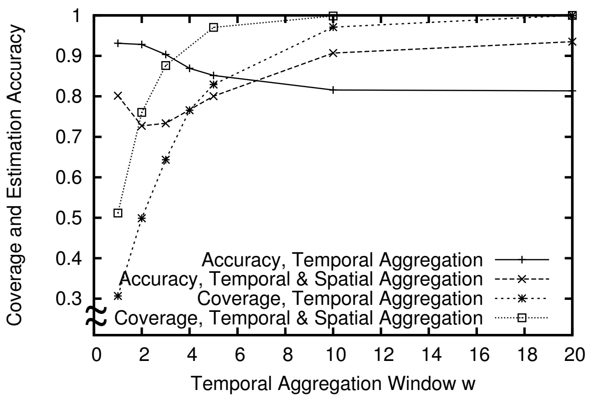


Figure 14: Varying temporal smoothing window size.

both the actual coverage and the accuracy of our coverage estimates as we vary the temporal smoothing window size. We show results for temporal smoothing alone and a pipeline consisting of temporal smoothing followed by spatial smoothing.

As expected, increasing $w$ produces data streams with better coverage. Again, despite the large change in stream quality, the accuracy of our estimates stays high.

These experiments illustrate that our techniques are robust to poor data cleaning mechanisms: we are able to produce accurate estimates of quality regardless of how well the data stream is cleaned.

## 8.4 Experimental Results Summary

We have demonstrated empirically that our techniques for tracking quality are accurate and provide significant benefit over no quality information at all. Furthermore, our estimates remain accurate as the data is processed through multiple cleaning modules, and they are robust to a wide range in the actual "dirtiness" of the data.

## 9 Related Work

There is a large body of work relating to various aspects of ODA applications, and ODA data processing, but less existing work on data quality assessment.

Many platforms produce ODA streams, e.g., [10, 17, 18, 22, 27, 28]. While some of these systems perform limited low-level data correction, none provide an indication of the quality of the streams produced.

Data quality of RFID streams has received particular attention both commercially and in research. RFID middleware systems typically use a pipeline of filters designed to convert raw RFID data to application data [7, 16, 29]. Many of these systems incorporate stages closely related to the modules addressed in this paper. Other projects have addressed, to various extents, cleaning of sensor data [9, 13, 25, 20, 21]. Again, no quality estimates are included, so our techniques are designed to augment these systems.

Although not concerned with data cleaning, the BBQ system addresses data reliability issues for physical sensors [12]. BBQ models sensor data as a set of time-varying Gaussians. Over these models, it answers user queries based on a user-specified confidence. User-driven quality requirements present an interesting application for our techniques: a system can use the quality estimates our mechanisms provide to more effectively answer queries.

Confidence and coverage were introduced in [31] in the context of a general-purpose system for uncertain data. There has been substantial prior work on uncertain, probabilistic, approximate, inconsistent and imprecise databases, and on approximate query answering; examples include [1, 5, 19, 11, 15, 2]. While a large fraction of this work is theoretical, there are several systems that manage uncertainty, e.g., [4, 8, 23, 31]. In this paper we estimate quality rather than manage it, and we do so in a data stream environment targeted specifically for data cleaning and ODA applications.

## 10 Conclusions and Future Work

As sensor device deployments are becoming more ubiquitous, object detection applications (ODAs) are gaining importance in diverse areas such as supply chain management and wildlife tracking. The data streams on which these applications are built, however, tend to contain numerous errors. Without accurate knowledge of data quality, these applications may not function correctly.

To address this issue, we developed a *quality tracking pipeline* that shadows the main data cleaning pipeline and propagates quality estimates with the data as the streams flow through the cleaning modules. We developed online algorithms for modules in this shadow pipeline that estimate the quality of the data as the ODA streams are processed through typical ODA cleaning modules. These modules track the quality of the data stream using two metrics, confidence and coverage, and we show how estimates of these measures can be accurately propagated through multiple cleaning modules in a streaming fashion.

We report an extensive case study of our techniques on an RFID application. We show how the quality of the raw RFID data stream is estimated, then provide a suite of experimental results for various cleaning modules and assemblies of them in a pipeline. Our experiments show that for this scenario our intuitive quality estimation algorithms work well, and are far better than assuming accurate data. Our experiments also show that the estimates of confidence can be used to improve the quality of the stream. We are not aware of any past work that estimates the quality of data streams in an object-detection application.

Our work poses some new and interesting directions for future work. ODA applications may be inherently tolerant to some quality loss in the data stream, and they could specify this tolerance to the middleware cleaning and estimation modules. This quality tolerance might then be used as input to the middleware to optimize data processing. The overall idea is to increase efficiency by minimizing the data processing to achieve the required quality.

We showed that confidence estimates could be used to help improve the quality of the stream to some extent. An important direction of future work is to more closely tie the quality improvement and quality estimation modules, since we have evidence that each of these can benefit from the other.

## References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated Ranking of Database Query Results. In *Proc. of CIDR*, 2003.

[3] Application Level Event (ALE) Specification Version 1.0. http://www.epcglobalinc.org/standards_technology/EPCglobal_ Application_Level_Events %20%28ALE%29_Specification_v1.pdf.

[4] D. Barbará, H. Garcia-Molina, and D. Porter. The Management of Probabilistic Data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[5] R. S. Barga and C. Pu. Accessing Imprecise Data: An Approach Based on Intervals. *IEEE Data Engineering Bulletin*, 16(2):12–15, 1993.

[6] Barnaby J. Feder. Despite Wal-Mart's Edict, Radio Tags Will Take Time. *New York Times*, Dec 27 2004.

[7] C. Bornhövd, T. Lin, S. Haller, and J. Schaper. Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure. In *VLDB*, pages 1182–1188, 2004.

[8] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *Proc. of ACM SIGMOD*, pages 891–893, 2005.

[9] P. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden. TASK: Sensor Network in a Box. In *EWSN*, 2005.

[10] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. E. Sarma. Managing RFID Data. In *VLDB*, 2004.

[11] N. Dalvi and D. Suciu. Answering Queries from Statistics and Probabilistic Views. In *Proc. of VLDB*, 2005.

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

[13] E. Elnahrawy and B. Nath. Cleaning and querying noisy sensors. In *WSNA*, 2003.

[14] A. Faradjian, J. Gehrke, and P. Bonnet. GADT: A Probability Space ADT for Representing and Querying the Physical World. In *Proc. of ICDE*, pages 201–211, 2002.

[15] N. Fuhr. A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In *Proc. of VLDB*, 1990.

[16] A. Gupta and M. Srivastava. Developing Auto-ID Solutions using Sun Java System RFID Software. Oct 2004.

[17] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, G. Zhou, J. Hui, and B. Krogh. Vigilnet: An Integrated Sensor Network System for Energy-Efficient Surveillance . In *ACM Transactions on Sensor Networks*, 2005.

[18] Hogthrob. http://www.hogthrob.dk/.

[19] T. Imielinski and W. L. Jr. Incomplete Information in Relational Databases. *Journal of the ACM*, pages 761–791, 1984.

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

[21] S. R. Jeffery, M. Garofalakis, and M. J. Franklin. Adaptive cleaning for rfid data streams. *Under submission to SIGMOD*, 2006.

[22] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. *SIGOPS Oper. Syst. Rev.*, 36(5):96–107, 2002.

[23] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. Subrahmanian. ProbView: A Flexible Probabilistic Database System. *ACM TODS*, 22(3):419–469, 1997.

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

[25] S. Mukhopadhyay, D. Panigrahi, and S. Dey. Data aware, low cost error correction for wireless sensor networks. In *WCNC*, 2004.

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

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

[28] L. Schenato, S. Oh, and S. Sastry. Swarm Coordination for Pursuit Evasion Games using Sensor Networks. In *Proc. of ICRA*, 2005.

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

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

[31] J. Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In *Proc. of CIDR*, 2005.