Fidelity-Based Continuous Query Introspection and Adaptation

Christine Julien[1], Vasanth Rajamani[2], Jamie Payton[3], and Gruia-Catalin Roman[4]
[1]The University of Texas at Austin, c.julien@mail.utexas.edu
[2]Oracle Corporation, vasanth.rajamani@oracle.com
[3]The University of North Carolina at Charlotte, payton@uncc.edu
[4]Washington University in Saint Louis, roman@wustl.edu

***Abstract*—In this paper, we address the fidelity of information collected from distributed, interactive, pervasive computing environments. Queries in these dynamic networks come in two forms: *snapshot queries* and *continuous queries*. The former executes through the network and returns values that satisfy the query criteria at a particular instant in time. The latter expects a result that is updated over time to reflect changes in the system. The *fidelity*, or quality, of the results of these queries can be significantly impacted by the unpredictability of the dynamic network; as the query is distributed and executed, changes that occur due to network and environmental dynamics can cause the query to miss potential results or to report inconsistent information. Our previous work has shown how we can provide semantically meaningful statements about the *fidelity*, or quality, of the result of a snapshot query in a dynamic networked environment. In this paper, we demonstrate how this fidelity can also be used to adapt a continuous query's processing to make its execution best match the conditions in the underlying dynamic environment.**

I. Introduction

Understanding the fidelity of information in dynamic pervasive computing environments is an essential but largely unexplored component in developing well-behaved applications. Emerging pervasive computing applications are characterized by distributed networks of sensing devices embedded into the physical environment. Mobile entities, including humans and vehicles, are also an integral part of these heterogeneous systems. To function in these spaces, applications must be context-aware, where context defines perceived situational awareness that stems from information about the surrounding environment. Understanding the *fidelity*, or quality, of information collected about the environment enhances decision processes and generates more well-informed, robust, and reliable applications that can more expressively and reliably adapt their behavior (and thus their performance) in response to a changing environment.

Building such applications is difficult since the system is dynamic and processing is expensive. Applications' queries propagate through the network, generating results that may or may not be correct, complete, or consistent, due to the network and environmental changes during query processing. We consider two forms of queries: *snapshot queries* and *continuous queries*. The former 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 automobile, 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 embedded nearby 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 a continuous query's evaluation within the pervasive computing interaction space can be used to assess tradeoffs between quality and cost of execution, and to adapt the query's execution.

We aim to develop a theory of the fidelity of queries in a pervasive computing network to enhance the decision processes of applications in these dynamic environments. Our original work developed a grounding theory of snapshot query fidelity in these environments [13], [14], which we review in Section III. We have shown that defining a continuous query as a sequence of snapshot queries can reduce the cost of monitoring in dynamic environments [17]; Section II details our model of the environment and both snapshot and continuous queries. The contribution of this paper, reported in Section IV 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.

II. Modeling the Environment and Queries

Understanding the execution environment is fundamental to modeling their fidelity. 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) [13], [14]. 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

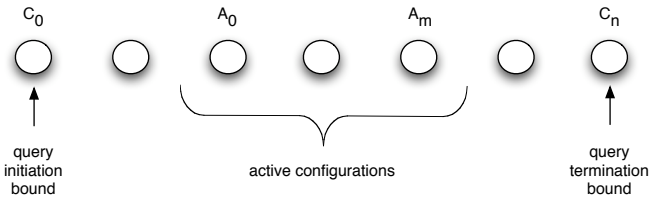


Figure 1. Query bounds and active configurations

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 Figure 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 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 Figure 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). 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 i^{th} component snapshot query) is $\pi_i = f(\rho_0 \ldots \rho_i)$, where f is an integration function.

Integration 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. 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 results from the continuous query that monitors some aspect of the quality of the continuous query. An introspection strategy can trigger an *adaptation strategy* that governs changing the continuous query's inquiry strategy. 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.

III. Fidelity of Snapshot Queries: A Review

In this section, we overview the fidelity of snapshot queries [13], [14], which lays the foundation for our use of this fidelity as an introspection metric for continuous queries.

To express fidelity snapshot query fidelity, we formalize the relationship between the state of the network and a query's result. We assign qualitative metrics (i.e., labels) to results; two of these fidelities (ATOMIC and WEAK) match the extremes currently available in mobile computing protocols. The others show 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 from 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:

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

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:

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

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 QUALIFIED SUBSET 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.

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.

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:

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

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

IV. 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.

A. Defining Fidelity-Based Introspection

To support introspection, we apply adequacy metrics to query results [16]. 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 such an adequacy metric based on the snapshot query fidelity defined in the previous section.

Our snapshot query fidelities 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., in 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.

Intuitively and empirically, inquiry strategies that can achieve comparable fidelity semantics incur greater overhead. The query protocols that provide this semantic require increased interaction among the participants; exchanging more messages incurs more resources. A continuous query can also be labeled with a fidelity semantic. An intuitive way to do this is to use the fidelities of the component snapshots, e.g., 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. In this situation, 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). We use these insights to define introspection based on the fidelity of snapshot queries and an adaptation strategy that uses this introspection to adapt a continuous query's behavior.

$\mathcal{S}_i$ is the fidelity of the continuous query's i^{th} snapshot, and $\mathcal{A}$ is the set of atomic fidelities (e.g., $\mathcal{A} = \{\text{atomic}, \text{atomic subset qualified subset}\}$). We define $d_{fidelity}(w)$ based on the fidelity of the past w snapshots:

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

where 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. An application can associate a threshold with $d_{fidelity}(w)$; 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.

B. Defining Fidelity-Based Adaptation

Applications use introspection to assess the continuous query's quality. If the result does not meet the application's requirements, adaptation strategies can change the query. In fidelity-based introspection, the application adapts its inquiry strategy to achieve higher fidelity results or less overhead. The following is a generic and simple structure for defining adapation strategies: $\langle\langle \mathcal{I}, freq\rangle, d, \delta^{+/-}, \langle \mathcal{I}^*, freq^*\rangle\rangle$, where $\langle \mathcal{I}, freq\rangle$ is an inquiry strategy, d is the introspection strategy, δ is the introspection threshold, and $\langle \mathcal{I}^*, freq^*\rangle$ is a new inquiry strategy. $\mathcal{I}$ is the protocol used to process the snapshot query, while *freq* is the frequency of the snapshot

[1] In the three-part notation: $\langle$ 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 $\forall$.

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.

Adaptation strategies are highly application dependent. In using our fidelity-based introspection, 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 100ms. If the application cannot achieve atomic fidelity, even with a sampling rate of 100ms, it desires to reduce the sampling rate to 1s 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):

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

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 100ms without satisfying the threshold for atomicity, the query frequency adapts back to 1s, 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 [15]. A straightforward fix would 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.

C. Demonstrating the Utility of Fidelity-Based Introspection

In this section, we briefly benchmark the use of fidelity-based introspection. An introspection is useful it if maps qualities of the environment that are important to the application to a tangible metric the application can use for adaptation. That is, the fidelity metric should capture dynamic aspects of the operating environment that have the potential to impact application performance.

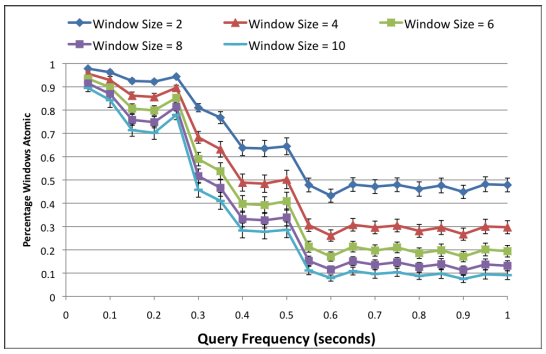
We used OMNeT++ [18], its mobility framework [10], and an implementation of a two phase protocol for snapshot queries that assesses its achieved fidelity [14]. The results below are executed on varying numbers of nodes at varying speeds within a 1000x900m^2 area. In each result, we specify a maximum speed; node speeds are chosen uniformly randomly between 0m/s and the specified maximum. The nodes move according to random waypoint mobility [4], in which each node is initially placed randomly, chooses a random destination, and moves in the direction of the destination at its assigned speed. We used the 802.11 MAC protocol; when possible we show 95% confidence intervals.

Figure 2 characterizes how a continuous query's inquiry strategy is related to fidelity-based introspection. Figure 2(a) shows the percentage of atomic windows for varying query frequencies, i.e., the y-axis plots $d_{fidelity}(w)$ for $w = \{2, 4, 6, 8, 10\}$. Figure 2(b) shows the average overhead of query processing. It is immediately obvious that using a higher query frequency achieves much higher fidelity but at a significantly increased cost. This is exactly the tradeoff our adaptation strategies identified in the previous section. In addition, even for very high continuous query frequencies that are quite costly, our very strong two phase query processing protocol cannot achieve perfect atomicity in this dynamic environment. This motivates an adaptation strategy that does not require a perfect result but can tolerate a weak integration for the continuous query every once in a while.

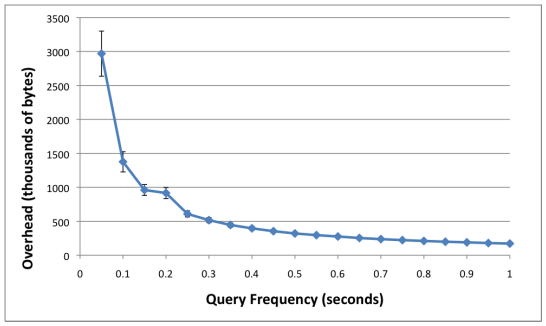
Figure 3 explores a single aspect of the dynamic environment, namely the node speed. As expected, fidelity-based introspection does provide a reflection of the degree of dynamics in the environment. Specifically, it is more difficult for our continuous query to provide an atomic measure of the environment as that environment grows more dynamic. We use this result next to demonstrate how using fidelity-based adaptation dynamically adjusts the continuous query.

D. A Continuous Query Fidelity-Based Adaptation

We now show how fidelity-based introspection can be used to adapt a continuous query to provide higher quality results or reduce the 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 seconds (we chose the short time scale simply to demonstrate adaptation). The maximum



(a) Percentage of atomic windows for varying query frequencies



(b) Overhead of protocol execution (in bytes)

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

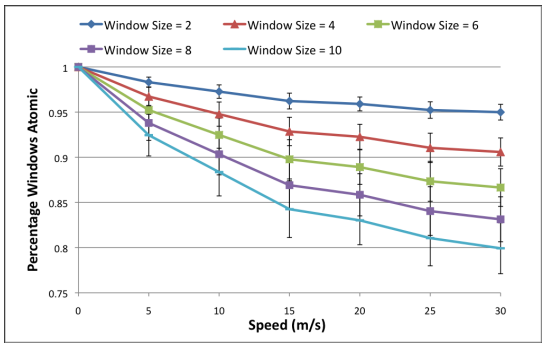


Figure 3. Impact of node speed on fidelity-based introspection (query frequency = 100ms, query ttl = 3 hops, number of hosts = 50)

speed of the nodes increases to 5m/s after 20 seconds; to 10m/s after 60 seconds, and to 15m/s after 100 seconds. We use our definition of $d_{fidelity}(w)$ with $w = 5$ and our two adaptation rules from Section IV-B, with the threshold δ set to 0.8. Figure 4 shows the adaptation of the query's frequency over time and the percentage of windows that are atomic over time. For brevity, we omit a plot of the query's overhead; it exhibits the behavior expected given Figure 2. As Figure 4 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.

Our initial set of adaptation rules adjust the query frequency to compensate for decreased atomicity. A more

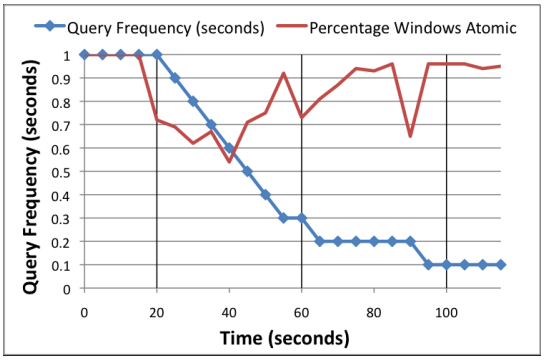
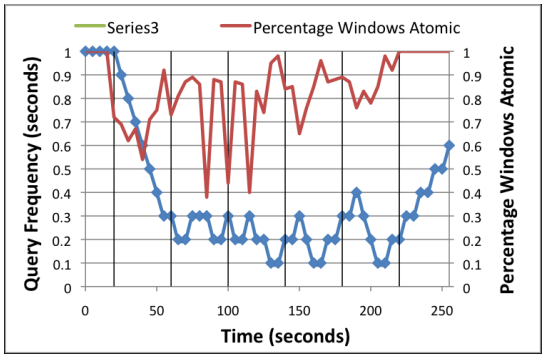


Figure 4. Impact of continuous query adaptation (query ttl = 3 hops, number of hosts = 50, max speed starts at 0m/s, goes to 5 m/s at 20 seconds, to 10m/s at 60 seconds, to 15m/s at 100 seconds)

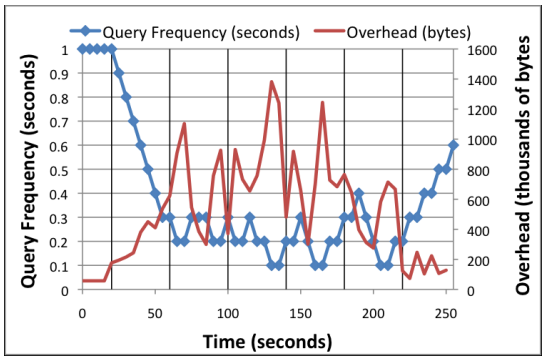
sophisticated set could weaken the continuous query to test whether the network conditions have changed such that an inquiry strategy with lower overhead may still achieve the query's atomicity goals. Our second scenario begins like the first, but after 140 seconds, the maximum speed decreases to 10m/s, after 180 seconds it decreases to 5m/s, and after 220 seconds, it decreases back to 0m/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 10 snapshot queries at a given frequency, we decrease the query frequency by 100ms. This provides a third adaptation rule to the set listed in Section IV-B. The results for this scenario are shown in Figure 5, which provides the adapted query frequency and the resulting fidelities and overheads. This adaptation strategy still does a good job of maintaining the atomicity requirement (with a few blips that result from testing higher query frequencies than were acceptable). More importantly, this continuous query strategy maintains low overhead when possible, and incurs high overhead only when necessary to maintain the application's desired atomicity.

V. Related Work

Query processing should adapt to the application's changing environment [9]. In adapting query processing, the focus is typically to change the order of query operations to optimize for dynamics. For example, Continuous Queries (CACQ) [11] rely on eddies [1] to determine the order in which partial query results are processed. StreaMon [2] 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 [8], 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 [12] 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



(a) Atomicity given adaptation



(b) Overhead given adaptation

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

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 [17]. Such *reflection* is common in mobile computing and middleware models [5], [6]; 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. This work has largely focused on how to filter query results into those that are likely to be of high-quality to the user [3] or to allow users to specify control parameters with web-based queries that dictate their resolution [7].

VI. Conclusions

The fidelity of query processing in dynamic pervasive networks depends heavily on the stability of the executing environment *during* query execution. In previous work, we demonstrated how to associate semantic tags with snapshot query results. Here, we extend that work and demonstrate how functions defined on these semantic tags can be used to adapt processing of continuous queries. With respect to our goals, this adaptation allows a continuous query to learn from its previous results 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).

References

[1] R. Avnur and J. Hellerstein. Eddies: Continuously adaptive query processing. In *Proc. of SIGMOD*, 2000.

[2] S. Babu and J. Widom. StreaMon: An adaptive engine for stream query processing. In *Proc. of SIGMOD*, pages 931–932, 2004.

[3] C. Bizer and R. Cyganiak. Quality-driven information filtering using the WIQA policy framework. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):1–10, January 2008.

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

[5] L. Capra, G. S. Blair, C. Mascolo, W. Emmerich, and P. Grace. Exploiting reflection in mobile computing middleware. *ACM Mobile Computing and Communications Review*, 6(4):34–44, October 2002.

[6] A. Chan and S.-N. Chuang. MobiPADS: A reflective middleware for context-aware mobile computing. *IEEE TSE*, 29(12):1072–1085, December 2003.

[7] Y. Chen, Q. Zhu, and N. Wang. Query processing with quality control in the world wide web. *World Wide Web*, 1(4):241–255, 1998.

[8] A. Deshpande, C. Guestrin, S. Madden, J. Hellersetin, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. of VLDB*, 2004.

[9] A. Deshpande, Z. Ives, and V. Raman. Adaptive query processing. *Found. and Trends in DB*, 1(1):1–140, 2007.

[10] M. Loebbers, D. Willkomm, and A. Koepke. The Mobility Framework for OMNeT++ Web Page. http://mobility-fw.sourceforge.net, 2008.

[11] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proc. of SIGMOD*, 2002.

[12] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. of SIGMOD*, 2003.

[13] J. Payton, C. Julien, and G.-C. Roman. Automatic consistency assessment for query results in dynamic environments. In *Proc. of FSE*, pages 245–254, September 2007.

[14] J. Payton, C. Julien, G.-C. Roman, and V. Rajamani. Semantic self-assessment of query results in dynamic environments. *ACM TOSEM*, 19(4), April 2010.

[15] V. Rajamani and C. Julien. Sama: Static analysis for mobile applications. In *Proc. of HotMobile*, 2010.

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

[17] V. Rajamani, C. Julien, J. Payton, and G.-C. Roman. PAQ: Persistent query middleware for dynamic environments. In *Proc. of Middleware*, 2009.

[18] A. Vargas. OMNeT++ Web Page. http://www.omnetpp.org, 2008.