Pervasive and Mobile Computing 8 (2012) 317–330

Contents lists available at SciVerse ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Fast track article

Using snapshot query fidelity to adapt continuous query execution

Jamie Payton[a,*], Christine Julien[b], Vasanth Rajamani[c], Gruia-Catalin Roman[d]

[a] *The University of North Carolina at Charlotte, 9201 University City Blvd, Charlotte, NC 28223, United States*
[b] *The University of Texas at Austin, 1 University Station C5000, Austin, TX 78712, United States*
[c] *Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, United States*
[d] *The University of New Mexico, 1 University of New Mexico, Albuquerque, NM 87131, United States*

A R T I C L E I N F O

Article history:
Available online 3 March 2012

Keywords:
Query fidelity
Adaptive query processing
Continuous query
Quality of information

A B S T R A C T

This paper explores the fidelity of queries issued in pervasive computing networks. A query's fidelity, or how well its results reflect the state of the environment, can be significantly impacted by dynamics that occur during its distributed execution. We focus on *continuous queries* that can be built out of sequences of consecutive *snapshot queries* and show how the fidelity of snapshots can be used to determine the fidelity of continuous queries. This simple notion of continuous query fidelity can be used to adapt query processing to impact quality and cost tradeoffs given the current state of the environment.

1. Introduction

Pervasive computing networks enable an emerging class of applications that integrate devices embedded within the physical environment with mobile devices carried by people to consume, create, and share digital information. In these highly dynamic information spaces, applications must often be context-aware, adapting their behavior to support users or automate tasks given the instantaneously perceived state of the world. The programming tasks associated with collecting information that defines an application's context can be challenging and are often simplified through the use of a software infrastructure that provides an abstraction for accessing the distributed state of the pervasive computing system. Alongside these mechanisms that facilitate access to the digital representation of the physical world, it is important to provide an understanding of the *quality* of the data used to drive decisions that impact actions to be taken by automated processes or a human user.

A *query* is an enabling abstraction that has been widely embraced for information collection in pervasive computing environments. However, the *fidelity* of a query, a measure of information quality that describes how well a query's results reflect the actual state of the environment, is significantly impacted by the unpredictability of the opportunistically formed wireless networks common in pervasive computing deployments. As a query is distributed and executed over the dynamic network substrate, changes that occur due to network and environmental dynamics can cause the query to miss potential results or to report information that is inconsistent with the actual state of the world. Understanding the fidelity of a query that is used to collect information about the surroundings is essential in pervasive computing environments; such knowledge can enhance decision processes and lead to the development of more robust and reliable systems.

Our ultimate goal is to develop a theory of the fidelity of queries issued over pervasive computing networks. We consider two forms of queries: *snapshot queries* and *continuous queries*. A snapshot query executes through the network and returns values that satisfy the query criteria at a particular instant in time. For example, in supporting a decision process on an

* Corresponding author.
E-mail addresses: payton@uncc.edu (J. Payton), c.julien@mail.utexas.edu (C. Julien), vasanth.rajamani@oracle.com (V. Rajamani), gcroman@unm.edu (G.-C. Roman).

1574-1192/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.pmcj.2012.02.005

automobile that is low on fuel, an application may query for nearby gas stations. A *continuous* query provides a result that is updated over time to reflect changes in the network of nearby embedded devices. For example, an automobile's application may continuously collect the locations and velocities of nearby vehicles to monitor safety conditions. Information about the fidelity of the query can be used to assess tradeoffs between the quality and cost of execution and, for continuous queries, to adapt the query's execution over time in order to meet an application's needs in the dynamic operational environment.

From an application's perspective, a continuous query's result should be continuously updated to reflect changes in the network. From a practical standpoint, continuously monitoring changes requires maintenance of a distributed data structure within a highly dynamic setting, which, in many network situations, can be prohibitively expensive in terms of resource consumption. In previous work, we have shown that defining a continuous query as a sequence of snapshot queries can provide a reasonable approximation of the continuous query and substantially reduces the cost of monitoring in dynamic environments [1]. We embrace this perspective here, which allows us to leverage our theory of snapshot query fidelity [2,3] as a foundation for expressing the fidelity of continuous queries. Specifically, this paper explores how the fidelity of component snapshot queries can be used to reason about an encompassing continuous query's fidelity and how we can use this relationship to adapt the execution of the continuous query in a way that considers an application's requirements with respect to the tradeoffs between query fidelity and the cost of query execution.

Consider again a continuous query issued by a moving vehicle to monitor safety conditions. Intuitively, it is important to adapt this continuous query to the changing pervasive computing environment. Choosing how to use the underlying network to answer this question in a way that finds the best balance between the quality of the query result and the cost of query processing depends on many factors, including the speed of the vehicle and surrounding vehicles, the density of the network connecting them, and the capabilities of the networking technologies. All of these facets also impact the quality of the component snapshot queries [3]. We posit in this work that by assessing the fidelity of the component snapshot queries, we can glean important information about the fidelity of the continuous query that is reflective of the changing state of the environment. This information will allow us to adapt the continuous query's behavior to balance query quality and execution cost.

In a preliminary version of this work [4], we showed how snapshot query fidelity could be used as an *introspection* method for assessing the quality of a continuous query. We constructed an introspection metric to evaluate a continuous query using snapshot query fidelity, and we demonstrated the potential of its use to adapt continuous queries through the use of an application scenario. In this paper, we extend this preliminary work with the following contributions:

- We explicitly relate the *frequency* with which snapshot queries are issued to the fidelity of the continuous query that integrates them over time.
- We assemble our original results and additional measurements to provide concrete guidance to pervasive computing application developers with respect to choosing the optimal query frequency given the tradeoff between fidelity and cost through a new *bang for your buck* metric that codifies the balance of quality and cost.
- We compare the overhead of our adaptive approach to a state-of-the-art non-adaptive approach to continuous query processing.

The remainder of this paper is organized as follows. We review our model of the pervasive computing environment, snapshot queries, and continuous queries in Section 2, and present an overview of our existing definitions of snapshot query fidelity in Section 3. The contribution of this paper, reported in Section 4, explores how we can use reasoning about the fidelity of the component snapshot queries of a continuous query to adapt a continuous query's processing to best reflect both the environment and the application's requirements and expectations.

2. Modeling the environment and queries

Understanding the execution environment is fundamental to modeling query fidelity. Therefore, we begin with a formal model of the environment, the data it contains, and the queries that interact with that data. We model a dynamic pervasive computing environment as a closed system of hosts in which each host has a location and data value (though a single data value may represent a collection of values) [2,3]. A host is represented as a tuple (ι, ζ, ν), where ι is a unique identifier, ζ is the context (e.g., host location or network information), and ν is the host's data value.

The global abstract state of such a network, a *configuration* (C), is simply a set of host tuples. To capture network connectivity, we define a binary logical connectivity relation, $\mathcal{K}$, to express the ability of one host to communicate with another. Using the values of a host triple, we can derive physical and logical connectivity relations. For example, if the host's context, ζ, includes the host's location, we can define a connectivity relation based on communication range. $\mathcal{K}$ is not necessarily a symmetric relation; if it is symmetric, $\mathcal{K}$ specifies bi-directional communication links.

We model network evolution as a state transition system where the state space contains possible configurations, and transitions are *configuration changes*. A configuration change consists of: (1) a *neighbor change*; (2) a *value change*; or (3) a *message exchange*. To refer to the connectivity of a given configuration, we give configurations subscripts (e.g., C_0, C_1, etc.); $\mathcal{K}_i$ is the connectivity for configuration *i*.

Consider the configurations shown in Fig. 1. A single snapshot query may span such a sequence starting with the configuration in which the query is issued (the *query initiation bound*, C_0) and ending when the result is delivered (the *query termination bound*, C_n). Since there is processing delay associated with issuing a query to and returning results from

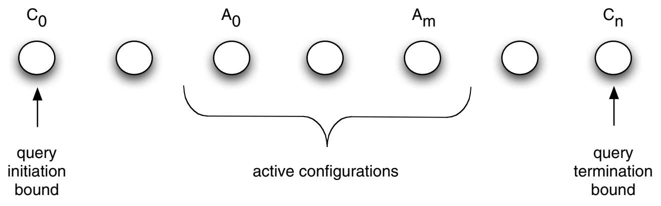


Fig. 1. Query bounds and active configurations.

the network, a snapshot query's *active configurations* are those within $\langle C_0, C_1, \ldots, C_n\rangle$ during which the query actually interacted with the network. Every component of a query's result must be a part of some active configuration.

A snapshot query can be viewed as a function from a sequence of active configurations to a set of host tuples that comprise the result. Since a configuration is simply a set of host tuples, this model lends itself directly to a straightforward representation of a query's result (ρ). In fact, the result is itself a configuration and directly correlates the result with the environment in which the query was executed, simplifying the expression of the fidelity of query results.

The ground truth of a *continuous query* is equivalent to a complete picture of the configurations ($C_0, \ldots, C_j$) over which the query is active. Since accurate evaluation of such a query is feasible only in relatively static networks, we approximate the results of a continuous query by modeling it as a sequence of non-overlapping snapshot queries, the results of which are $\rho_0, \ldots, \rho_i$. There is not necessarily a one-to-one correspondence between $C_0, \ldots, C_j$ and $\rho_0, \ldots, \rho_i$ since a single snapshot query may itself execute over multiple configurations (see Fig. 1). A continuous query's *inquiry strategy* includes the protocol used to implement the snapshot queries and the pattern of snapshot query invocations (e.g., frequency). Individual snapshot queries can be processed in many different ways, including flooding, gossiping, and location-based protocols, and these different styles of communication can result in radically different qualities of query results. The result of a continuous query usually requires *integration* over the sequence of snapshot query results. The result of a continuous query at stage i (i.e., including the results of the *ith* component snapshot query) is $\pi_i = f(\rho_0, \ldots, \rho_i)$, where f is an integration function.

A continuous query's integration strategy defines how a sequence of results together provide a result for the continuous query. An application may be concerned about the quality with which the results reflect the sensed phenomenon; given feedback on this quality, the application can adapt to network and environmental dynamics by changing its inquiry strategy to provide better results. In addition, inquiry strategies that achieve a high fidelity often do so at a high cost; applications may desire to weaken the inquiry strategy to reduce the overhead of query processing. We define an *introspection strategy* as a function over the sequence of snapshot query results that form the continuous query; the introspection strategy monitors some aspect of the quality of the continuous query. In this paper, we investigate how a concrete measure of the fidelity of the component snapshot queries can be used as an introspection strategy for a continuous query. We show that continuous query inquiry strategies that achieve a high degree of fidelity often do so at a high cost; applications may desire to weaken the inquiry strategy to reduce the cost of continuous query processing in terms of overhead. In our model of continuous query processing, an introspection strategy can trigger an *adaptation strategy* that governs changing the continuous query's inquiry strategy; in this paper, we explore how a fidelity-based introspection strategy can be used to adapt the inquiry strategy to consider an application's quality and cost requirements within the current operational environment.

3. Fidelity of snapshot queries: a review

In our previous work, we have formally defined metrics of the fidelity of snapshot queries [2,3]. This section reviews this semantic model, which lays the foundation for our use of this fidelity as an introspection metric for continuous queries.

The goal of a snapshot query is to capture the state of the world at the time that the query is issued. The choice of query protocol used to execute the snapshot query (i.e., the inquiry strategy) and the changes that occur within the environment during the snapshot query's execution can impact the query's ability to provide an accurate reflection of the ground truth in the pervasive computing environment. The *fidelity* of a snapshot query is a measure of information quality; the fidelity describes the degree to which a snapshot query's results reflect the actual state of the operational environment at the time the query is issued. To express the fidelity of a snapshot query, we formalize the relationship between the state of the network and the query's result. We assign qualitative metrics (i.e., labels) to results. Two of these fidelities match the extremes currently available in mobile computing protocols, with distributed locking protocols that guarantee that a snapshot query will execute with *atomic* fidelity and best-effort protocols that can only guarantee *weak* fidelity. The other types of fidelity lie in between these extremes and illustrate how a query's fidelity can be partially, but not completely, weakened to express the quality of a query given the dynamics of the execution environment.

Using existing query processing protocols, applications can insist on very strong semantic guarantees; if a strong semantic is not possible, the protocol aborts. In our model, a query with a strong fidelity should return only results that are collected in the same configuration. The *atomic* fidelity requires that the query was performed on a *single* configuration ($A_i(\overline{h})$) and returned a snapshot of exactly that configuration:

$$atomic \equiv \exists i : 0 \leq i \leq m \wedge \rho = A_i(\overline{h}).$$

Setting ρ *equal* to the configuration signifies not that the application uses all of the results but that they are all available, which is this fidelity's strength.

Network dynamics make it not possible to capture a perfect snapshot. The *atomic subset* fidelity captures cases when all of a query's results come from the same configuration, but the query cannot guarantee that the query returned the *entire* configuration:

$$\textit{atomic subset} \equiv \exists i : 0 \le i \le m \wedge \rho \subset A_i(\overline{h}).$$

Here, ρ is a proper subset of some active configuration.

A slightly more complex semantic provides the query issuer with a sense of what fraction of the results the query possibly missed. If the returned result represents a large sample of the possible results, the query issuer may have more confidence in subsequent usage of the result. We refer to this as the *qualified subset* semantic because the result is qualified with respect to the potential result. The formalization of the *qualified subset* fidelity specializes *atomic subset* and requires that at least α percent of the results that were available in all of the active configurations are returned.

$$\textit{qualified subset} \equiv \exists i : 0 \le i \le m \wedge \rho \subset A_i \wedge |\rho| > \alpha |A_i|$$

where $|\rho|$ is the cardinality of the set of results returned and $|A_i|$ is the total number of results that were present in the configuration of which the query is a snapshot.

Some existing protocols provide weak semantics with no guarantees. For this weak fidelity, the only assurance is that each result existed in at least one active configuration.

$$\textit{weak} \equiv \rho \subseteq \bigcup_{0}^{m} A_i(\overline{h}).$$

A slightly stronger version of *weak* fidelity is *weak qualified*. The results collected may come from across all active configurations (i.e., they may not have existed at the same time), but the query issuer is guaranteed to receive at least some specified fraction of the possible results:

$$\textit{weak qualified} \equiv \rho \subseteq \bigcup_{i=0}^{m} A_i \wedge |\textit{hosts\_in}(\rho)| > \alpha \left| \textit{hosts\_in} \left(\bigcup_{i=0}^{m} A_i \right) \right|$$

where *hosts_in* is a function that counts the number of unique hosts in a set that may contain multiple results from each host.

These definitions of snapshot query fidelity essentially fit into two classes: *atomic* (or *comparable*) and *non-atomic* (or *non-comparable*). In the first class, the stronger semantics, all of the elements returned as part of the snapshot query are guaranteed to have existed at the same time, i.e., within the same configuration. In the weaker class, all of the results of the snapshot query are guaranteed to have existed at some time during the query execution, but nothing can be said about the temporal relationships between individual items in the result.

4. Snapshot query fidelity as introspection

Query introspection is the process of determining if a continuous query's inquiry mode is suitable, given the current network and environmental conditions. This decision is often related to the tradeoff between the desired properties of the result and the cost of query execution. To support introspection, we apply adequacy metrics to query results [5]. An adequacy metric, *d*, measures the logical distance between a desired property of a continuous query and properties of the query result. For each adequacy metric, the application can define an associated threshold (δ). This simple construction supports arbitrary adequacy metrics that can enrich decision-making processes. In this section, we create an adequacy metric for introspection that describes the fidelity of the continuous query.

4.1. Defining fidelity-based introspection

We create an adequacy metric for query introspection that describes the fidelity of the continuous query using the fidelity of the component snapshot queries in an intuitive and straightforward manner. As a simple example, an application can define the fidelity of a continuous query to be the minimum fidelity over a recent window of snapshot queries. This is a conservative definition; any *non-atomic* snapshot query within the window makes the continuous query non-atomic. This conservative definition is based on the nature of integration; if one non-atomic snapshot will be integrated with atomic ones, the resulting continuous query is effectively non-atomic.

Consider again a continuous query on a highway that monitors a safety condition using the relative positions and speeds of a set of nearby vehicles. In our previous work, we defined an introspection metric based on the rate of change of information (e.g., if the identities and positions of the monitored vehicles have changed substantially from one snapshot to the next, the component snapshot queries should be issued more frequently) [5]. This metric could be combined with one that also assesses the *fidelity* of those component snapshot queries. For example, if all of the snapshot queries have atomic (or comparable) fidelity, then the relative positions of the vehicles in the result are meaningful. If they are not comparable, then they do not represent the positions of the vehicles *at the same time*, and therefore may misrepresent the potential for collisions. Therefore, the quality of the result of the continuous query that measures the potential for a collision over time is only as good as the quality of the snapshot query that measures the vehicles' positions at a given time.

We generalize the above definition of fidelity-based introspection slightly to define our query introspection metric to be the percentage of the snapshot queries in some recent history that had an atomic, or comparable, fidelity. Formally, we define the adequacy metric $d_{fidelity}(w)$ based on the fidelity of the past w snapshots:

$$d_{fidelity}(w) = \frac{\langle \textbf{sum}\, p : (k-w) < p \leq k \wedge \mathcal{S}_p \in \mathcal{A} :: 1\rangle^{1}}{w},$$

where $\mathcal{S}_i$ is the fidelity of the continuous query's *ith* snapshot, $\mathcal{A}$ is the set of atomic (comparable) fidelities (e.g., $\mathcal{A} = \{atomic, atomic\ subset, qualified\ subset\}$) described in Section 3, and k is the current snapshot. The value of $d_{fidelity}(w)$ is the percentage of the w previous snapshot query results that had an atomic fidelity semantic. If the value of $d_{fidelity}(w)$ is one, then the integrated continuous query over the previous w windows has an atomic semantic.

4.2. Demonstrating the utility of fidelity-based introspection

Intuitively and empirically, continuous query inquiry strategies that can achieve the atomic (comparable) fidelity semantics incur greater overhead. The snapshot query protocols that guarantee atomic snapshot fidelity often require increased interaction among the participants (e.g., to perform two- or three-phase locking procedures); exchanging more messages requires more resources. From empirical analysis, we also find that increasing the frequency with which the snapshot queries are issued increases the fidelity of the continuous query; such a strategy also increases the overhead of continuous query processing.

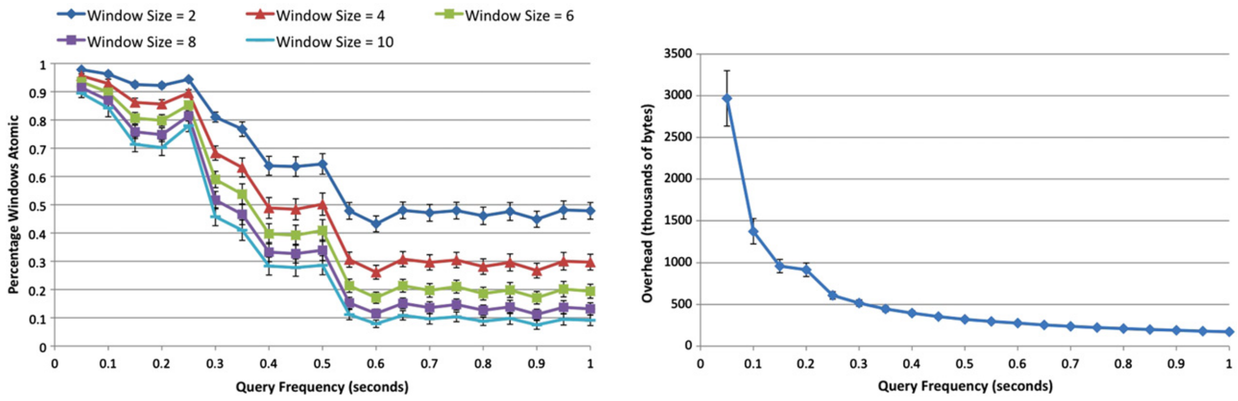
To evaluate our introspection metric and its ability to tradeoff quality and cost (in terms of overhead), we used OMNeT++ [6], its mobility framework [7], and an implementation of a two phase protocol for snapshot queries that assesses its own achieved fidelity [3] to execute a continuous query at a given frequency. The protocol uses a controlled flooding approach to distribute and evaluate the snapshot query and maintains state about the participating hosts (and their values) during these phases of query processing to assess the fidelity of a query's execution. The first phase establishes a set of query participants, which provides a reference that is used by the protocol to observe changes that impact the query's fidelity. The second phase evaluates the query. Each node that receives the query will disseminate the query to its children and wait for their responses before responding with its own local data value and relevant state information. We use our fidelity-based adequacy metric defined above to drive adaptation of the frequency of the continuous query's component snapshot queries.

The results in this section are generated by executing snapshot queries over a simulated network consisting of varying numbers of nodes at varying speeds within a 1000×900 m^2 area. A single node is designated as the query issuer; this node issues consecutive snapshot queries using the aforementioned two-phase protocol at a given frequency over a duration of 200 s. Flooding is controlled by setting the time to live (TTL) parameter for the query message; our simulations use a TTL value of 3. In each result, we specify a maximum speed; node speeds are chosen uniformly and randomly between 0 m/s and the specified maximum. The nodes move according to random waypoint mobility [8] with a zero pause time, in which each node is initially placed randomly, chooses a random destination, and moves in the direction of the destination at its assigned speed. When it reaches that destination, it immediately chooses a new destination and repeats the process. We used the 802.11 MAC protocol implemented in the mobility framework; when possible we show 95% confidence intervals.

Fig. 2 provides a pair of measurements that together characterize how a continuous query's inquiry strategy is related to fidelity-based introspection. Fig. 2(a) shows the percentage of atomic windows for varying query frequencies, i.e., the y-axis plots $d_{fidelity}(w)$ for $w = \{2, 4, 6, 8, 10\}$. It is immediately obvious that, regardless of the window size used for introspection, issuing the component snapshot queries more frequently (the left hand side of the figure) achieves a much higher fidelity. This is a result of the fact that both the network topology and the state of the data stored in the network are very dynamic, and as queries are issued less frequently, the more likely it is that this state (and the network's ability to execute a snapshot query with atomic fidelity) has changed from one snapshot query to the next. A more subtle fact in this figure is that even when queries are issued at a very high frequency, our very strong two-phase commit-based protocol cannot achieve perfect atomicity given the dynamics of the environment (e.g., even for a window size of 2 and queries issued 10 times per second, the percentage of the windows observed that were atomic topped out around 95%).

Fig. 2(b) shows the average overhead of query processing as a function of frequency. Clearly (and matching our intuition), decreasing the frequency of issuing snapshot queries decreases the cost associated with processing that query. The two plots in Fig. 2 taken together give us important insights on the relationship between the query frequency and the resulting fidelity of the continuous query, which can help both application developers and adaptive applications to make decisions about the continuous query's execution in order to meet quality and cost requirements in a given environment.

[1] In the three-part notation: $\langle \mathrm{op}\, quantified\_variables : range :: expression\rangle$, the variables from *quantified_variables* take on all possible values permitted by *range*. Each instantiation of the variables is substituted in *expression*, producing a multiset of values to which op is applied. If no instantiation of the variables satisfies *range*, the value of the three-part expression is the identity element for op, e.g., *true* if op is $\forall$.



(a) Percentage of atomic windows for varying query frequencies. (b) Overhead of protocol execution (in bytes).

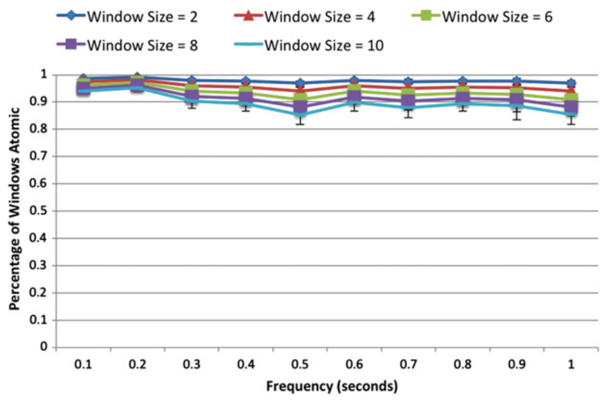
Fig. 2. Impact of query frequency on fidelity (max speed = 15 m/s, ttl = 3 hops, hosts = 50).

To further understand these tradeoffs in different styles of networks, we next quantify the relationships between fidelity, overhead, and network characteristics. In situations where the developer knows something about these essential characteristics of the network environment *a priori* and does not expect drastic changes that impact this characterization of the network, this insight can be used to help choose a frequency that achieves the best tradeoff between fidelity and cost in the target environment. Fig. 3 illustrates the link between fidelity, cost, and query frequency for various network characteristics. The charts in Fig. 3 show the results of executing continuous queries over dynamic networks with varying degrees of node density (from sparse networks with an average of three neighbors per node, through decently connected networks with an average of 10 neighbors per node, to dense networks, with an average of 26 neighbors per node). As the evaluation shows, when the density of the network is known to be low (Fig. 3(a) and (b)), the tradeoff between quality and cost is slight; the developer can choose a low frequency, which still has a very high quality, to get the lowest possible cost. The same is not true as the density of the network increases; as Fig. 3(e) and (f) show, in dense networks, the developer has to think very critically about the tradeoff between cost and quality. High quality queries are possible, but they incur a very significant cost; on the other hand, decreasing the query frequency even slightly rapidly degrades the quality, indicating that the middle values for query frequency in dense networks are not all that useful. Networks with average density, however, offer much more complex tradeoffs for the developer, and the information in Fig. 3(c) and (d) show this complexity; there is clearly middle ground where the queries can achieve decent quality at a reduced cost; the best choice of protocol settings in these cases will depend on the developer's knowledge of the application requirements. These results hold for our specific definition of query quality as given by our fidelity-based adequacy metrics; other adequacy metrics would engender different relationships that would require similar elicitation to enable guidance for application developers and deployers.

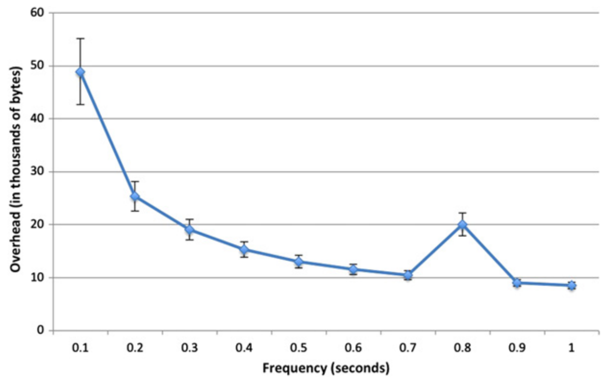
We further elicit the relationships between quality and cost as given by our fidelity-based adequacy metric through a new "*bang for your buck*" metric, defined by measuring the amount of atomicity a sampling approach achieves per unit of overhead. We compute this metric relative to the best possible quality our sampling approach can achieve; i.e., that achieved with a query frequency of 0.1. In Fig. 4, which plots this metric for different network densities, data points above the value of 1 "beat" the best possible quality (a sample frequency of 0.1) with respect to this bang for your buck metric. Fig. 4 shows a very interesting result that demonstrates a significant utility to developers who can quantify the nature of their deployment environments. In our example situations, when the density of the network is very low, the best bang for your buck turns out to be to sample as infrequently as possible. This is because, as shown in Fig. 3(a), the query fidelity is very high, regardless of the query frequency. This observation remains true when the average number of neighbors is 10 and 14, though the improvement in the bang for your buck as the query frequency decreases also decreases (as demonstrated by the flattening out of the lines in Fig. 4 for densities of 10 and 14. Most interestingly, though, for an average density of 26 neighbors, the most bang for your buck in this example is actually at a frequency of 0.2 (which is slightly better than the baseline of 0.1). This means that the quality of the query in these dense networks is so poor at the lower frequencies that the extra overhead of frequent snapshots is worth it (assuming the value is measured in terms of our simple bang for your buck metric).

This may be an oversimplification of the inherent tradeoffs of fidelity and cost. There may be fidelity thresholds below which an application cannot operate, regardless of how inexpensive they are (in terms of overhead of query processing). However, this bang for your buck metric clearly quantifies that tradeoff, which is valuable information for a developer or deployer.

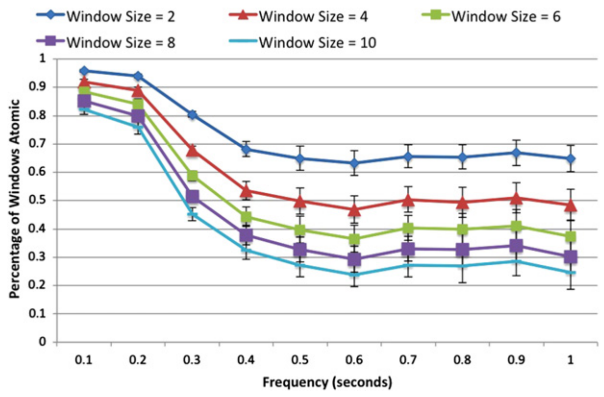
When network characteristics (e.g., density) are expected to remain in a steady state over the course of an application's lifetime, the results shown in Fig. 3 can be used to select the most appropriate query frequency to meet an application's needs in terms of fidelity and cost. However, it is frequently the case that these network characteristics change over time. Fig. 5 gives insight into the impact of environmental characteristics that change during the course of an application's lifetime on the fidelity of a continuous query. Specifically, Fig. 5 explores a single aspect of the dynamic environment: node speed.



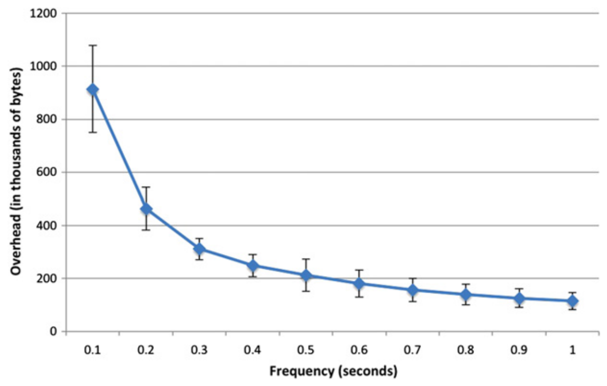
(a) Percentage of atomic windows for varying query frequencies for sparse networks (10 hosts; average num neighbors = 3).



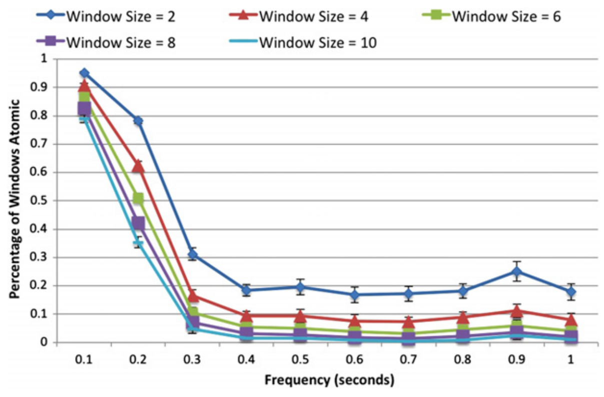
(b) Overhead of protocol execution (in bytes) in sparse networks (10 hosts; average num neighbors = 3).



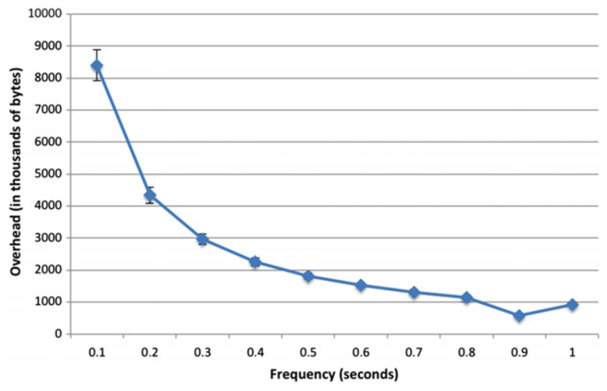
(c) Percentage of atomic windows for varying query frequencies for medium density networks (35 hosts; average num neighbors = 10).



(d) Overhead of protocol execution (in bytes) in medium density networks (35 hosts; average num neighbors = 10).



(e) Percentage of atomic windows for varying query frequencies for dense networks (100 hosts; average num neighbors = 26).



(f) Overhead of protocol execution (in bytes) in dense networks (100 hosts; average num neighbors = 26).

Fig. 3. Impact of query frequency on fidelity (max node speed = 10 m/s, query ttl = 3 hops).

As expected, fidelity-based introspection does provide a reflection of the degree of dynamics in the environment. Specifically, it is more difficult for our continuous query to provide an atomic measure of the environment as that environment grows more dynamic.

In situations where the essential nature of the environment rapidly changes, the execution of the continuous query should adapt in response to meet an application's needs with respect to fidelity and cost. When a continuous query includes a snapshot with non-atomic results, either the inquiry strategy should be "strengthened" to generate more atomic snapshots that are integratable or "weakened" to reduce overhead. These decisions depend on application requirements; applications may need to specify tolerance for non-atomic results and for acceptable inquiry strategies (e.g., the frequency with which the continuous query must be updated, regardless of fidelity). In the following sections, we apply these insights to show

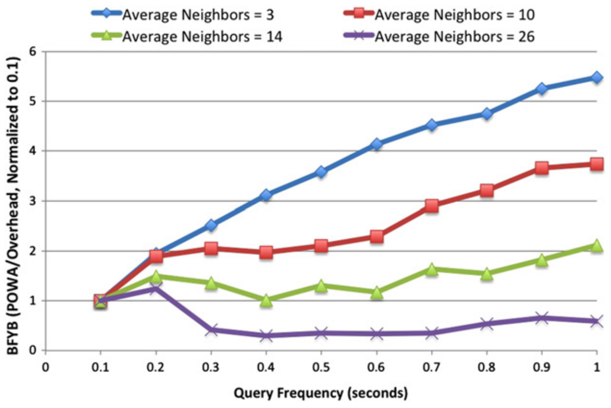


Fig. 4. *Bang for your buck* (max node speed = 10 m/s, query ttl = 3 hops, window size = 5).

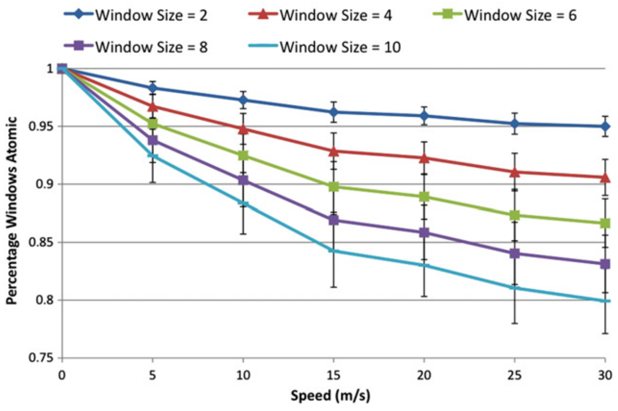


Fig. 5. Impact of node speed on fidelity-based introspection (query freq = 100 ms, query ttl = 3 hops, num hosts = 50).

how one can create an adaptation strategy that uses our fidelity-based introspection metric to adapt a continuous query's behavior to meet quality and cost requirements within a dynamic environment.

5. Fidelity-based adaptation

The previous section provides and demonstrates the capabilities of a new metric for measuring the *quality* of a continuous query. In our system model, this provides an *introspection* strategy by which an application can monitor the query. If the result's quality does not meet the application's requirements, the application can specify *adaptation* strategies that can change the query.

5.1. Defining adaptation strategies

Using our fidelity-based introspection metric defined above, the application can dynamically adapt its inquiry strategy to meet quality and cost requirements. The following is a generic and simple structure for defining adaptation strategies:

$$\langle\, \langle \mathcal{I}, \mathit{freq} \rangle, d, \delta^{+/-}, \langle \mathcal{I}^*, \mathit{freq}^* \rangle\, \rangle,$$

where $\langle \mathcal{I}, \mathit{freq} \rangle$ is the continuous query's current inquiry strategy, d is the introspection strategy, δ is the introspection threshold, and $\langle \mathcal{I}^*, \mathit{freq}^* \rangle$ is the new inquiry strategy that should be applied if the value of the introspection metric violates δ. $\mathcal{I}$ is the protocol used to process the snapshot query, while *freq* is the frequency of the snapshot queries. If the superscript of δ is $+$, adaptation is triggered if the value of d exceeds δ; if the superscript is $-$, adaptation is triggered if the value of d falls below δ.

For fidelity-based adaptation, the threshold δ associated with $d_{\mathit{fidelity}}(w)$ indicates that if the percentage of atomic snapshot queries in a window of size w falls below the threshold, the continuous query's inquiry strategy should be adapted, either to increase the quality of the continuous query or to reduce its overhead. Adaptation strategies are highly application dependent. With respect to the fidelity of a continuous query, some applications may only be able to function if the continuous query is atomic. Best effort applications may desire to focus exclusively on saving overhead. We define a sample set of adaptation strategies that straddles these tradeoffs. For simplicity, we look only at adapting the snapshot query frequency (i.e., $\mathcal{I}$ will not change), though one could also adapt the query protocol itself. Consider an application that desires to achieve atomic fidelity if possible but is not willing to incur more overhead than is associated with issuing the snapshot queries every 100 ms. If the application cannot achieve atomic fidelity, even with a sampling rate of 100 ms, it desires to reduce the sampling rate to 1 s to reduce overhead. The following two adaptation rules achieve this tradeoff ($\mathcal{I}$ designates the single available query processing protocol and w is an application-provided window size):

$$\langle\, \langle \mathcal{I}, >\ 100\text{ ms}\rangle, d_{fidelity}(w),\, 1^{-},\, \langle \mathcal{I}, freq - 100\text{ ms}\rangle\, \rangle$$
$$\langle\, \langle \mathcal{I}, \leq 100\text{ ms}\rangle, d_{fidelity}(w),\, 1^{-},\, \langle \mathcal{I}, 1\text{ s}\rangle\, \rangle.$$

These rules emphasize fidelity-based introspection; different applications will need different functions, and applications' actual adaptations are likely to be complex. There are two limitations of the above rules. First, using a threshold of 1 communicates that the window must be entirely atomic or adaptation will be triggered. Given the pervasive computing environment dynamics, there will likely be intermittent non-atomic queries, even when the environment is largely amenable to atomicity. Therefore, applications that can tolerate a small degradation in fidelity can set δ to be less than one, e.g., for a window size of 10, setting δ to 0.8 allows two snapshot queries in a window to be non-atomic, but three non-atomic snapshots triggers adaptation. The second limitation is that it will constantly attempt to achieve atomic snapshots, even in environments that are never amenable to them. If the adaptation process reaches a query frequency of 100 ms without satisfying the threshold for atomicity, the query frequency adapts back to 1 s, which will begin triggering the first adaptation strategy again. Automatically detecting these cyclical dependencies in reactive rules like our adaptation strategy is a research challenge in itself [9]. A straightforward fix would be to add some hysteresis, effectively disabling the first adaptation rule until some number of snapshots after the second rule has been executed (e.g., based on a multiplier of w); in fact, in general, it is useful to place an intentional delay between adaptation steps to minimize oscillations in competing adaptation rules.

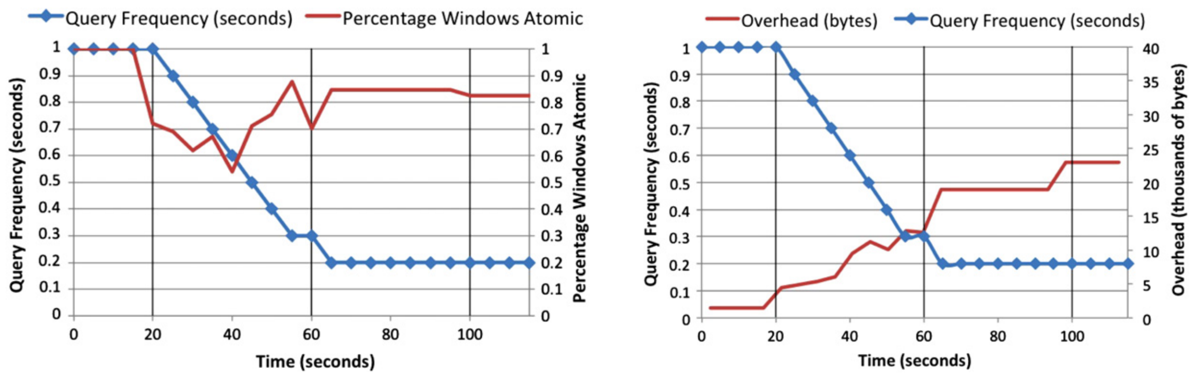
5.2. Continuous query fidelity-based adaptation scenario

We next show how fidelity-based introspection can be used to adapt a continuous query to provide higher quality results or reduce the overhead using an application scenario in which we apply fidelity-based adaptation of continuous query processing. We also show how our adaptive approach compares to an existing non-adaptive query processing protocol in terms of overhead.

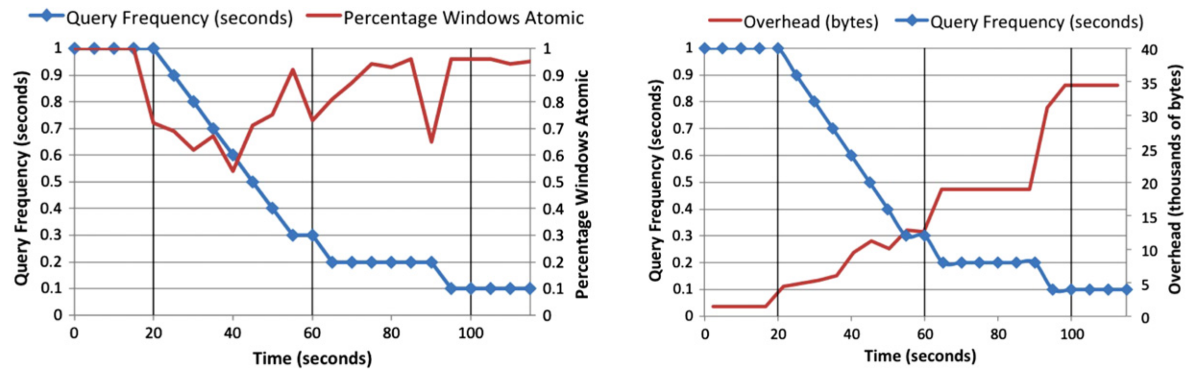
Consider an application on a car executing a continuous query to monitor other nearby cars to maintain safety conditions. Initially, the car is stationary. As the trip progresses (e.g., the car navigates the parking lot, then a side street, then a busy street), the car gradually picks up speed, triggering query adaptation. We execute a continuous query that simply collects the locations of nearby cars for 115 s (we chose the short time scale simply to demonstrate adaptation). The maximum speed of the nodes increases to 5 m/s after 20 s; to 10 m/s after 60 s, and to 15 m/s after 100 s. We use our definition of $d_{fidelity}(w)$ with $w = 5$ and our two adaptation rules from Section 5, with the threshold δ set to 0.8. Fig. 6(a) shows the *expected* (theoretical) values for the atomicity of the continuous query and its processing overhead over time for this scenario. These are values the developer can estimate for his adaptation strategy in advance of deployment to determine (at least, theoretically), what the potential merits and demerits of his strategy are. Fig. 6(b) shows what happens for a live adaptive query that adapts its query frequency over time; the figure shows both the percentage of windows that are atomic over time and the overhead of querying. As Fig. 6 shows, the continuous query rapidly adapts its query frequency to the changing conditions, which enables the continuous query to maintain a high level of fidelity. In this simple scenario, the actual behavior matches the expected behavior fairly well; there are a few exceptions that result from unexpected conditions in the mobile network; this is to be expected of our highly dynamic and unpredictable environments.

Within the context of this simple scenario, we now compare our approach for fidelity-based adaptive continuous query processing to an existing non-adaptive approach to continuous query execution. Fig. 7 compares the cost of execution (in terms of message overhead) for a non-adaptive query, issued as a sequence of best-effort snapshot queries, to the cost of our adaptive continuous query, issued as a sequence of two-phase snapshot queries that self-assess and report their fidelity. Each best effort query is implemented as a controlled flood of the network, with a single phase to request and return values. As expected, a non-adaptive continuous query that is constructed from infrequently issued best-effort queries (e.g., every 1 s) will have a lower average cost over the lifetime of the application than the adaptive approach, which attempts to find an application-specified balance between overhead and quality. By looking at the query frequency reported for the adaptive query in Fig. 6(b) in conjunction with the overhead results reported in Fig. 7, however, we see that when snapshots that comprise the adaptive query are issued at the same frequency as the non-adaptive query in a similar environment (here, characterized by the speed of nodes at a given time), these two approaches have approximately the same cost in terms of overhead. For approximately the same cost, our approach provides the advantage of capturing a measure of the quality of the continuous query's execution, whereas a non-adaptive continuous query that issues best-effort queries makes no guarantee or report about the quality of its execution.

The initial set of adaptation rules for the scenario described above adjust the query frequency to compensate for decreased atomicity. A more sophisticated set could weaken the continuous query to test whether the network conditions have



(a) Theoretical adaptation.



(b) Actual adaptation.

Fig. 6. Simple continuous query adaptation (query ttl = 3 hops, number of hosts = 50, max speed starts at 0 m/s, goes to 5 m/s at 20 s, to 10 m/s at 60 s, to 15 m/s at 100 s).

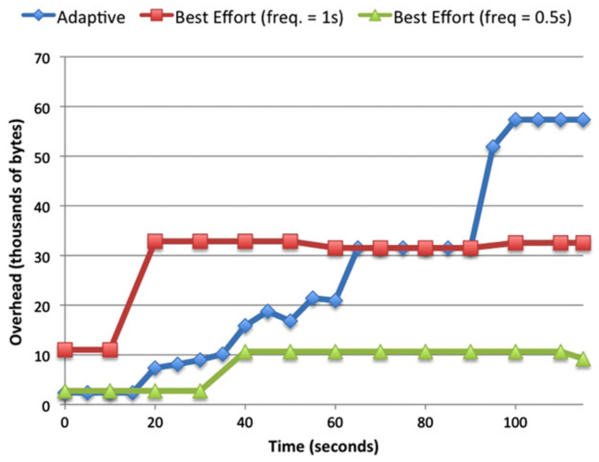
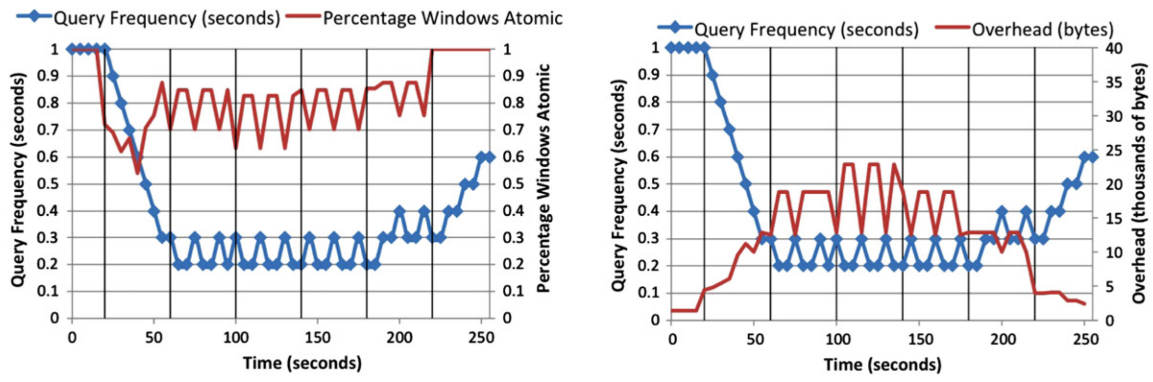
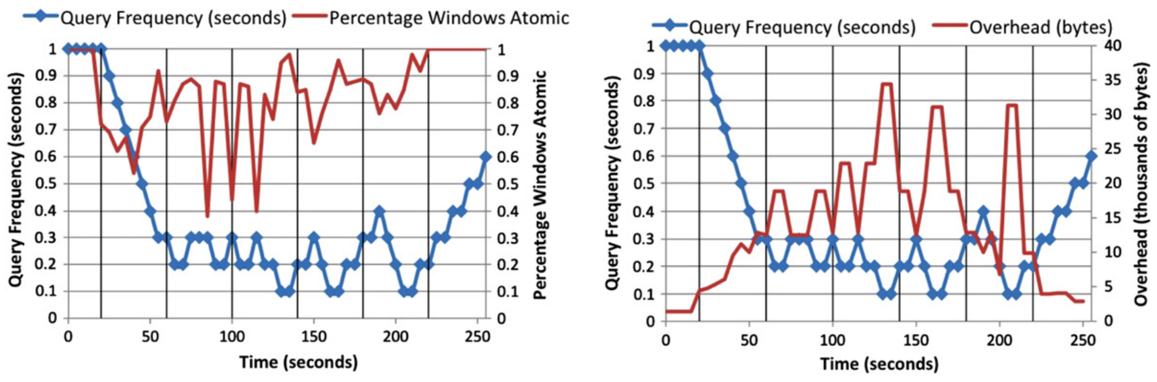


Fig. 7. Comparison of overhead for adaptive and non-adaptive continuous queries (query ttl = 3 hops, number of hosts = 50, max speed starts at 0 m/s, goes to 5 m/s at 20 s, to 10 m/s at 60 s, to 15 m/s at 100 s).

changed such that an inquiry strategy with lower overhead may still achieve the query's atomicity goals. Our second scenario begins like the first, but after 140 s, the maximum speed decreases to 10 m/s, after 180 s it decreases to 5 m/s, and after 220 s, it decreases back to 0 m/s. We augmented our adaptation rules to also periodically test a lower query frequency to see if the atomicity threshold would be violated. Specifically, after adequately maintaining the atomicity requirement for at least 10 s at a given frequency, we decrease the query frequency by 100 ms. This provides a third adaptation rule to the set listed in Section 5. The results for this scenario are shown in Fig. 8, which provides the adapted query frequency and the resulting fidelities and overheads. Fig. 8(a) gives the theoretical results. As shown, given this dynamic scenario, we can



(a) Theoretical adaptation.



(b) Actual adaptation.

Fig. 8. A smarter continuous query adaptation (query ttl = 3 hops, number of hosts = 50, max speed starts at 0 m/s, goes to 5 m/s at 20 s, to 10 m/s at 60 s, to 15 m/s at 100 s, followed by a 5 m/s decrease in speed every 40 s).

expect this adaptation strategy to keep the atomicity level around 80% (where the threshold for adaptation is set), trading off overhead for atomicity. The valleys in the theoretical results come from the adaptation strategy periodically testing lower query frequencies to determine if they will satisfy the atomicity requirement; when they do not, the adaptation strategy returns the inquiry strategy to the higher frequency. Fig. 8(b) shows what actually happens for a trace using this adaptation scenario in the real environment. As the figure shows, this strategy still does a good job of maintaining the atomicity requirement, though the results do not follow the expected ones directly. Notice in particular that, especially as the average speed of the automobiles is increasing (in the first part of the experiment), the adaptation strategy's ability to model the expected atomicity level is lower. This is a result of the unpredictabilities and inconsistencies that arise in mobile environments that have significant potential impact on continuous queries. More importantly, however, Fig. 8 shows that, using an adaptive strategy, our continuous query maintains low overhead when possible, and incurs high overhead only when necessary to maintain the application's desired atomicity.

6. Related work

This paper presents an approach to capturing the *fidelity* of a continuous query, which expresses a measure of the quality of information retrieved by the query. Quality of Information (QoI) has been widely studied for traditional distributed systems (e.g., [10–15,13]). Emerging work on QoI for sensor networks focuses on the quality of data provided by sensors to an application; proposed measures include the accuracy of a sample [16,17], the freshness of data [18], and spatiotemporal relevance [19]. While many of the QoI metrics for sensor networks focus on the quality of the individual data items that are collected, our approach focuses on the *fidelity of a query*, which describes the quality of a query's execution process. Fidelity describes how well the results collected by a query reflect the ground truth of the environment and indicates whether or not the data values collected actually existed at the same point in time, which is an important consideration when applying a function over the raw collected data.

Researchers have identified the need to create a framework to promote quality-aware data collection in sensor networks [20,21]; these frameworks embrace a multi-dimensional approach to defining QoI in such settings. The query fidelity metric presented here represents one facet of the quality of information associated with query processing, which can fit into such a framework. The use of an expressive model for query processing (presented in Section 2) as the foundation for our definition of continuous query fidelity makes it possible to express and apply fidelity and other QoI metrics in a complementary way. For example, since the host tuples in the foundational model capture context information about the host or individual data items, it is possible to use the model to capture QoI metrics that describe spatial and temporal properties of the individual data items or a collection of data items returned by a query.

The concept of query fidelity is closely related to the notion of consistency in distributed databases [22]. Work in this area has explored maintaining consistency with dynamic cache allocation [23] or optimistic replication [24], while other approaches relax the constraints imposed by the ACID properties (atomicity, consistency, isolation, and durability) and execute queries using transactions that adhere to a weaker set of properties [25–30]. These models are generally limited to use in settings where periodic access to the wired infrastructure is available, and solutions tend to rely on the use of a resource-rich fixed node to manage transactions. Because of this, such weakened transactional models cannot be applied to the more extreme form of mobility found in the mobile ad hoc networks our work targets. Our approach differs significantly in that we avoid caching data locally, instead acquiring it on-demand by executing a distributed query processing protocol over a dynamic environment. We also postpone the decision of how weak an operation to perform until run time, providing applications with the strongest semantic achievable in a given operational context. Rather than select a single snapshot query protocol that offers a particular guarantee, we instead use a flexible, self-assessing snapshot query protocol that provides the strongest semantic achievable in a given operational context and reports the achieved semantic [2,3]. We build a continuous query out of these snapshot queries, and we use a fidelity-based introspection metric to adapt execution of the continuous query to meet an application's needs in terms of fidelity and cost of query execution.

Similar to query fidelity, data fidelity has been expressed in terms of precise metrics defining numerical error and order error, for example, exploring the design space between strong consistency and no consistency for data access in replicated file systems [31]. We focus on non-replicated data items and aim to provide a combination of qualitative and quantitative measures for query fidelity of the collection of that distributed data. Similarly, *completeness* describes the probability that a particular datum has been included in a result [32]. While similar to our goal, this approach has been applied only to a distributed shared memory system without consideration of the challenges of mobility. Closely related to our definitions of snapshot query fidelity is a new class of semantics based on "single site validity", in which a snapshot query result appears to be equivalent to an atomic execution from the query issuer's perspective [33]. That work defines a particular class of semantics, while we provide a model that can define and compare multiple classes of semantics across both snapshot and continuous queries. Others have since expressed inconsistency in query results through network imprecision [34], monitoring the network for numerical imprecision and reporting it with the query result. Our approach to assessing query fidelity is similar, but we extend the concept of network monitoring to a dynamic, unreliable, and unpredictable network. In addition, we allow applications to use a low-cost query and discover that it achieves high fidelity when the network is able. Other times, the applications may need to increase the cost of continuous query processing to achieve the same fidelity due to instability. By reflecting on the fidelity of previous snapshot query results, applications can use fidelity-based introspection and adaptation to dynamically adjust the continuous query's execution.

Others have recognized the need to adapt a query's processing in response to the application's changing environment [35]. In adapting query processing, the focus is typically to change the order of query operations to optimize for dynamics. For example, Continuous Queries (CACQ) [36] rely on eddies [37] to determine the order in which partial query results are processed. StreaMon [38] adapts a query plan to accommodate arbitrary changes in the data stream. These approaches use system-defined (instead of application-defined) adaptation. In model-driven approaches [39], a local model of the environment is used to answer queries. The model obtains data from the network only when it cannot answer a query. Adaptive filters [40] use a model of the network to adjust the rate of updates that stream from each node in the network to a collector as part of a continuous query; the adjustment is based on acceptable tradeoffs between an application's tolerance of numerical imprecision and the current cost of sending updates. Such model-based approaches are not well-suited for dynamic environments because unpredictability counteracts the temporal correlations that form the basis for the models.

None of these approaches to adaptive query processing provide general support for dynamically adapting a continuous query based on application-specified strategies, which is important because the application is explicitly trading fidelity for cost [1]. Such *reflection* is common in mobile computing and middleware models [41,42]; our work recognizes the importance of reflection to the adaptivity of mobile applications and provides a formal foundation for exposing information about query results to applications through a set of principled adaptation mechanisms. Other related approaches have looked at query quality for web-based queries, which has largely focused on how to filter query results into those that are likely to be of high quality to the user [43] or to allow users to specify control parameters with web-based queries that dictate their resolution [44].

7. Conclusions

In this paper, we have demonstrated an approach to defining the fidelity of continuous queries, which are approximated as a sequence of snapshot queries, issued over dynamic pervasive computing environments. In previous work, we demonstrated how to associate a semantic tag with a snapshot query's results that describes the snapshot query's fidelity. Here, we start with this notion of snapshot query fidelity as a foundation and demonstrate how functions defined over these semantic tags can be used to define the fidelity of a continuous query. Our empirical analysis highlights the relationship between this new notion of continuous fidelity and the frequency with which component snapshot queries are issued and provides guidance to application developers on selecting the query frequency that achieves quality and cost tradeoffs. We also have shown how this definition of continuous query fidelity can be used to adapt the continuous query's execution. With respect to our goals, this adaptation allows a continuous query to use information about its history of execution and change its operation to, for example, increase the fidelity (usually at a higher cost) or reduce the cost (usually at a loss of fidelity). We have applied this approach to a simple application example; our evaluation shows that this approach can provide a desirable degree of fidelity, where a non-adaptive best-effort protocol fails to do so, and will do so with less overhead than a non-adaptive protocol that guarantees atomic fidelity.

The "percentage of windows atomic" (POWA) measure of continuous query fidelity presented here represents a single dimension of the quality of information associated with query processing in pervasive computing networks and provides a starting point for further exploration of continuous query fidelity. It is not the only (and possibly not even the best) metric for continuous query fidelity; it can certainly be used incorrectly or incompletely by applications to assess the fidelity of their continuous queries. For example, if applications issue component snapshot queries only very infrequently, they may all achieve atomic snapshot query fidelity, making them appear integratable into an atomic continuous query; however, the integration may not, in fact, be a good representation of the targeted trends in the underlying pervasive computing network. Additional metrics for capturing continuous query fidelity may be needed to express other properties of importance when using the integration of snapshot query results within a continuous query to guide decision-making processes. For example, such a metric may express the likelihood that an important change in value or network topology was not captured between two successively issued snapshot queries. We will continue to explore the development of new kinds of continuous query fidelity definitions, and the application of such metrics to adapt continuous query processing.

Finally, the adaptation approach explored here is admittedly simple; this paper illustrates the use of adaptation rules that simply adjust the frequency with which a continuous query's component snapshot queries are issued in order to satisfy an application's requirements in terms of query fidelity and associated communication costs. More sophisticated approaches to adapting the continuous query in a way that considers an application's requirements with respect to information quality and the associated costs of data collection are certainly possible. One key insight is that the choice of the query protocol used to execute component snapshot queries over the network can have a significant impact on these concerns. For example, an application may employ a snapshot query protocol that floods the entire network if communication overhead is not a concern, while an application that seeks to maximize the lifetime of the network may employ a snapshot query protocol that randomly samples a subset of network nodes. The suitability of the selected snapshot query protocol may change as the dynamics of the network change. In this paper, we have used a two-phase *flooding* snapshot query protocol [3] that is capable of dynamically assessing its achieved fidelity to construct an adaptive continuous query. To apply the fidelity-based adaptation approach introduced in this paper to other kinds of snapshot query protocols will require the creation of new methods to dynamically assess the fidelity achieved by the execution of those protocols and an analysis of how the dynamics of the environment impact the query fidelity achieved by those protocols.

References

[1] V. Rajamani, C. Julien, J. Payton, G.-C. Roman, PAQ: persistent adaptive query middleware for dynamic environments, in: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, Middleware'09, 2009, pp. 12:1–12:20.

[2] J. Payton, C. Julien, G.-C. Roman, Automatic consistency assessment for query results in dynamic environments, in: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC-FSE, 2007, pp. 245–254.

[3] J. Payton, C. Julien, G.-C. Roman, V. Rajamani, Semantic self-assessment of query results in dynamic environments, ACM Transactions on Software Engineering and Methodology 19 (4) (2010) 12:1–12:33.

[4] C. Julien, V. Rajamani, J. Payton, G.-C. Roman, Fidelity-based continuous query introspection and adaptation, in: Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops—Workshop on Information Quality and Quality-of-Service for Pervasive Computing, IQ2S, 2011, pp. 14–19.

[5] V. Rajamani, C. Julien, J. Payton, G.-C. Roman, Inquiry and introspection for non-deterministic queries in mobile networks, in: Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering, FASE, 2009, pp. 401–416.

[6] A. Vargas, OMNeT++ web page, 2008. http://www.omnetpp.org.

[7] M. Loebbers, D. Willkomm, A. Koepke, The mobility framework for OMNeT++ web page, 2008. http://mobility-fw.sourceforge.net.

[8] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, in: Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking, 1998, pp. 85–97.

[9] V. Rajamani, C. Julien, Sama: static analysis for mobile applications, in: Proceedings of the 11th International Workshop on Mobile Computing Systems and Communications, HotMobile—Demo Sesion, 2010.

[10] R. Wang, D. Strong, Beyond accuracy: what data quality means to data consumers, Journal of Management Information Systems 12 (4) (1996) 5–33.

[11] D. Strong, Y. Lee, R. Wang, Data quality in context, Communications of the ACM 40 (5) (1997) 103–110.

[12] Y. Lee, D. Strong, B. Kahn, R. Wang, AIMQ: a methodology for information quality assessment, Information and Management 40 (2) (2002) 133–146.

[13] E. Herrera-Viedma, G. Pasi, A. Lopez-Herrera, C. Porcel, Evaluating the information quality of web sites: a methodology based on fuzzy computing with words, Journal of the American Society for Information Science and Technology 57 (4) (2006) 538–549.

[14] A. Nicolaou, D. McKnight, Perceived information quality in data exchanges: effects on risk, trust, and intention to use, Information Systems Research 17 (4) (2006) 332.

[15] R. Price, G. Shanks, A semiotic information quality framework: development and comparative analysis, Journal of Information Technology 20 (2) (2005) 88–102.

[16] Q. Han, I. Lazaridis, S. Mehrotra, N. Venkatasubramanian, Sensor data collection with expected reliability guarantees, in: Proceedings of the First IEEE International Workshop on Sensor Networks and Systems for Pervasive Computing, 2005, pp. 374–378.

[17] M. Sharaf, J. Beaver, A. Labrinidis, P. Chrysanthis, Balancing energy efficiency and quality of aggregate data in sensor networks, International Journal on Very Large Databases 13 (4) (2004) 384–403.

[18] C. Lu, G. Xing, O. Chipara, C.-L. Fok, S. Bhattacharya, A spatiotemporal query service for mobile users in sensor networks, in: Proceedings of the 25th International Conference on Distributed Computing Systems, ICDCS, 2005, pp. 381–390.

[19] C. Bisdikian, J.W. Branch, K.K. Leung, R.I. Young, A letter soup for the quality of information in sensor networks, in: Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops—Workshop on Information Quality and Quality-of-Service for Pervasive Computing, IQ2S, 2009, pp. 1–6.
[20] Q. Han, D. Hakkarinen, P. Boonma, J. Suzuki, Quality-aware sensor data collection, International Journal of Sensor Networks 7 (3) (2010) 127–140.
[21] C. Bisdikian, L. Kaplan, M. Srivastava, D. Thornley, D. Verma, R. Young, Building principles for a quality of information specification for sensor information, in: Proceedings of the 12th International Conference on Information Fusion, 2009, pp. 1370–1377.
[22] D. Barbara, Mobile computing and databases: a survey, IEEE Transactions on Knowledge and Data Engineering 11 (1) (1999) 108–117.
[23] A. Sistla, O. Wolfson, Y. Huang, Minimization of communication cost through caching in mobile environments, IEEE Transactions on Parallel and Distributed Systems 9 (4) (1998) 378–390.
[24] J. Kistler, M. Satyanarayanan, Disconnected operation in the Coda file system, ACM Transactions on Computer Systems 10 (1) (1992) 3–25.
[25] R. Dirckze, L. Gruenwald, A toggle transaction management technique for mobile multidatabases, in: Proceedings of the 7th International Conference on Information and Knowledge Management, CIKM'98, 1998, pp. 371–377.
[26] M. Dunham, A. Helal, S. Balakrishnan, A mobile transaction model that captures both the data and movement behavior, ACM—Baltzer Journal on Mobile Networks and Applications 2 (2) (1997) 149–161.
[27] Q. Lu, M. Satyanarayanan, Isolation-only transactions for mobile computing, Operating System Reviews 28 (2) (1994) 81–87.
[28] S. Madria, B. Bhargava, A transaction model for mobile computing, in: Proceedings of the International Database Engineering and Applications Symposium, 1998, pp. 92–102.
[29] G. Walborn, P. Chrysanthis, Transaction processing in PRO-MOTION, in: Proceedings of the 1999 ACM Symposium on Applied Computing, 1999, pp. 389–398.
[30] E. Pitoura, B. Bhargava, Maintaining consistency of data in mobile distributed environments, in: Proceedings of the 15th International Conference on Distributed Computing Systems, ICDCS, 1995, pp. 404–413.
[31] H. Yu, A. Vahdat, Design and evaluation of a conit-based continuous consistency model for replicated services, ACM Transactions on Computer Systems 20 (3) (2002) 239–282.
[32] A. Singla, U. Ramachandran, J. Hodgins, Temporal notions of synchronization and consistency in Beehive, in: Proceedings of the Symposium on Parallel Algorithms and Architectures, 1997, pp. 211–220.
[33] M. Bawa, A. Gionis, H. Garcia-Molina, R. Motwani, The price of validity in dynamic networks, Journal of Computer and System Sciences 73 (2007) 245–264.
[34] N. Jain, D. Kit, D. Mahajan, P. Yalagandula, M. Dahlin, Y. Zhang, Network imprecision: a new consistency metric for scalable monitoring, in: Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, 2008, pp. 87–102.
[35] A. Deshpande, Z. Ives, V. Raman, Adaptive query processing, Foundations and Trends in Databases 1 (1) (2007) 1–140.
[36] S. Madden, M. Shah, J. Hellerstein, V. Raman, Continuously adaptive continuous queries over streams, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2002, pp. 49–60.
[37] R. Avnur, J. Hellerstein, Eddies: continuously adaptive query processing, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2000, pp. 261–272.
[38] S. Babu, J. Widom, StreaMon: an adaptive engine for stream query processing, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2004, pp. 931–932.
[39] A. Deshpande, C. Guestrin, S. Madden, J. Hellersetin, W. Hong, Model-driven data acquisition in sensor networks, in: Proceedings of the 30th International Conference on Very Large Databases, 2004, pp. 588–599.
[40] C. Olston, J. Jiang, J. Widom, Adaptive filters for continuous queries over distributed data streams, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2003, pp. 563–574.
[41] L. Capra, G.S. Blair, C. Mascolo, W. Emmerich, P. Grace, Exploiting reflection in mobile computing middleware, ACM Mobile Computing and Communications Review 6 (4) (2002) 34–44.
[42] A. Chan, S.-N. Chuang, MobiPADS: a reflective middleware for context-aware mobile computing, IEEE Transations on Software Engineering 29 (12) (2003) 1072–1085.
[43] C. Bizer, R. Cyganiak, Quality-driven information filtering using the WIQA policy framework, Web Semantics: Science, Services and Agents on the World Wide Web 7 (1) (2008) 1–10.
[44] Y. Chen, Q. Zhu, N. Wang, Query processing with quality control in the world wide web, World Wide Web 1 (4) (1998) 241–255.