Blurring Snapshots: Temporal Inference of Missing and Uncertain Data

Vasanth Rajamani and Christine Julien
Department of Electrical and Computer Engineering
The University of Texas at Austin
{vasanthrajamani,c.julien}@mail.utexas.edu

***Abstract*—Many pervasive computing applications continuously monitor state changes in the environment by acquiring, interpreting and responding to information from sensors embedded in the environment. However, it is extremely difficult and expensive to obtain a continuous, complete, and consistent picture of a continuously evolving operating environment. One standard technique to mitigate this problem is to employ mathematical models that compute missing data from sampled observations thereby approximating a continuous and complete stream of information. However, existing models have traditionally not incorporated a notion of *temporal validity*, or the quantification of imprecision associated with inferring data values from past or future observations. In this paper, we support continuous monitoring of dynamic pervasive computing phenomena through the use of a series of snapshot queries. We define a *decay function* and a set of inference approaches to filling in missing and uncertain data in this continuous query. We evaluate the usefulness of this abstraction in its application to complex spatio-temporal pattern queries in pervasive computing networks.**

***Keywords*-sensor networks, queries, dynamics, interpolation**

I. Introduction

As applications place an increased focus on using distributed embedded networks to monitor both physical and network phenomena, it becomes necessary to support efficient and robust continuous monitoring that can communicate the uncertainty associated with data collected from a dynamic network. The emergence of pervasive computing is characterized by increased instrumentation of the physical world, including small sensing devices that allow applications to query a local area using a dynamic and distributed network for support. On the roadways, all vehicles may be equipped with devices that sense and share location, and that information can be queried by other nearby vehicles to understand traffic flow patterns. On an intelligent construction site, workers, equipment, assets, and even parts of buildings may be equipped with sensors to measure location, temperature, humidity, stress, etc., with the goal of generating meaningful pictures of the project's progress and maintaining safe working conditions.

Central to these and other applications is the ability to monitor some condition and its evolution over a period of time. On a construction site, the amount of an available material at a particular time may be useful, but it may be just as useful to monitor how that material is consumed (and resupplied) over time. Such trends are usually measured through *continuous queries* that are often registered at the remote information sources and periodically push sensed data back to the consumers [2], [10]. Such a "push" approach to continuous query processing requires maintaining a distributed data structure, which can be costly in dynamic settings. In addition, this often requires that a query issuer interact with a collector that is known in advance and reachable at any instant, which is often unreasonable. We have demonstrated that, in dynamic networks, it often makes sense to generate a continuous queries using a sequence of *snapshot queries* [19]. A snapshot query is distributed through the network at a particular point in time, takes measurements of the target phenomenon, and sends the results back to the the query issuer. In our model (Section II), a continuous query is the integration over time across a sequence of snapshot queries.

In generating a continuous and accurate reflection of an evolving environment, uncertainty is introduced in several ways [16], [17]. First, there is a significant tradeoff between the cost of generating the continuous query result and the quality of the result. For instance, the more frequently the snapshot queries execute, the more closely the continuous query reflects the ground truth, but the more expensive it is to execute in terms of communication bandwidth and battery power. In addition, the snapshot queries can be executed using different protocols that consider the same tradeoff (e.g., consider the differences in quality and cost of a query flooded to all hosts in the network and one probabilistically gossiped to some subset). On a more fundamental level, the quality of any interaction with a dynamic network is inherently affected by the unreliability of the network—packets may be dropped or corrupted, and communication links may break. The fact that a continuous query fails to sense a value at a particular instant may simply be a reflection of this inherent uncertainty.

Even when these uncertainties weaken a continuous query, applications can still benefit if the query processing can provide some knowledge about the degree of the uncertainty. For example, in a continuous query on a construction site

978-1-4244-5328-3/09/$25.00 ©2009 IEEE

for the amount of available material, it would be useful to know that, with some degree of certainty (i.e., a *confidence*) there is a given amount of available material. This may be based on information collected directly from the environment (in which case the confidence is quite high), historical trends, or knowledge about the nature of the phenomenon. Model-driven approaches that estimate missing data using mathematical models can alleviate these uncertainties [6], [7]. In these approaches, the goal is to build a model of the phenomenon being observed and to only query the network to rebuild the model when the confidence in the model has degraded to make relying on it unacceptable. Section VII examines these approaches and the relationship to our work in more detail.

Because we build a continuous query from a sequence of snapshot queries, handling uncertainty is twofold. First, we must be able to provide estimates of the continuous query result between adjacent snapshot queries. Second, even if we fail to sample a data point in a given snapshot, we may have some information about that data point at a previous time (and potentially a future time) that we may use to infer something about the missing data. In both cases, we are not actually changing the amount of information available to the application; instead we are *blurring the snapshot* queries and associating a level of confidence with inferred results.

Our approach relies on a simple abstraction called a *decay function* (Section III) that quantifies the temporal validity associated with sensing a particular phenomenon. We use this decay function as the basis for performing model-assisted inference (Section IV) to use sampled data values from the snapshot queries to infer values into the past and future. This inference can allow us to fill in gaps in the sequence of snapshot queries to enable trend analysis on the components of the continuous query. The inference and its associated confidence can also provide the application a concrete sense of what the degree of the uncertainty is. Finally, by smoothing across the available data, this inference makes the information that is available more viewable and understandable by the application and its user. We examine these benefits in Sections V and VI.

Our novel contributions are threefold. First, we introduce decay functions that allow applications to define temporal validity in a principled way. Second, we build a set of simple statistical models that allow us to effectively blur snapshot queries into continuous queries and use them to study the use of model-assisted inference for a variety of different types of dynamic phenomena. Finally, we demonstrate through an implementation and evaluation and a set of usage scenarios the efficacy and usefulness of using inference to fill in missing data in real world situations. If the network supporting data collection is highly dynamic, our approaches help mitigate the impact of the dynamics on the inherent uncertainty; however, even in less dynamic situations, our approach helps applications reasonably trade off the cost of executing continuous queries for the quality of the result.

II. Background

This paper builds on our previous approaches defining snapshot and continuous query fidelity and an associated middleware [16], [19]. These approaches approximate a continuous query using a sequence of snapshot queries evaluated over the network at discrete times. We model a dynamic pervasive computing network as a closed system of hosts, where each host has a location and data value (though a single data value may represent a collection of values). A host is represented as a triple (ι, ζ, ν), where ι is the host's identifier, ζ is its context, and ν is its data value. The context can be simply a host's location, but it can be extended to include a list of neighbors, routing tables, and other system or network information.

The global state of a network, a *configuration* (C), is a set of host tuples. Given a host $\overline{h}$ in a configuration, an *effective configuration* (E) is the projection of the configuration with respect to the hosts *reachable* from $\overline{h}$. Practically, $\overline{h}$ is a host initiating a query, and E contains the hosts expected to receive and respond to the query. To capture connectivity, we define a binary logical connectivity relation, $\mathcal{K}$, to express the ability of a host to communicate with a neighboring host. Using the values of the host triple, we can derive physical and logical connectivity relations. As one example, if the host's context, ζ, includes the host's location, we can define a physical connectivity relation based on communication range. $\mathcal{K}$ is not necessarily symmetric; in the cases that it is symmetric, $\mathcal{K}$ specifies bi-directional communication.

The environment evolves as the network changes, values change, and hosts exchange messages. We model network evolution as a state transition system where the state space is the set of possible configurations, and transitions are *configuration changes*. A single configuration change consists of one of the following: 1) a *neighbor change*: changes in hosts' states impact the connectivity relation, $\mathcal{K}$; 2) a *value change*: a single host changes its stored data value; or 3) a *message exchange*: a host sends a message that is received by one or more neighboring nodes. To refer to the connectivity relation for a particular configuration, we assign configurations subscripts (e.g., C_0, C_1, etc.) and use $\mathcal{K}_i$ to refer to the connectivity of configuration C_i. We have also extended $\mathcal{K}$ to define *query reachability*. Informally, this determines whether it was possible to deliver a one-time query to and receive a response from some host h within the sequence of configurations [18].

A snapshot query's result (ρ) is a subset of a configuration: it is a collection of host tuples that constitute responses to the query. No host in the network is represented more than once in ρ, though it is possible that a host is not represented at all (e.g., because it was never reachable from the query issuer). Depending on both the protocol used to execute the snapshot query (e.g., whether the query was flooded to all hosts in the

network or whether it was gossiped) and inherent network failures, only a subset of the reachable hosts may respond. This results in missing and uncertain data in the results of snapshot queries, which may result in a degradation in the quality of and confidence in the continuous query's result.

III. Modeling Uncertainty

Our approach to query processing allows users to pose continuous queries to an evolving network and receive a result that resembles a data stream even though it is obtained using discrete snapshot queries. This stream can then be analyzed to evaluate trends in the sensed data. However, missing and uncertain sensed items can be a bane to this process, especially in monitoring the evolution of the data. For example, on a construction site, a site supervisor may use a continuous query to monitor the total number of available bricks on the site. This query may be accomplished by associating a sensor with each pallet of bricks; the snapshot queries collect the identity of the pallets and the number of bricks the pallet holds. If consecutive snapshot queries do not sample the same subset of pallets, the sums they report are not comparable, resulting in inconsistent information supplied to the site supervisor.

Consider the continuous query in Fig. 1. The three networks on the left of the dark line show the results of the continuous query's first three snapshot queries. Each circle represents a host; a circle's color represents the host's data value; and lines represent connectivity. Throughout the continuous query, some hosts depart, some arrive, and others change their data value. In this case, the trend the application is analyzing is the data items that remain available and unchanged throughout the continuous query. When our snapshot queries are not impacted by any missing or uncertain data, the stable set the trend analysis generates is the actual stable set.

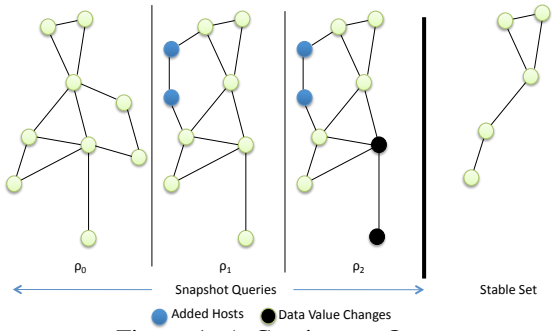

Figure 1: A Continuous Query

Consider, however, what happens when data is missing or uncertain, as depicted in Fig. 2. In this situation, the ground truth (i.e., what the snapshot queries *should* have returned) is equivalent to that shown in Fig. 1, but due to network dynamics or other sources of uncertainty, the sample from host A was not collected in the second snapshot query (ρ_1), and the sample from host B was not collected in the third snapshot query (ρ_2). Consequently the result of the trend analysis in Fig. 2 is quite different from that in Fig. 1. On a construction site, if the data items represent pallets of bricks, this trend analysis may cause the site supervisor to have additional supplies delivered when it is unnecessary or even impractical.

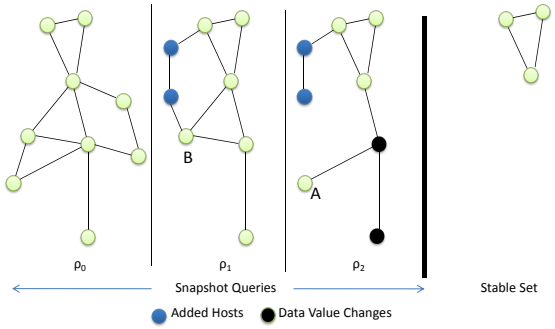

Figure 2: A Continuous Query with Missing Data

One way to handle this uncertainty is to *blur the snapshot* queries. In Fig. 2, given the fact that we know the network to be dynamic, we can say with some confidence that host A should have been represented in ρ_1; the level of this confidence depends on the temporal validity of the phenomenon sensed (i.e., how long do we *expect* a data value to remain valid), the frequency with which the snapshot queries are issued, and the degree of network dynamics. The fact that A "reappeared" in ρ_2 further increases our confidence that it may have, in fact, been present in ρ_1 as well. Fig. 3 shows a simple example of how this inference can be used to project data values into future snapshots (e.g., from ρ_1 to ρ_2) and into past snapshots (e.g., from ρ_1 to ρ_0). In this figure, the black circles represent hosts the snapshot query directly sampled; gray circles represent hosts for which data values have been inferred. The question that remains, however, is how to determine both the values that should be associated with the inferred results and the confidence we have in their correctness. We deal with the former concern in the next section; here we introduce *decay functions* to ascribe temporal validity to observations and calculate confidence in unsampled (inferred) values.

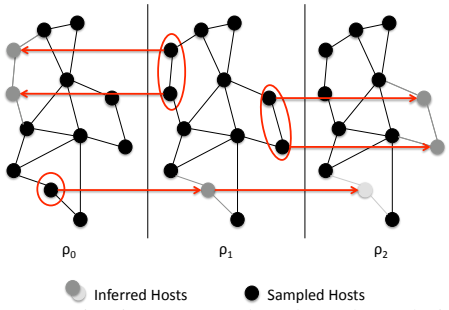

Figure 3: Projection Forward and Backwards in Time

To address temporal validity, we rely on the intuitive observation that the closer in time an inferred value is to a

sensed sample, the more likely it is to be a correct inference. For example, in Fig. 3, the value projected from ρ_0 to ρ_1 is more likely to be correct than the value projected from ρ_0 to ρ_2. If the sample missing in ρ_1 is also missing in ρ_2, it becomes increasingly likely that the host generating the sample has, in fact, departed. We exploit this observation by allowing applications to specify the temporal validity of different sensed phenomena using a decay function that defines the validity of a measured observation as a function of time.

Formally, a decay function is a function $d(t) = f(|t - t_l|)$ where t is the current time and t_l is a time from either the future or the past of the nearest (in time) actual sample of the data value. The period $|t - t_l|$ is the period of uncertainty; the larger the period of uncertainty, the less likely it is that the sampled value retains any correlation with the actual value. The decay function's value falls between 0 and 1; it is a measure of percentage likelihood. These decay functions are an intuitive representation of *confidence* and are easy for application developers to grasp. It is also straightforward to define decay functions to describe a variety of phenomena. For instance, on a construction site, a moving truck's GPS location might be associated with a decay function of the form: $d(t) = e^{-(|t-t_l|)}$, which is a rapid exponential drop in confidence over time. On the other hand a GPS mounted on a stationary sensor on the site might have a decay function of the form: $d(t) = 1$ because the location value, once measured, is not expected to change. Possibilities for formulating decay functions are numerous and depend on the nature of the phenomenon being sensed and the sensing environment.

Given a user-defined decay function, it is straightforward to determine a confidence measure of an inferred value. We measure this confidence probabilistically. At any time instant t, the inferred data value's *degree of confidence* p, is updated using the following rule.

- if time t is the time at which an actual data reading was acquired, then the value of p at time t is set to 1;
- otherwise, p is updated using the formula: $p_t = d(t)$.

Thus, at every point in time an data value of interest has an imprecision that ranges from one to zero depending on when it was last sampled. The further in time the inferred value is from an actual sensed value, the less confidence it has. With this understanding, we look next at how to estimate how a sampled value may have changed during periods where it is not sampled, allowing us to infer its value.

IV. Temporal Inference for Continuous Queries

Decay functions allow applications to define the *validity* of projecting information across time. We now address the question what the value of that projected data should be. Specifically, we present a suite of simple techniques that estimate inferred values. We also demonstrate how this inference can be combined with decay functions to associate confidence with inferred values. In later sections, we evaluate the applicability of these inference approaches to real phenomena.

A. Nearest Neighbor Inference

For some applications, data value changes may be difficult to predict, for instance when the underlying process observed is unknown or arbitrary. These changes are usually discrete; at some instant in time, the value changes to some potentially unpredictable value. Consider a construction site where pallets of bricks are distributed to different locations around the site for storage and use. A distributed query may execute across the site, measuring how many bricks are present at each location at query time. The bricks are laid and restocked during the day as trucks and construction workers perform their tasks. Without any knowledge of the project's goals and the rate of brick laying at different sites, it is difficult to create a model that effectively estimates the number of bricks at any given location for instants that have no recorded observations.

In such cases, one technique to estimate missing data is to assume the sampled value closest in time is still correct. As the temporal validity decays, the sensed value is increasingly unreliable. Consider again the pallets of bricks on a construction site and an application that samples the number of available bricks periodically (e.g., every 10 minutes). The application then sums across all of the data readings to generate a total number of bricks on the site. Fig. 4 shows an example where the value for the number of pallets at node A changes between the two samples. Up until $t = 5$, the total number of pallets is estimated using the original sample; after that, it is assumed that the value is the sample taken at $t = 10$.

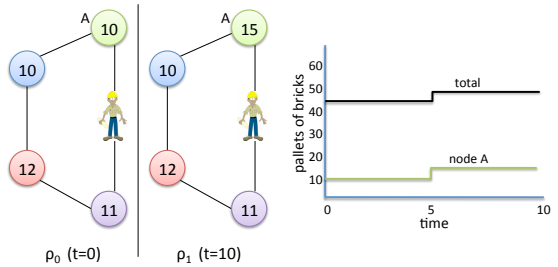

Figure 4: Nearest Neighbor Inference for Uncertain Data

The example in Fig. 4 focuses on uncertain data; i.e., inferring data values that the application did not attempt to sample. The same approach can be used to infer missing data, e.g., if the application failed to sample a value for node A at time $t = 10$ but did resample it at time $t = 20$. This example also demonstrates the importance of inferring missing data. Because this data is used to monitor the total number of pallets of bricks on the site, if data values are missing from a particular snapshot, the site supervisor might observe radical fluctuations in the number of bricks that actually did not occur.

B. Interpolation and Regression

The evolution of many pervasive computing phenomena can be fairly accurately represented by continuous functions. If a truck is driving at a steady speed across the site, and we sample its location at $t = 0$ and $t = 10$ it may be reasonable to infer that at $t = 5$, the truck was at the midpoint of a line drawn between the two sample points. In such cases, standard statistical techniques like interpolation and regression can be employed to infer data across snapshots. In interpolation, the observed values are fit on a function, where the domain is typically the time of observation and the range is the attribute's value. For any point in time where there is no recorded observation, the value is estimated using the function. Interpolation approaches range from simple (e.g., linear interpolation) to complex (e.g., spline interpolation).

Linear interpolation connects consecutive observations of a data item with a line segment. Polynomial interpolation generalizes the function to a degree higher than one; in general, one can fit a curve through n data points using a function of degree $n - 1$. Spline interpolation breaks set of data points into subsets, and applied polynomial interpolation to each subset. Fig. 5 shows an example of interpolation. The data values sensed are the locations of the devices on a 3x4 grid; the moving truck's data is missing from snapshots ρ_1 and ρ_3. The bottom figures show how linear interpolation and an example of polynomial interpolation estimate the missing data.

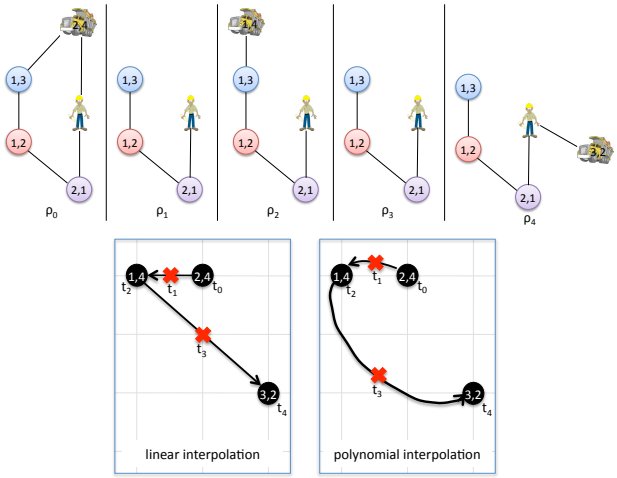

Figure 5: Interpolation of Missing Location Data

Regression identifies relationships between a dependent sensed variable (e.g., location or temperature at a particular device) and an independent variable (e.g., time). However, regression does not try to fit a curve or a function through *every* observed data point. Instead, the end result of regression encodes an approximation of the relationship between the independent and dependent variables. As with interpolation, regression comes in several flavors ranging from simple techniques like linear regression to more complex non-linear variants. Effectively, regression provides a "looser fit" function for the data; this can be effective when the underlying data is noisy (e.g., when the samples may contain errors), and it may not be useful to fit a curve through every observed data point, since those data points may not be an accurate reflection of the ground truth. Fig. 6 demonstrates the differences in nearest neighbor inferencing, interpolation, and regression pictorially.

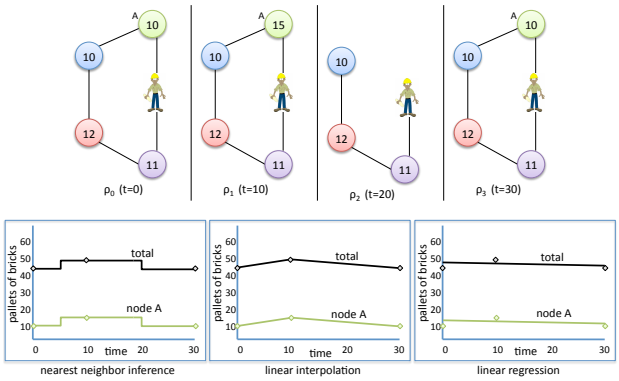

Figure 6: Comparison of Types of Inference

C. Determining Inferencing Error

When employing statistical techniques like interpolation and regression, the observed data acts as the only source of ground truth and serves as input to generate a function that estimates missing or uncertain data. To measure how well the model fits the ground truth, we choose metrics that estimate the distance between the model and reality. For regression models, a common metric is the root mean squared error, which is the difference between the actual observation and the value predicted by the regression . Similar measures of error for interpolation are difficult to define because the interpolation function is defined such that there is no error in fitting the sampled points to the function. However, as Fig. 7 demonstrates, wildly different interpolation functions (e.g., polynomials of different orders) can fit the same set of sampled points. The standard technique for determining the error of interpolation is to favor minimizing the derivative. That is, from among the interpolation functions that "fit" the data, we favor those that minimize the function's rate of change.

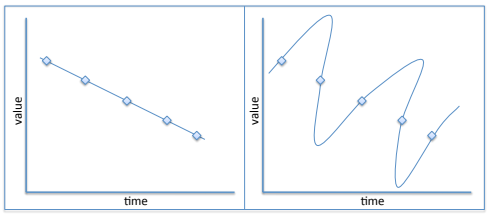

Figure 7: Interpolation Error

D. Computing Confidence from Decay and Error

Our goal is to associate with our inference of missing and uncertain data a *confidence* in the quality of that estimate

using a combination of decay functions and the error measures defined above. This confidence can impact applications and users who use sensed data to support decision making processes.

It is simple to combine nearest neighbor inference with decay functions to provide a measure of confidence in inferred values. A benefit of doing so is understanding how frequently to sample a phenomenon. If the application is inferring values between samples with low confidence, it should likely increase the frequency of sampling. If the decay function is uniform in decaying values into the past and into the future, the continuous query result the application receives is what is shown on the right of Fig. 4 coupled with a temporal validity representation of the confidence. For instance, if the associated decay function is linear in both directions, the result for the application in Fig. 4 would look something like shown in Fig. 8, where the shaded area indicates the level of confidence.

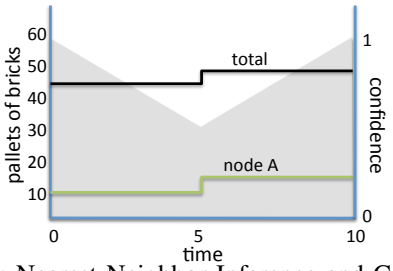

Figure 8: Nearest Neighbor Inference and Confidence

If the decay function is not uniform in its application to past and future values, the combination of that confidence with nearest neighbor inference effectively generates a new definition of "nearest." For example, if a sample is more reliable in predicting future values than it is in predicting past values, then we will rely on the past measurements for more than just half of the time difference between samples, i.e., the "step" in Figs. 4 and 8 would move further to the right.

Things are slightly more complicated for interpolation and regression. In addition to applying the decay function to the area between successful samples, we can also use information about the estimated inferencing error to strengthen or weaken our confidence in inferred values. Minimizing error like the root mean squared error of a regression or the derivative of an interpolation can increase the confidence in an inferred value. Fig. 9 demonstrates how this works for interpolation. The diamonds are sampled values; since observations can be lost or unsampled, the series may not be sampled with a fixed periodicity. The confidence plots at the bottom of the figure communicate the confidence an application has in the values inferred at times 7, 24, and 36. The gray areas show the application of the decay function from the sampled values. The lines measuring the slope of the interpolation provide a crude measure of the error associated with interpolation; the further the slope is from 0, the more rapidly the function is changing, and therefore the less likely it is to be predictable[1]. At time 24, we can generate an inferred value with high confidence, while the same is not true at time 36 or time 7.

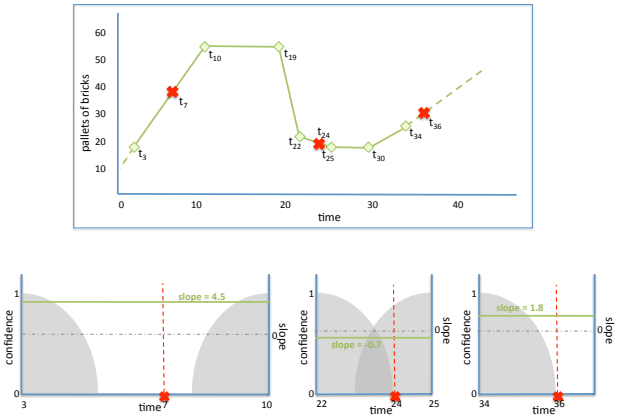

Figure 9: Associating Confidence with Inferred Values

V. Usage Scenarios

This approach to inferring missing and uncertain data applies to a wide variety of pervasive computing applications that are increasingly supported by sensor networks. In this section, we provide some example queries that benefit from the use of our temporal validity metric; we use these queries in the next section to evaluate our approach.

In the introduction, we overviewed an intelligent construction site, where people, vehicles, pieces of equipment, parts of the building, and assets are all equipped with sensors that can monitor conditions on the site and share information with applications. Within the intelligent construction site, there are many opportunities for the use of a continuous query. For example, every user on the site may carry a personal computing device that can communicate with locally embedded sensors monitoring hazardous chemical emissions. The site supervisor may continuously monitor the movement of all the cranes on the site for potentially dangerous conditions or collisions. Sensors associated with pallets of bricks can be continuously monitored to map the the positions of the assets and understand their consumption and delivery to plan the project.

Broadly speaking, given a series of snapshot queries formed into a continuous query, an application can issue two types of requests for information from the continuous query: *point* requests, for a value of the continuous query at a single point in time, and *range* requests, that monitor the continuous query over a specified period of time. Both are issued *over* a continuous query that monitors the dynamic environment.

[1]The use of slope as a representation of error here is for demonstration purposes only; the use of this type of error metric may not be entirely applicable for non-differentiable points in linear interpolation.

A point request is most useful when an application is interested in what the value of the monitored phenomenon is or was at a particular instant. These include requests for the current value (whether it was actually sampled or not), requests for historical information, and requests that project the continuous query into the future. An example for our intelligent construction site is:

Q1: *How many trucks will be available at time t?*

This request asks to look into the future of the continuous query. The response to this query can be used to plan the movement of assets in the future, especially if the answer comes with a high degree of confidence.

A range request over a continuous query specifies the time duration interest, and we use the inference approaches described previously to generate a function that represents the continuous query over that time period. This is also associated with two measures of confidence: decay functions and error metrics. An example range request is:

Q2: *How many pallets of bricks were available between t_1 and t_2?*

This requests information about discrete measurements; the number of available pallets of bricks is likely to change by large jumps as deliveries arrive and bricks are used. It may prove difficult to use a continuous function to infer missing values for this type of phenomenon; we also define a third query that more intuitively fits a continuous function:

Q3: *What was the maximum value of the concentration of a hazardous gas between time t and now?*

Requests of both types can also be used to adapt the strategy underlying the continuous query, for example to change the frequency with which the snapshot queries are issued or to change the manner in which they execute. Adaptation can change the confidence our approaches have in the quality of the inferred missing or uncertain results.

VI. Evaluation

We have prototyped our framework using OMNeT++ and the MiXiM framework [13], [14]. We implemented the queries given in the previous section and evaluate our framework's performance. In this prototype, requests are flooded through the network, and each node has a reply probability of 0.5, i.e., every sensor node responds to half of the requests it receives. We have implemented many inference approaches in our framework, but we use three of these approaches to drive our discussion: nearest neighbor inference, linear interpolation, and polynomial regression. We focused on a simple uniform linear decay function for all of these results; specifically, $d(t) = 1 - \frac{t_n - t_l}{30}$ where t_n is the current time and t_l is the time for the nearest observation. Each experiment was repeated for at least 50 runs.

In general, 45 nodes roamed in a 1000m x 900m space. A 100m "buffer" around the edge of the simulation area was considered "off-site;" the construction site itself was a rectangle of size 800m x 700m. The nodes were divided into mobility classes: 6 "truck" nodes moved at 15m/s; 10 "people" nodes moved at 5m/s, and the remaining 29 nodes were stationary. To construct a continuous view of the dynamic phenomena, we issued snapshot queries every 30 seconds for 1000 seconds; we performed inference at 5 second intervals. We generated a gas leak at a random location on the site; the leak lasted for 200 seconds, and a node's value for gas concentration was a function of its distance from the location of the leak. Bricks were located either on trucks or at designated storage locations around the site (i.e., one of the stationary sensors). Trucks moved randomly; when they left the construction site, bricks were randomly added to or removed from the truck. When trucks encountered a fixed storage location, they transferred bricks to the storage location.

A. Measuring Confidence

We first evaluate the correctness and usefulness of applying our decay functions to determine confidence in inferred responses. We executed all three queries described previously and attempted to infer missing and uncertain data for each of them using each of the three aforementioned inference strategies. Fig. 10 plots the inferencing error versus the confidence reported by our decay function; specifically the figure shows the results for applying linear interpolation to the results of **Q3**. The other combinations of queries and inference approaches had similar trends and features.

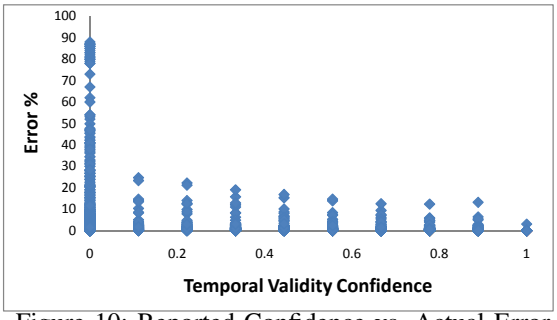

Figure 10: Reported Confidence vs. Actual Error

When our framework reports a higher confidence in an inferred value, the error of that value from the ground truth should be lower. As shown in the figure, the framework behaves as expected; regardless of the inference type and the query, the general trend is that our framework does in fact report higher confidence in values whose error (on inference) is lower. These initial experiments served simply to validate our query inferencing and decay function framework.

B. Cost Savings

Employing inference models allows applications to trade expense for error. Given that our models allow us to blur across these dynamics, we are able to query the network far

less frequently. In addition, instead of querying every node in the network, we can intentionally skip some nodes in each snapshot query, also reducing the communication overhead. We omit charts plotting the communication overhead of our approach for brevity; however, we achieve approximately a 6x reduction in communication overhead in comparison to a flooding approach that queries the network frequently enough to catch every significant change in the dynamic data (which we estimate to be every 5 seconds in our case). This reduction in communication translates directly to a reduction in energy expenditures, a significant concern in resource constrained pervasive computing networks.

C. Application Performance

We next evaluate the usefulness of blurring the snapshot queries in forming a continuous query. Consider the delivery and consumption of bricks on the construction site and **Q2** that requests the total number of bricks available over a time window. Fig. 11 plots the number of bricks sampled and inferred for a single node over time.

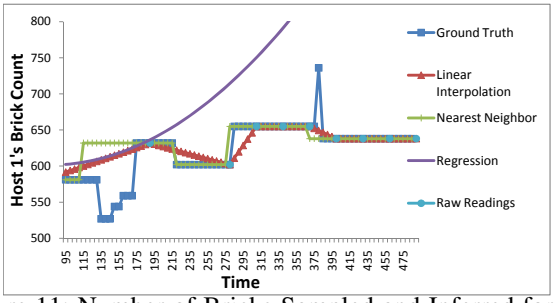

Figure 11: Number of Bricks Sampled and Inferred for One Host

This figure shows the behavior of our inferencing approaches on a single sample. Notice that regression performs quite poorly while nearest neighbor and linear interpolation mimic the real data well. This is due to the fact that the phenomenon under observation here is subject to very local and discrete data changes. Regression assumes trends that are observable over an entire time frame. Therefore more local approaches to inference perform better for this data.

Fig. 12 plots the total number of bricks, an aggregate measure that sums samples from multiple nodes. The line through the middle of the figure shows the ground truth, i.e., the total number of bricks actually on the site over time. The dots show the raw data values collected from the network every 30 seconds; these values are below the ground truth because our approach intentionally does not sample every node each time instant. The other show our three inference methods (nearest neighbor, linear interpolation, and regression, respectively). In all cases, these inference methods overestimate the total number of bricks because they are incorporating "stale" readings that were measured in previous or future samples but may not be valid at the reported time instant.

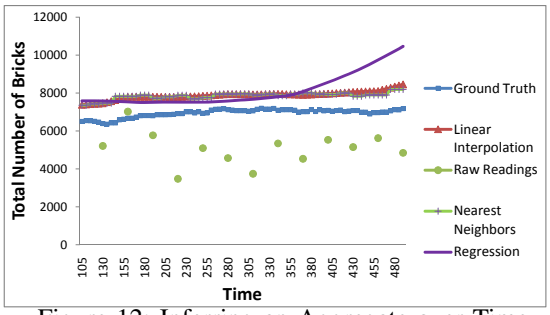

Figure 12: Inferring an Aggregate over Time

We also evaluated the use of the decay function in combination with inferencing. For **Q2** with inferencing alone, the site supervisor does not know what degree of confidence of each request response. Our framework combines this inference with a confidence measure, generating a picture such as shown in Fig. 9 for the time window of interest. To understand how well these approaches combine, we took the application aggregation query (in this case for the *sum* of bricks on the site) and evaluated the error achieved in generating this aggregate. Fig. 13 shows the results. The x-axis plots the minimum confidence required of a reading to be included in the aggregate, e.g., at $p = 0.8$, the sum was only calculated from readings with confidence greater than 0.8.

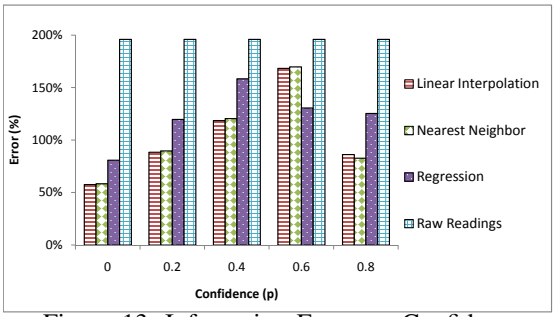

Figure 13: Inferencing Error vs. Confidence

It is clear that blurring the snapshots offers a significant benefit over relying on the available samples. No single inferencing technique performs consistently better, but they all perform better than relying on the sampled data itself. Overall, regression performs slightly worse than nearest neighbor and linear interpolation.[2] We expected nearest neighbor inference to be preferred in this situation due to the discrete data; it would likely be the application's choice due to its consistent performance and simplicity in comparison to interpolation. Another interesting trend in this figure is that the error of aggregate inferencing is not linear with confidence. In Fig. 13, a lower confidence appears to generate a lower error, which increases up to a point ($p = 0.6$) before decreasing again. This is due to the fact

[2]We tried different basis functions in support of regression; we report the best results here.

that, although the individual samples contributing to the aggregate response individually have a low confidence, the fact that there are many of them smooths out their error. In the midrange, there is a smaller number of values contributing to the aggregate, so the increasing confidence in individual data items is overshadowed by the decreasing sample size. Our framework overcomes this challenge when it reaches a high confidence; at $p = 0.8$; even though the sample size is small, the individual estimates are robust enough to lower the overall aggregate error. These observations open an important piece of future work. While our decay function provides a strong measure of confidence for *individual data estimates*, it is possible to develop a more sophisticated metric for confidence in *aggregate estimates* that combine data values to produce a combinatorial measure (e.g., a sum, average, maximum, etc.). Refining our confidence metric to account for aggregate measures is left as future work.

VII. Related Work

Our work relates to a variety of approaches from querying sensor networks to understanding uncertainty in databases. Deshpande *et al.* use model-driven querying in sensor networks [6] to construct a stochastic model of the phenomenon being observed at a base station and to acquire live data from the network only when the model proves inadequate. In addition to associating a probability density function with every entity, their model also specifies correlations between sensed attributes through conditional distributions. These models can be used to retrieve estimates of the data under observation (associated with an uncertainty measure). This differs from our approach in applicability; our use of continuous queries to monitor the underlying sensor network presupposes that the observed phenomena are significantly dynamic; in the face of such extreme dynamics, rebuilding of the local model would occur too frequently for the approach to be effective.

Kanagal *et al.* extend model-driven data acquisition using more complex dynamic probabilistic models (e.g., hidden markov models and Kalman filters) instead of simple probability density functions [11]. Similarly, particle filtering approaches enable event queries [12], [20], [24] of the form of a notification upon a condition, e.g., "Alert the user when entity X enters room A." At any instant, the location of entity X is stored as a probability distribution using particle filtering. The query is then evaluated probabilistically to provide a bound on the likelihood that X is, in fact in room A at the given time. Variants of this approach have also been used to process and manipulate streams of RFID data [5], [22]. Cheng *et al.*'s work on uncertainty associates with every mobile entity a probability density function of the entity's location [3], [4]. Then results to queries for locations of mobile entities can be associated with an estimate of the result's validity.

The above approaches construct a single, centralized model of an observed phenomenon, usually relying on an *a priori* understanding of the phenomenon observed. They execute in a centralized manner because the techniques required to generate the probability models (e.g., particle filters) are computationally intensive. In contrast, our decay models are similar but can be exercised on resource-constrained sensor nodes or personal computing devices. In addition, these pieces of existing work focus exclusively on answering instantaneous queries and associating with them an uncertainty metric; our focus is instead on continuous queries. These approaches have not been applied to queries that collect information over a period of time; it would be difficult to do efficiently using only probability density functions for single instants, and computing probability density functions for temporal queries is expensive in practice. Our approach can execute temporal queries capable of understanding trends in data, even in the presence of missing and uncertain information. Finally, these existing models also typically require an intensive learning or training phase used to establish the model's parameters; retraining the model in the face of dynamics and unpredictability is very expensive.

Our work also overlaps with systems that use statistical models for different purposes. In PAQ [23], sensors use time series forecasting built on regression models to predict local values. The application uses these models and input from the sensors about outliers to understand the observed phenomenon. Distributed regression [8] uses *kernel linear regression* in which the sensors share a general model and communicate constraints on that model that fit their data. These approaches both focus on observing phenomena with predictable degrees of low dynamics (e.g., sensing temperature reduction at night) rather than highly dynamic phenomena. MauveDB [7] provides database style interactions with uncertain sensor data, using interpolation and regression to remove irregularities. In these approaches it is unclear how often the underlying model is (or should be) updated to reflect dynamics; we have demonstrated how our approach can be used to provide such guidance. In addition, existing techniques do not support the simple temporal validity our decay function provides, which enables us to process temporal range queries.

Another project similar to ours in spirit involves the use of data-driven processing to suppress sensor network communication when data values do not deviate significantly from expected ranges [21]. This work does not rely on *a priori* models fit to the sensed data but on the live data itself. However, the focus of their work is on the use of models to distinguish between suppression and failure when employing suppression based techniques for data reduction; we instead focus on filling in the missing pieces using statistical inference. Statistical inference using decay has been explored in other domains like storage and inventory management before [9]. The notion of perishability is dis-

cussed to optimize large scale distributed systems. Concepts like perishability should be captured as first class citizens in pervasive applications and our decay model is one step towards that process.

Finally, there has been some work in pervasive computing to infer high-level patterns or behaviors from low-level data [1], [15]. These approaches use sophisticated models to represent higher level tasks that are inferred from lower level observable phenomena. In their application to date, these models must be generated for each application domain, which is infeasible in application to general sensor network inference. In addition, it is unclear how efficient these models are when performing temporal inference in a very large network.

VIII. Conclusions

Pervasive computing is increasingly supported by sensor networks that perform continuous monitoring of network or physical phenomena. However, continuous monitoring can be expensive in terms of communication and energy costs. Therefore, continuous queries in pervasive computing applications inevitably contain missing or uncertain data items. Attempting to understand trends and patterns in measuring these continuous phenomena is hindered by this inherent uncertainty. In this paper, we designed a framework that employs statistical modeling to both infer missing and uncertain data and to understand the degree of confidence an application should place in that inferred information. We demonstrated how applications can use both inference and confidence to create robust and meaningful queries of continuously monitored dynamically changing phenomena in an efficient manner. The addition of these easy to understand validity metrics provides a powerful level to understanding and employing continuous monitoring in pervasive computing applications.

Acknowledgements

This research was funded, in part, by NSF Grants # CNS-0620245 and OCI-0636299. The authors would also like to thank UT EDGE. Jamie Souvenir contributed towards the initial ideas in this work. The conclusions herein are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

[1] T. S. Barger, D. E. Brown, and M. Alwan. Health-status monitoring through analysis of behavioral patterns. *IEEE Trans. on Systems, Man, and Cybernetics*, 35(1):22–27, January 2005.

[2] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing. In *Proc. of SIGMOD*, pages 668–668, June 2003.

[3] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. of SIGMOD*, pages 551–562, June 2003.

[4] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Trans. on Knowledge and Data Engineering*, 16(9):1112–1127, September 2004.

[5] Y. Daio, B. Li, A. Liu, L. Peng, C. Sutton, T. Tran, and M. Zink. Capturing data uncertainty in high-volume stream processing. In *Proc. of CIDR*, January 2009.

[6] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. of VLDB*, pages 588–599, August/September 2004.

[7] A. Deshpande and S. Madden. MauveDB: Supporting model-based user views in database systems. In *Proc. of SIGMOD*, pages 73–84, June 2006.

[8] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed regression: An efficient framework for modeling sensor network data. In *Proc. of IPSN*, pages 1–10, April 2004.

[9] K. Hildrum, F. Douglis, J. L. Wolf, P. S. Yu, L. Fleischer, and A. Katta. Storage optimization for large-scale distributed stream-processing systems. *Trans. of Storage*, 3(4):1–28, 2008.

[10] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. on Networking*, 11(1):2–16, February 2003.

[11] B. Kanagal and A. Deshpande. Online filtering, smoothing, and probabilistic modeling of streaming data. In *Proc. of ICDE*, pages 1160–1169, April 2008.

[12] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. Challenges for event queries over markovian streams. *IEEE Internet Computing*, 12(6):30–36, November/December 2008.

[13] MiXiM. http://mixim.sourceforge.net.

[14] OMNeT++. http://www.omnetpp.org.

[15] D. J. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring high-level behavior from low-level sensors. In *Proc. of UbiComp*, pages 73–89, October 2003.

[16] J. Payton, C. Julien, G.-C. Roman, and V. Rajamani. Semantic self-assessment of query results in dynamic environments. *ACM Trans. on Software Engineering and Methodology*, 2009. (to appear).

[17] V. Rajamani, C. Julien, and J. Payton. Automated assessment of aggregate query imprecision in dynamic environments. In *Proc. of DAIS*, pages 59–72, June 2009.

[18] V. Rajamani, C. Julien, J. Payton, and G.-C. Roman. Inquiry and introspection for non-deterministic queries in mobile networks. In *Proc. FASE*, pages 401–416, March 2009.

[19] V. Rajamani, C. Julien, J. Payton, and G.-C. Roman. PAQ: Persistent query middleware for dynamic environments. In *Proc. Middleware*, 2009. (to appear).

[20] C. Ré, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams. In *Proc. of SIGMOD*, pages 715–728, June 2008.

[21] A. Silberstein, G. Puggioni, A. Gelfand, K. Munagala, and J. Yang. Suppression and failures in sensor networks: A bayesian approach. In *Proc. of VLDB*, pages 842–853, September 2007.

[22] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. Shenoy. Probabilistic inference over RFID streams in mobile environments. In *Proc. of ICDE*, pages 1096–1107, March/April 2009.

[23] D. Tulone and S. Madden. PAQ: Time series forecasting for approximate query answering in sensor networks. In *Proc. of EuroSys*, pages 21–37, January 2006.

[24] E. Welbourne, N. Khoussainova, J. Letchner, Y. Li, M. Balazinska, G. Borriello, and D. Suciu. Cascadia: A system for specifying, detecting, and managing RFID events. In *Proc. of MobiSys*, pages 281–294, June 2008.