Semantic Self-Assessment of Query Results in Dynamic Environments

Jamie Payton
Christine Julien
Gruia-Catalin Roman
Vasanth Rajamani

TR-UTEDGE-2008-012

Semantic Self-Assessment of Query Results in Dynamic Environments

JAMIE PAYTON
University of North Carolina at Charlotte
CHRISTINE JULIEN
University of Texas at Austin
GRUIA-CATALIN ROMAN
Washington University in Saint Louis
and
VASANTH RAJAMANI
University of Texas at Austin

Queries are convenient abstractions for the discovery of information and services, as they offer content-based information access. In distributed settings, query semantics are well-defined, e.g., queries are often designed to satisfy ACID transactional properties. When query processing is introduced in a dynamic network setting, achieving transactional semantics becomes complex due to the open and unpredictable environment. In this paper, we propose a query processing model for mobile ad hoc and sensor networks that is suitable for expressing a wide range of query semantics; the semantics differ in the degree of consistency with which query results reflect the state of the environment during query execution. We introduce several distinct notions of consistency and formally express them in our model. A practical and significant contribution of this paper is a protocol for query processing that automatically assesses and adaptively provides an achievable degree of consistency given the operational environment throughout its execution. The protocol attaches an assessment of the achieved guarantee to returned query results, allowing precise reasoning about a query with a range of possible semantics. We evaluate the performance of this protocol and demonstrate the benefits its serves applications through examples drawn from an industrial application.

Categories and Subject Descriptors: D.2.10 [**Software Engineering**]: Design—*methodologies*

General Terms: Design, Performance

Additional Key Words and Phrases: query processing, mobile computing, consistency

Jamie Payton is with the Department of Computer Science at the University of North Carolina, Charlotte. Email: payton@uncc.edu.
Christine Julien is with the Department of Electrical and Computer Engineering at the University of Texas at Austin. Email: c.julien@mail.utexas.edu.
Gruia-Catalin Roman is with the Department of Computer Science and Engineering at Washington University in Saint Louis. Email: roman@wustl.edu.
Vasanth Rajamani is with the Department of Electrical and Computer Enginering at the University of Texas at Austin. Email: vassanthrajamani@mail.utexas.edu.

1. INTRODUCTION

The widespread adoption of portable devices has the potential to support truly ubiquitous computing. These developments have led to heightened interest in designing software-intensive systems for mobile ad hoc networks, i.e., dynamic networks formed opportunistically by nodes within wireless communication range. Applications in such settings are often designed to exploit information and services provided by other applications in the network. In marketplace applications, a shopper may search for nearby services or products. On an intelligent construction site, a site supervisor can collect information from a network deployed on the site to manage assets and maintain safety.

An abstraction that can help simplify the process of discovery in such applications is a query. Query processing masks the details of complex network communication required to discover information and services distributed across a mobile ad hoc network. Query use in such open and dynamic settings is particularly appropriate, as queries eliminate the unrealistic assumption of knowing in advance the location or exact nature of the desired information.

Traditionally, database query semantics have been precisely defined to ensure that executing a query results in a single, correct answer, usually requiring a transaction which upholds the *ACID properties* of *atomicity*, *consistency*, *isolation*, and *duration*. In distributed databases, preserving these properties often requires a distributed locking protocol that prevents changes to data during query execution. In effect, a query appears to execute over all hosts in the network in a single step. Applying the ACID properties becomes more complicated when hosts are mobile because such locking protocols are expensive in highly dynamic environments rife with disconnections. In addition, using locking protocols in these networks, which are often designed to provide access to streaming data, is not feasible. Attempting to strictly adhere to the ACID semantics makes it difficult, if not impossible, to receive any query results under common operational conditions.

We contend that a number of applications for dynamic computing environments may require guarantees other than strict transactional semantics. We propose a new perspective on query semantics that allows us to discover, precisely define, and reason about the kinds of query semantics needed by applications in these dynamic environments. We introduce a model that can be used to formalize a range of consistency semantics associated with query execution in mobile ad hoc networks. To our knowledge, this is the first attempt to provide a general specification method for query execution semantics in such networks. We use this model to formally express novel consistency semantics that lie in between the extreme strong and weak forms of consistency typically identified in query processing models.

The ability to express query consistency semantics will provide a solid intellectual foundation for discovery and enhanced understanding, but without a practical realization of a particular semantic, it is of no use in application design. For this reason, our work couples the formal expression of query semantics with protocol development. We present a protocol for query execution in dynamic ad hoc networks that automatically assesses the changing operating environment and adapts its execution to provide query results. An assessment of the achieved consistency semantic is provided with the query results. Such a protocol offers a more flexible approach to query execution than transactional approaches yet allows for careful reasoning about query results.

This remainder of this paper is organized as follows. Section 2 overviews related work and uses it as a foundation to motivate our problem definition. Section 3 introduces our model of dynamic networks and query execution within them. A range of consistency semantics are introduced in Section 4, coupled with application examples and formalizations. In Section 5, we present an adaptive, self-assessing protocol for query execution that provides varying degrees of consistency; an implementation and evaluation of that protocol is described in Section 6, including an application-driven analysis. Section 7 concludes.

2. RELATED WORK AND PROBLEM DEFINITION

In this section, we review existing approaches that relate to our proposed query processing and assessment framework. We then use this existing work to crystallize the definition of the specific problem we undertake.

2.1 Existing Approaches

Distributed databases have traditionally focused on wired, strongly connected environments. As devices become increasingly mobile, the research community has responded by investigating the deployment of databases in mobile and peer-to-peer network settings [Barbara 1999]. Several of these strategies focus on issues related to dynamic cache allocation [Sistla et al. 1998] or optimistic replication [Kistler and Satyanarayanan 1992], while others allow applications to explicitly issue *weak operations* that are allowed to operate over potentially inconsistent data [Pitoura and Bhargava 1995]. Our approach differs in that we avoid caching data locally, instead desiring to acquire it on-demand from a dynamic environment. We also postpone the decision of how weak of an operation to perform until run time, providing applications with the strongest semantic achievable in a given operational context.

In a similar vein, researchers have looked within mobile database systems at transactional semantics. This work has begun to address the need for a new view of consistency semantics by proposing new transaction models for mobile settings. Many of these models relax the constraints imposed by the ACID properties and execute queries using transactions which adhere to a weaker set of properties, though the approaches tend to differ significantly. A few [Dunham et al. 1997; Walborn and Chrysanthis 1999] use the concept of *split* transactions to handle intermittent disconnections and reconnections. Others focus on maintaining or relaxing a particular ACID semantic; isolation-only transactions [Lu and Satyanarayanan 1994] ensure only that committed transactions appear as though executed independently; toggle transactions [Dirckze and Gruenwald 1998] enable extended execution, relaxing both atomicity and isolation; and the pre-write transaction model [Madria and Bhargava 1998] focuses almost exclusively on data-availability. These models are generally limited to use in nomadic networks, in which periodic access to the wired infrastructure is available, and solutions can rely on the use of a resource-rich fixed node to manage transactions. Because of their reliance on powerful and fixed nodes on the fringe of the network, these weakened transactional models cannot be applied to the more extreme form of mobility found in mobile ad hoc networks. Moreover, the frequent disconnections and reconnections in a mobile ad hoc network could result in significant overhead when employing similar approaches.

Closely related to our work is a study of query semantics for dynamic networks [Bawa et al. 2004]. The authors define a new class of semantics based on the "single site validity" principle, in which a query result appears to be equivalent to an atomic execution from the query issuer's perspective. While the themes are similar, the work differs in scope. Their study defines a particular class of semantics, while we attempt to provide a model that can define classes of semantics.

2.2 Problem Definition

As a motivation for the problem we undertake, consider an application querying a network spanning a construction site. Such intelligent construction sites are becoming increasingly commonplace, connecting sensors distributed around the site to measure environmental and structural conditions with small mobile devices carried by workers and inspectors and more powerful stationary computers. A worker may pose queries to the network to retrieve measures of the concentration of a potentially hazardous compound. Given that the worker knows what concentration constitutes a danger, he can use the result of this query to take subsequent actions. However, in a dynamic environment, there may be some degree of uncertainty associated with the query's result, due simply to movement in the network over which the query was issued. If the worker has an understanding of the degree of this uncertainty, his subsequent actions may change. For example, while a dangerous concentration may call for evacuation in all cases, containment measures may depend on the actual concentration. If the worker is certain of the query response (i.e., the query was executed with the traditional strong semantics), he may be able to begin the appropriate containment procedure immediately. However, if uncertainty exists, further measurements may be necessary before the appropriate action is known.

Such a scenario provides a motivation for a query processing protocol that can assess a query's achieved guarantee as it processes the query. This is in contrast to existing protocols that provide rigid implementations at one extreme or the other. For example, a protocol providing a strong guarantee may only deliver a result when that strong guarantee can be achieved, leaving applications without any information when the guarantee is not possible. On the other hand, a best-effort protocol may be able to always provide a result, but an application has no idea how that result relates to the ground truth. Therefore, it becomes meaningful to provide a self-assessing query processing protocol that not only delivers a query result but also labels that result with the achieved semantic.

In this paper, we build on our previous work [?] in addressing the need for a fundamental reexamination of query processing theory and practice. This requires a formal framework that enables characterization and reasoning about query consistency coupled with a precise formal characterization of a range of application-relevant types of consistency. Providing these consistency ranges to concrete applications requires a protocol that not only supports the semantics but can reflect upon its own behavior to provide an assessment of its achieved consistency. Such self-assessment allows application logic to rely on query consistency to adapt decision making processes.

3. A MODEL OF QUERY EXECUTION

In this section, we introduce the concepts that lay the foundation for our study of query processing in mobile environments. The idea is to model the evolution of the system as transitions between successive configurations and to relate new concepts such as query consistency semantics to this model. By modeling configurations as sets of mobile entities with associated attributes, we provide a model that is general and flexible enough to capture different types of queries over different types of mobile entities independent of a query specification language, and we can easily express mobility and time as state transitions. We subsequently use this model to precisely define the query processing guarantees that can be offered to queries in dynamic mobile computing environments. Furthermore, this model can drive the discovery of new semantics that may be further beneficial to application development.

3.1 Modeling the Environment

Understanding the environments in which queries execute is fundamental to modeling their semantics. We view a mobile ad hoc network as a closed system of hosts where each host h has a location and a single data value (though a single data value may represent a collection of values). A host is represented as a tuple (ι, ν, λ), where ι is a unique host identifier, ν is the host's data value, and λ is the host's location. The global abstract state of an ad hoc network, which we call a *configuration*, is simply a set of host tuples. Formally, we describe a configuration as:

$$C \equiv \bigcup_{i=0}^{H} (\iota, \nu, \lambda)_i$$

where H is the number of hosts in the network.

Given a specific host $\bar{h}$, called the *reference host*, we define an *effective configuration* as the projection of the configuration with respect to the set of hosts that are *reachable* from $\bar{h}$. Reachability is often defined in terms of physical network connectivity, captured by a relation that conveys the existence of a (possibly multi-hop) network path between a pair of hosts. We use a binary logical connectivity relation $\mathcal{K}$ to express the existence of a direct (one-hop) communication link between two hosts. Reachability is defined as the reflexive transitive closure relation $\mathcal{K}^*$. Using a host's state (i.e., the values of fields of a host tuple), we can derive physical and logical connectivity relations in a configuration and, in turn, the reachability relation on hosts in the ad hoc network. A physical connectivity relation that represents a connectivity model with a circular, uniform communication range can be defined using the location field of host tuples:

$$(h_1, h_2) \in \mathcal{K} \Leftrightarrow |h_1 \uparrow 3 - h_2 \uparrow 3| \leq d$$

where $\uparrow 3$ refers to a tuple's third field (i.e., the host's location) and d is the communication range. It is possible to model other physical connectivity models in a similar fashion, and logical connectivity relations can be defined using constraints on the identifier and value fields of a host tuple.

Given this definition of reachability, we define the portion of the global abstract state of the ad hoc network that may, in principle, be visible to a reference host.

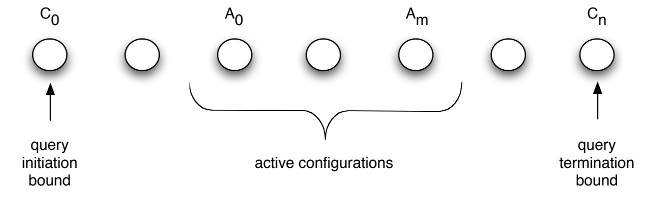

Fig. 1. Query bounds and active configurations

We call this locally visible state an *effective configuration*, which is a projection of a configuration with respect to the reachable hosts. We formally define an effective configuration E for a reference host $\bar{h}$ in a configuration C as:

$$C \downarrow \bar{h} = \langle \cup h, \bar{h} : h \in C \wedge \bar{h} \in C \wedge \bar{h}\mathcal{K}^* h :: h \rangle$$

where $\mathcal{K}^*$ is logical connectivity, and $\downarrow$ denotes projection.

The environment, however, is not static; it evolves as the network topology changes and value assignments occur. We model this evolution as a state transition system in which the state space is defined by the set of possible system configurations, and transitions are defined as configuration changes. Sources of configuration change include:

—*variable assignment* a single host changes its data value ν, resulting in a new configuration. Formally, this is:

$$\begin{aligned} \mathit{value\_change} \equiv\ & \langle \exists h : h \in C_i :: \langle \exists h', v : h' \in C_{i+1} \wedge \\ & v \neq h \uparrow 2 :: h' = (h \uparrow 1, v, h \uparrow 3) \rangle \rangle \end{aligned}$$

—*neighbor change*: the change in a host's location impacts the logical connectivity of the network; as a result, some host in the network will experience a change in its set of logically connected neighbors. A neighbor change occurs when a host is no longer connected to a previous neighbor (i.e., the pair of hosts no longer belongs to the connectivity relation $\mathcal{K}$) or becomes connected to a new neighbor (i.e., the pair of hosts now belongs to the relation). We formally describe this as:

$$\begin{aligned} \mathit{neighbor\_change} \equiv\ & \langle h_1, i : h_1 \in C_i :: \\ & \langle h'_1, h_2, l : h'_1 \in C_{i+1} \wedge h_2 \in C_{i+1} :: \\ & h'_1 = (h_1 \uparrow 1, h_1 \uparrow 2, l) \wedge l \neq h_1 \uparrow 3 \wedge \\ & (((h'_1, h_2) \in \mathcal{K} \wedge (h_1, h_2) \notin \mathcal{K})) \vee \\ & (((h'_1, h_2) \notin \mathcal{K} \wedge (h_1, h_2) \in \mathcal{K})))\rangle\rangle \end{aligned}$$

We can now define a configuration change as:

$$\Delta C \equiv \langle \mathit{value\_change} \oplus \mathit{neighbor\_change} \rangle$$

The exclusive-or notation $\oplus$ indicates that we model one change at a time. From a global perspective, system evolution can be viewed as a sequence of configurations associated with successive transitions. For a reference host, this evolution can be viewed as a sequence of effective configurations.

3.2 Defining Queries and Results

We use this model of an evolving system to reason about the results of a query issued over a mobile ad hoc network. Consider the sequence of configurations in Figure 1. A single query may span such a sequence, starting with the configuration in which the query is issued (the *query initiation bound*, C_0) and ending in the configuration that corresponds to the delivery of the result (the *query termination bound*, C_n). We define $\langle C_0, C_1, \dots C_n \rangle$ as the set of configurations over which a query is executed. No configuration outside these bounds can impact the query's result. Since there is processing delay associated with issuing a query to and returning results from the network, we define a query's *active configurations* as the configurations within the sequence $\langle C_0, C_1, \dots C_n \rangle$ during which the query actually interacted with hosts in the network. Every component of a query's result must be a data element that is part of some configuration belonging to the set of active configurations. In reality, only the query issuer's effective configurations (containing the reachable hosts) can contribute to its active configurations. These are the *effective active configurations*, bounded by the *effective active initiation bound*, E_0, and the *effective active termination bound*, E_m as shown in Figure 2. $E_0, E_1, \dots E_m$ is the sequence of effective active configurations over which a query is evaluated.

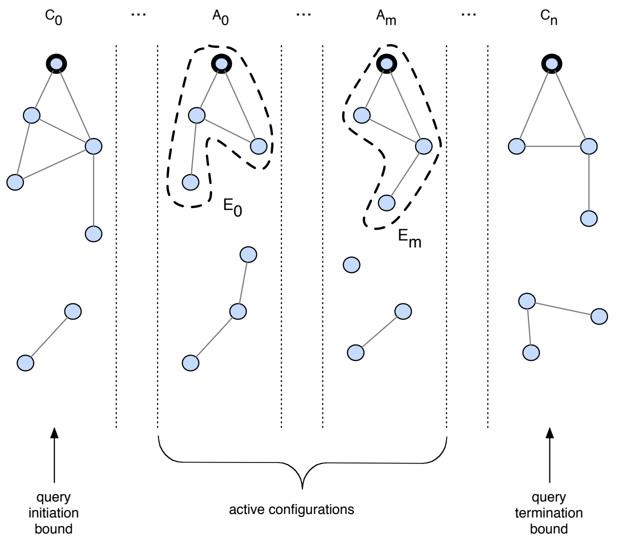

Fig. 2. Effective active configurations. Circles are hosts; solid lines represent the logical connectivity relation. Dashed lines show effective active configurations.

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

The configuration comprising the results is subject to a set of constraints. First, each element r in the result configuration ρ must be reachable from the query issuer ($\bar{h}$) in at least one of the effective active configurations. Second, only one query result per host should be present in the result set. Formally:

$$\begin{aligned} h \in \rho \;\Rightarrow\; & \langle \exists i : 0 \leq i \leq m :: h \in E_i \,\wedge \\ & \langle \forall r : r \in \rho - \{h\} :: h \uparrow 1 \neq r \uparrow 1 \rangle\rangle \end{aligned}$$

which states that any host tuple h in the result ρ must have existed in one of the effective active configurations (E_i) and that, for every host tuple in the result, there must not be another tuple in the result with the same unique id ($h \uparrow 1$).

Our goal is to define the degree of consistency for a query issued over a dynamic ad hoc network. Given our model, we can achieve this by examining the relationship between the result configuration ρ of a query and the effective active configurations that contributed to the query's evaluation. Next, we use this model to formalize new notions of consistency that are useful to application developers.

4. DEFINING QUERY CONSISTENCY

We wish to capture a range of consistency degrees that are desirable for applications in mobile ad hoc and sensor networks. In this section, we enumerate a set of consistency guarantees that can be determined for queries that involve a single request/reply exchange between the query sender and the other nodes in the active configuration(s). For each of the semantics we provide, we give a precise formalization that conveys the relationship between the state of the ad hoc network and the query's returned result. To make such specifications useful to application developers, the next section presents a single protocol that provides different consistency guarantees depending on the run-time environmental characteristics. The protocol also communicates the resulting consistency of a requested query to the application.

To demonstrate the usefulness of this new set of consistency guarantees, we provide application examples from two domains and indicate how results for each semantic can be used. In the first domain, we demonstrate how query consistency can prove useful in gathering information from a construction site. Specifically, we will look at a query that gathers information about the amount of a material (e.g., palettes of bricks) on the site. We revisit examples from construction sites in later sections to demonstrate the applicability of our approach. In the second example, a query searches a mobile ad hoc network for ticket reservation prices (or any commodity offered by multiple sellers) and returns specifics about the potential reservation (e.g., flight times) and the associated price.

4.1 Guaranteed Availability: IMMEDIATE

The strongest consistency guarantee ensures that all of the results a query returns were available at the same time and that they are still available when the query returns. In the construction site example, a query with IMMEDIATE semantics can give the construction site supervisor a complete and accurate picture of his site at the instant his query returns, allowing him to know which materials are currently present and to subsequently make future plans and schedules based on the results. In a travel reservation system, a query response with such strong semantics indicates that all of the returned potential reservations are competitive prices that can be purchased at the instant the query returns.

Formal Specification. The IMMEDIATE consistency states that not only were all of the results returned present in the same configuration but that the results were available when the query started and were still available to the requester when the query returned, i.e., that nothing changed while processing the query. Formally, this is:

$$\text{IMMEDIATE} \equiv \rho = E_0 \wedge m = 0$$

where ρ is the set of results returned and m indicates a configuration in the sequence of active configurations.

4.2 Strong Guarantees: ATOMIC

Many applications require that a query result provides an exact view of the surrounding environment but may not require the added component of the IMMEDIATE guarantee that the results are still available. In our construction site example, a sequence of results with ATOMIC semantics gives the construction site supervisor a temporal picture of how materials are consumed across the site. In a travel reservation system, a query with such semantics gives the shopper a guarantee that the prices quoted are comparable across different carriers since the results were all collected in the same configuration. In these cases, the relationship among the items returned is important; all of the responses returned should have been present in the same configuration to give an accurate picture of the network state at a single point in time.

Formal Specification. We capture the ATOMIC consistency level in our model by stating that the query was performed on a *single* effective active configuration ($E_i(h)$) and that it effectively returned a snapshot of that configuration. Formally, this is simply:

$$\text{ATOMIC} \equiv \exists i : 0 \leq i \leq m \wedge \rho = E_i(h)$$

where h is the reference host (so $E_i(h)$ is an effective active configuration for host h). Setting ρ *equal* to the configuration signifies not necessarily that the application uses all of the results but that they are available. We believe this is the strongest consistency semantic we can potentially provide given data and network dynamics.

4.3 Partial Results: ATOMIC SUBSET

In many instances, applications may only need a certain number of resources to complete a task. A construction site supervisor may know that he requires a certain amount of a given material for a particular task, and a query that returns the subset of the assets available may be sufficient to complete a particular task. In the reservation system, a query that has an ATOMIC SUBSET guarantee ensures that all the results that are returned are comparable (since they were all collected in the same configuration). It does not guarantee, however, that all possible ticket prices were returned.

Formal Specification. An ATOMIC SUBSET consistency dictates that all of the results that are returned should have been present in the same effective configuration, but does not require that everything present in that configuration is returned. Formally, we express this as:

$$\text{ATOMIC SUBSET} \equiv \exists i : 0 \leq i \leq m \wedge \rho \subset E_i(h)$$

which states that the result set ρ is exactly a subset of one of the effective active configurations. That is, all of the results in ρ were present in a single configuration, but the result set may not contain all of the values from that configuration.

4.4 Degrees of Partial: QUALIFIED SUBSET

A slightly better picture for the reservation system would provide the shopper some information about what fraction of results the query potentially missed. If the returned result represents a large sample of the possible results, the shopper may have more confidence in the lowest fare reported being near the actual lowest fare. We refer to this semantic as QUALIFIED SUBSET because the result is qualified with respect to the potential result. In the construction scenario, a query of materials on the site gives the site supervisor a view of a certain percentage of the available materials, potentially allowing him to make some worst-case plans.

Formal Specification. The formalization of the QUALIFIED SUBSET consistency level is a specialization of ATOMIC SUBSET to constrain the results returned. It requires that at least α percent of the results that were available in all of the effective active configurations are returned. Formally:

$$\text{QUALIFIED SUBSET} \equiv \exists i : \rho \subset E_i \wedge \mid \rho \mid > \alpha \mid E_i \mid$$

where $\mid \rho \mid$ is the cardinality of the set of results returned, and $\mid E_i \mid$ is the total number of results that were present over all the effective active configurations.

4.5 Weak Guarantees: WEAK

The weakest guarantee our framework provides to applications simply ensures that all of the results returned were present in at least one of the effective active configurations. Our construction site supervisor may have no significant use for weak semantics because they give him no reliable information about his materials. In our reservation system, on the other hand, there is no guarantee that the fares are directly comparable (since they may have been collected from different carriers at different times), but they offer a view of some of the options. This can give the shopper a quick idea of what the fare range is, but it is likely not something a shopper will want to base a purchase on unless pressed for time.

Formal Specification. We capture the weakest form of guarantee by ensuring that anything that was returned was at least present in one of the effective active configurations:

$$\text{WEAK} \equiv \rho \subseteq \bigcup_{i=0}^{m} E_i$$

This semantic does not provide any information about the relationships among the returned results and is the weakest meaningful consistency semantic we can provide.

4.6 Degrees of Weak: WEAK QUALIFIED

The final consistency semantic our framework can provide is WEAK QUALIFIED. In this case, the results collected may have come from across all effective configurations (i.e., they may not have all existed at the same time), but the requester is guaranteed to have received at least some fraction of the possible results. In the construction site scenario, this gives the commander some information about the relative recent availability of some materials. He can use this information to make some worst-case plans, but he can't base these plans on complete information on, for example, relative locations of materials, since the information comes from different configurations. In the reservation system, the shopper is again guaranteed to have received a certain percentage of the available fares, but since these may have come from different configurations, they may not be directly comparable.

Formal Specification. As a slightly stronger version of the weak guarantee, the WEAK QUALIFIED consistency specifies that the result contains at least some minimum fraction of the results that were present over all the effective active configurations. That is:

$$\text{WEAK QUALIFIED} \equiv \rho \subseteq \bigcup_{i=0}^{m} E_i \wedge |\rho| > \alpha \, |\bigcup_{i=0}^{m} E_i|$$

5. A SELF-ASSESSING QUERY PROTOCOL

In this section, we present a protocol that can provide any of the consistency semantics introduced in Section 4. The semantic achieved depends on the conditions of the environment during query execution. The protocol dynamically assesses which semantic is achieved and attaches this assessment to the returned query results. By providing this protocol, we demonstrate the feasibility of implementing the semantics and provide developers with a flexible mechanism for query execution that has an underlying formal foundation for precise reasoning about query results.

5.1 Protocol Overview

A typical approach to providing strong consistency relies on locking data items that contribute to a query's result. This solution may hinder concurrent execution; data items that are merely read and not changed by a query's execution are locked and therefore unavailable to others during query execution. Our approach does not require data items to be locked during query execution and instead maintains state about data values that will be accessed during query evaluation and determines if the values remain accessible and unchanged throughout execution. Using this information, the protocol can compute the semantic the query achieved.

We rely on a controlled flooding approach to distribute and evaluate a query. One can think of a message spreading throughout the reachable portion of the network like a wave. Hosts that have received the message are "behind" the wave, while hosts that have not yet received the message are "in front of" the wave. We use these notions of "behind" and "in front of" to determine the impact of environmental changes on the protocol's execution and the semantic achieved.

Our protocol uses two flooding phases, shown pictorially in Figure 3. The first phase precisely identifies the query initiation bound (as defined in Section 3), while

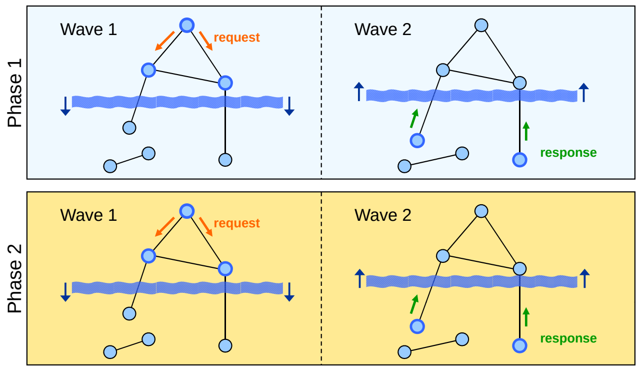

Fig. 3. Protocol Phases and Waves

the second collects the data values to return. The first phase constructs a tree of the query's initial participants, and every member in this tree knows both its parent and its children. This phase completes when the reference host has collected replies from all of its children, and the query initiation bound is established. When a host in the tree receives the second phase of the query, it passes the query to its children. When all of its children have replied, the host replies. The query is complete when the reference host has received replies from all of its children.

Each of these flooding phases comprises two waves: one that disseminates the request and one that returns the response. Each participating host monitors changes in its state (i.e., variable changes and neighbor changes) that occur behind and in front of each wave and may impact the achievable consistency semantic. For example, if a host that is established as a participant in the query during the first phase becomes disconnected before replying in the second phase (i.e., in front of the second phase's second wave), the ATOMIC guarantee cannot be provided. The disconnected host's parent logs the disconnection and passes this information to the query issuer with the result. The reference host communicates to the application the strongest possible semantic that the protocol can guarantee was satisfied.

In practice, flooding an entire network can be prohibitively expensive and may cause unreasonable response times. One way to control this cost is to limit the query's scope. In our approach, flooding is constrained by a query's logical connectivity relation $\mathcal{K}$. Previous work provides practical solutions for scoping [Julien and Roman 2002; Kabadayi and Julien 2007; Roman et al. 2002]; these can easily be adapted to provide foundational execution support for our protocol.

In Section 5.2, we provide a detailed description of this self-assessing query execution protocol. We assume the use of a reliable message delivery mechanism (research on reliability continues to advance, e.g., [Julien et al. 2008; Si and Li 2004; Vellambi et al. 2007]). Also, we assume that each host can detect connection and disconnection of its neighbors using one of the mentioned scoping approaches.

5.2 Protocol Description

The state variables for each host are shown in Figure 4. Only the state for a single query execution is shown; each query execution has a duplicate set of variables. To define the protocol's behavior, we use I/O Automata notation [Lynch and Tuttle 1989]. We show the behaviors of a single host, A, indicated by the subscript A on each behavior. Each *action* (e.g., $ParticipationRequestReceived_A(r)$ in Figure 5) has an effect guarded by a precondition. Actions without preconditions are *input actions* triggered by another host. Each action executes in a single atomic step. We abuse notation slightly by using, for example, "send *ParticipationRequest*(r) to *Neighbors*" to indicate a sequence of actions that triggers *ParticipationRequestReceived* on each neighbor.

Each host uses two boolean flags, *membership* and *monitoring*, to aid in the determination of the consistency semantic. The *membership* flag is used in the first flooding phase to identify participants in the query execution protocol, while the *monitoring* flag is used in the second flooding phase to identify data values that are being queried. The *departed-flag* and *added-flag* variables are used to support the determination of the consistency assessment; if either is non-zero, then atomic consistency cannot be provided.

State characterization for host *A*

id	– A's unique host identifier
neighbors	– A's logically connected neighbors
results	– set of (id, data value) pairs provided by A and its descendants.
membership	– boolean, indicates A is in the query; used in first phase
monitoring	– boolean, indicates A is preparing result; used in second phase
request	– the request currently being processed
parent	– A's parent in the tree
replies-waiting	– neighbors still to respond
participants	– A's descendants that are participating
results	– set of (id, data value) pairs provided by A and its descendants.
departed-flag	– true if one or more nodes below below A in the tree has departed
added-flag	– true if one or more nodes has been added below A in the tree

Fig. 4. State Variables for Protocol

5.2.1 *Establishing the Query Initiation Bound.* The first flooding phase of the protocol constructs a spanning tree that consists of all hosts that are initial participants in the query's execution. In terms of the query model presented in Section 3, the first flood defines the initial configuration members and establishes the query initiation bound. Two waves are used within this first flood: one to disseminate the participation request, and one to return the responses of participating hosts. The reference host is responsible for initiating the first wave to receive acknowledgments of participation. Figure 5 shows the action that occurs when a host receives this query participation request in the first wave. The host sets its *membership* flag and records its *parent* in the tree. The host then sends the request to its neighboring hosts and records them. The host must wait for all of its children to reply before it can send its own reply. Once the initial wave of the first flood reaches a host on the boundary of the network, the boundary host initiates the reply process, i.e., the second wave in the first flood. If a host receives the same participation request (i.e., along a second communication path), it cancels this request. When this message is processed at the parent, the parent removes the host from its *replies-waiting* variable (since another host is the parent). This action is omitted for brevity.

Since the network is open and hosts may be mobile, the set of hosts that participate in the query's execution may change over time. These changes can impact the consistency semantic achieved. Some changes to the set of participating hosts can be tolerated and the strongest form of consistency, ATOMIC, can still be achieved. For instance, we can tolerate additions to and deletions from the set of participating hosts until the members of the set are officially established at the query issuer. The actions *NeighborAdded* and *NeighborDeparted* in Figure 6 describe how our protocol handles these changes.

```
$ParticipationRequestReceived_A(r)$
  Effect:
    if $\neg membership$ then
      $membership := true$
      $parent := r.sender$
      $request := r$
      if $(neighbors - r.sender) \neq \emptyset$ then
        for each $B \in (neighbors - r.sender)$
          send $ParticipationRequest(r)$ to $B$
          $replies\text{-}waiting := neighbors - r.sender$
        end
      else
        send $ParticipationReply$ to $parent$
      end
    else
      send $CancelParticipationRequest$ to $r.sender$
    end
```

Fig. 5. The *ParticipationRequestReceived* action

In both actions in Figure 6, the first **if** condition handles the neighbor change event between the first and second waves of the first flood. In both cases, we can handle the neighbor change; we must simply ensure that the request propagation is handled correctly. In the case of an added neighbor, the new host is added to the participation request and becomes this host's child. For a departed neighbor, this host no longer waits for the host's reply. We will revisit the other cases in Figure 6 as we move through the flood phases.

Once the initial wave of the first flood reaches a host on the boundary of the network, the boundary host initiates the reply process, i.e., the second wave in the first flood, by sending a *ParticipationReply* to its parent. Figure 7 shows the action handling the reception of this message.

When a host receives all of the participation replies it is waiting on, it replies to its parent. When it does, it aggregates the participant information it has received and passes its parent a list of all participants in its subtree.

```
$NeighborAdded_A(B)$
  Precondition:
      $connected(A, B) \wedge B \notin neighbors$
  Effect:
      $neighbors := neighbors \cup \{B\}$
      if $membership$ then
          if $\neg monitoring \wedge (replies\text{-}waiting \neq \emptyset)$ then
              send $request$ to $B$
              $replies\text{-}waiting := replies\text{-}waiting \cup \{B\}$
          else
              $added\text{-}flag := true$
          end
      end

$NeighborDeparted_A(B)$
  Precondition:
      $\neg connected(A, B) \wedge B \in neighbors$
  Effect:
      $neighbors := neighbors - \{B\}$
      if $membership$ then
          if $B = parent$ then
              [reset state]
          else if $\neg monitoring \wedge (replies\text{-}waiting \neq \emptyset)$ then
              $replies\text{-}waiting := replies\text{-}waiting - \{B\}$
          else if $\neg monitoring$ then
              $departed\text{-}flag := true$
              $participants := participants - \{B\}$
          else if $(replies\text{-}waiting \neq \emptyset)$ then
              $departed\text{-}flag := true$
              $replies\text{-}waiting := replies\text{-}waiting - \{B\}$
          end
      end
```

Fig. 6. Actions for handling neighbor changes

```
$ParticipationReplyReceived_A(r)$
  Effect:
      $replies\text{-}waiting := replies\text{-}waiting - r.sender$
      $participants := participants \cup \{r.participants\}$
      if $replies\text{-}waiting = \emptyset$ then
          if $r.requester \neq id$
              send $ParticipationReply$ to $parent$
          else
              send $Query$ to $neighbors \cap participants$
          end
      end
```

Fig. 7. The *ParticipationReplyReceived* action

The first phase of the protocol is complete when the reference host has collected all replies from its children, and the query initiation bound is established. The

```
$QueryReceived(q)$
  Effect:
      if $membership \wedge \neg monitoring$
         $\wedge q.sender = parent$ then
           $monitoring := true$
           if $participants \neq \emptyset$ then
               $replies\text{-}waiting := participants)$
               send $Query$ to $neighbors \cap participants$
           else
               send $QueryReply$ to $parent$
               [reset state]
           end
      end
```

Fig. 8. The *QueryReceived* action

reference host's *participants* variable contains the query's established participants. Any changes in connectivity that result in change of membership after the completion of this phase will result in a semantic weaker than ATOMIC. At the end of the first phase, the reference host sends a *Query* to its participating neighbors to initiate the second flooding phase.

5.2.2 *Establishing and Reporting Query Results.* The protocol's second flood requests query results from hosts in the tree constructed in the first phase. Once again, two waves are used: one to disseminate the query and one to propagate results. The action performed by a host receiving a query is shown in Figure 8. Each host receiving the query sets its *monitoring* flag. As before, each parent in the tree must wait for responses from its children before sending its own query results. Boundary hosts initiate the second wave of the second flood to deliver query results. In constructing a query result, a boundary node includes its own data value and its *departed-flag* and *added-flag* variables. As these replies propagate up the tree, parents aggregate the results and flags of their children, add their own information, and send a summary further along. This allows the reference host to assess the query consistency. In this flooding phase, the setting of the *monitoring* flag and checking for changes in data during query execution is analogous to the use of locks in traditional protocols, but is less restrictive.

Changes in the environment that occur "in front of" the second flood's second wave may impact the set of hosts participating in the query as well as the available data, which will impact what consistency semantic the protocol can achieve. As shown in Figure 6, in this phase of the protocol, if a parent host detects the disconnection of a child, the parent alters its protocol-related flags to reflect that change. Specifically, the parent sets the *departed-flag* variable. Similarly, if a new host becomes connected "in front of" the second wave, the parent sets its *added-flag* variable. Recording this information allows the protocol to determine what guarantee can be provided to the query issuer. For example, when a neighbor departs "in front of" the second wave of the second flood, the protocol can provide the *atomic subset* guarantee by discounting the departed host(s) and reporting the remainder of the results. *QueryReply* messages propagate back to the query issuer in a manner

```
$QueryReplyReceived_A(r)$
  Effect:
      $replies\text{-}waiting := replies\text{-}waiting - r.sender$
      $results := results \cup \{r.results\}$
      $added\text{-}flag := added\text{-}flag \vee r.added\text{-}flag$
      $departed\text{-}flag := departed\text{-}flag \vee r.departed\text{-}flag$
      if $replies\text{-}waiting = \emptyset$ then
          if $r.requester \neq id$
              send QueryReply to parent
              [reset state]
          else
              [assess query consistency]
              [deliver result to application]
          end
      end
```

Fig. 9. The *QueryReplyReceived* action

similar to *ParticipationReply* messages. The action *QueryReplyReceived* is shown in Figure 9. In this protocol, changes that occur behind the second wave of the first flood (i.e., after the query's participants are set) and before the second wave of the second flood (i.e., before the query's results are returned) can impact the query's semantics. Specifically, the following changes during this period result in the following semantics:

—***No changes***: the ATOMIC semantic can be provided.

—***Only departing participants***: the ATOMIC SUBSET semantic can be provided. If the number of departing participants can be determined (e.g., by comparing the *participants* to the *results*), the QUALIFIED SUBSET semantic can be provided.

—***Departing and adding participants***: the WEAK semantic can be provided. If the number of departing participants *and* the number of added participants can be determined then the WEAK QUALIFIED semantic can be provided.

—***Data value changes***: data value changes can be modeled as departing participants; thus, the ATOMIC SUBSET semantic can be provided. If the number of data value changes is known, the QUALIFIED SUBSET semantic can be provided.

When the last *QueryReply* message that the query issuer is waiting on arrives, the host extracts the *departed-flag* and *added-flag* values from the messages it has received. It aggregates these values and determines the query semantic that was achieved. For example, if the values of *departed-flag* and *added-flag* are both *false*, then the query issuer can determine that the query was executed with *atomic* semantics. After making this determination, the host returns the query results *and* the achieved semantic to the application.

6. IMPLEMENTATION AND EVALUATION

We have implemented a prototype of the self-assessing protocol described in Section 5 using the open source OMNeT++ discrete event simulator [Vargas 2008] and its mobility framework extension [Loebbers et al. 2008]. This protocol is capable of executing one-time queries and assessing their achieved consistency semantics. In this section, we first provide a general-purpose evaluation of our self-assessing query protocol. We demonstrate the semantics that our protocol can achieve in different situations and provide a performance characterization for the protocol's behavior. We then provide a more thorough semantic evaluation grounded in a particular application example to demonstrate how the protocol and the consistency semantics it can achieve can be used by domain programmers. The source code and settings details we used to generate these results are available at `http://mpc.ece.utexas.edu/consistency/consistency.html`.

6.1 A General-Purpose Evaluation

The first of our two simulation scenarios is general purpose. We use generic simulation settings to demonstrate the behavior of our self-assessing protocol in different conditions. Because these conditions are not grounded in a specific application scenario, we subsequently demonstrate the use of our self-assessing protocol for a particular scenario, specifically queries issued in the construction domain.

6.1.1 *Simulation Settings.* The results below were obtained from running our query execution protocol 50 times on varying numbers of nodes within a 1000x900m^2 rectangular area. Since the area size is constant, varying the number of nodes in the network (discussed below) effectively changes the network's density. The nodes move according to the random waypoint mobility model [Broch et al. 1998], in which each node is initially placed randomly in the space, chooses a random destination within that space, and moves in the direction of the destination at a given speed. Once a node reaches the destination, it pauses for a specified interval (the *pause time*) then repeats the process. Our simulations use a pause time of 0 seconds to provide more dynamicity. We used the 802.11 MAC protocol. When possible, 95% confidence intervals are shown on the graphs.

Variables. To demonstrate our protocol under different environmental and application conditions, we varied three parameters. First, the number of nodes in the network varied from 5 to 100 in multiples of 5. Second, the average speed of nodes varied from 0m/s (completely static) to 30m/s (the speed of a fast moving vehicle on a highway). Finally, we varied a *time-to-live* (TTL) parameter that restricts the scope of a query in terms of the number of hops it can travel. A TTL value of 1 indicates that a query only contacts directly connected hosts. We varied the TTL from 1 to 3; with a TTL of 3, the queried nodes were between 85-100% of the total nodes in the network. Due to space limitations, we report results only for TTL values of 3.

Metrics. We report results for several metrics. The first two categories (reported in Sections 6.1.2 and 6.1.3) demonstrate the protocol's capability of assessing a query's consistency after it has completed execution. These results show which semantics from Section 4 can be achieved under which operating conditions. The results in Section 6.1.3 pertain to the qualified semantics (i.e., QUALIFIED SUBSET and WEAK QUALIFIED), and show what percentage of the nodes contributed to the subsets when those semantics were achieved. The final metrics, reported in Section 6.1.4, evaluate trends in the protocol's performance with respect to overhead (the number of bytes transmitted to evaluate a query) and latency (the time between when a query is issued and when its result is returned). These results serve as a

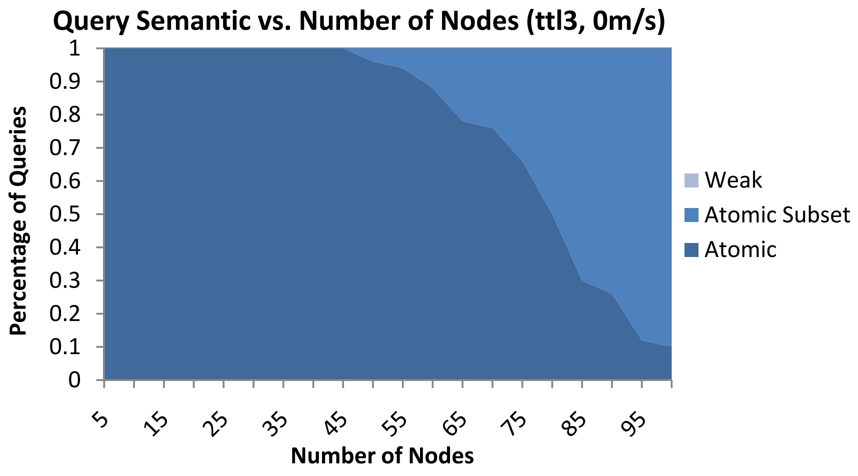

Fig. 10. Achieved query semantic vs. number of nodes in a static network.

sanity check to ensure that our protocol does not incur significant overheads or delays in reporting the semantics with the result.

6.1.2 *Reporting Query Consistency.* Because the goal of our protocol is to execute a query and deliver the results along with a report of the consistency with which the results match the execution environment, the most important aspect of our evaluation demonstrates which query semantics can be achieved under different conditions. In this section, we look at instances in which ATOMIC, ATOMIC SUBSET, and WEAK semantics can be provided. The next section looks at the qualified semantics: QUALIFIED SUBSET and WEAK QUALIFIED.

Figure 10 shows the query semantics achievable in a completely static network as the number of nodes participating in the query varies from 5 to 100. Two things are notable about this result. First, even if no mobility occurs, the ATOMIC guarantee is not achievable in all situations, especially as the number of nodes in the network grows. This is a result of increasing network density and the fact that nodes must compete to access the shared (wireless) medium. Second, in all cases, if the ATOMIC consistency cannot be achieved, at least the ATOMIC SUBSET consistency can be. This means that nodes only seem to have lost neighbors, not added any new neighbors after the query began. In fact, nodes have neither added nor lost neighbors (there is no mobility). Instead, the higher density networks suffer because nodes are competing to return their query results, making it appear as though some did not respond at all.

Figure 11 adds a small amount of mobility (5 m/s is approximately equivalent to 11mph, or a very slow moving vehicle). In this case, given a query that spans three network hops, the figure shows when each of the ATOMIC, ATOMIC SUBSET, and WEAK semantics can be achieved.

Figure 12 shows the same metric for a high degree of mobility (20 m/s). Here, the percentage of time in which the ATOMIC semantic can be achieved is even further reduced. However, in comparison to other approaches that simply fail they cannot

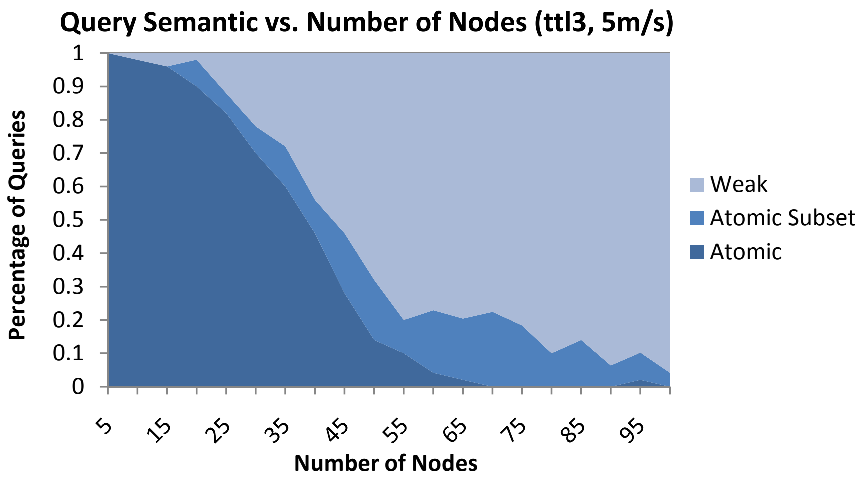

Fig. 11. Achieved query semantic vs. number of nodes in a low mobility network.

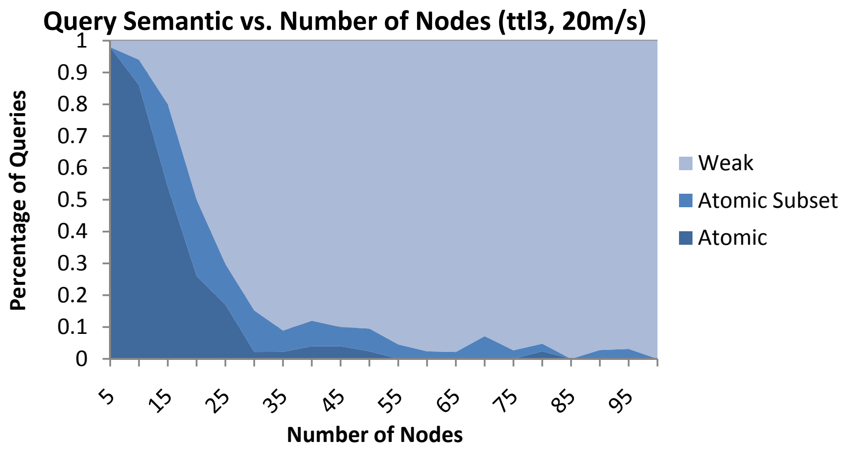

Fig. 12. Achieved query semantic vs. number of nodes in a highly dynamic network.

achieve the ATOMIC semantic, our approach can often (around 10% of the time in the 20 m/s case) still achieve some degree of atomicity *and* report a formal description of that degree of atomicity.

Figure 13 shows the effect of changing speed on the achievable query semantic. In this case, we plotted the achievable semantic as the speed varied from 5 to 30 m/s in a 30 node network. Again, the key observation is that, even in highly dynamic situations, our protocol can provide a query semantic better than best-effort more than 10% of the time. If an application developer were choosing from existing protocols, in these instances he may be forced to choose one with best-effort semantics. Using our self-assessing protocol, roughly 10% of the time, he can achieve a better guarantee.

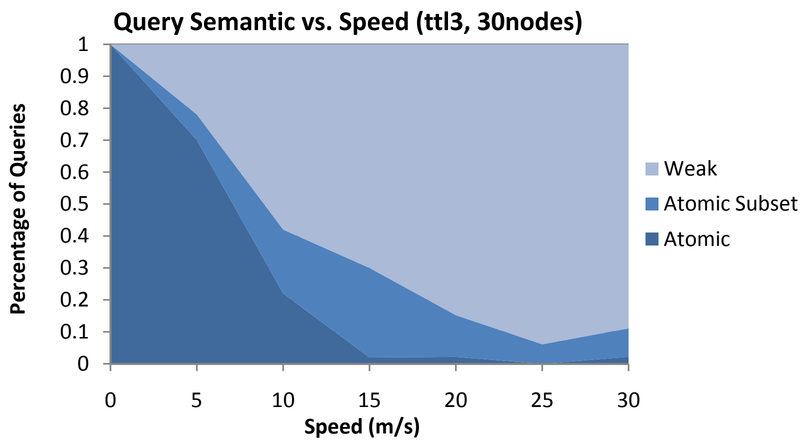

Fig. 13. Achieved query semantic vs. speed for a network of 30 nodes.

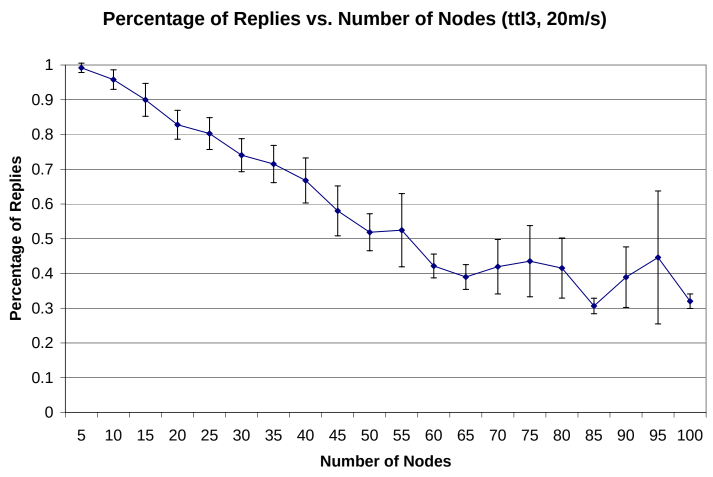

Fig. 14. Percentage of nodes replying vs. number of queried nodes for a highly dynamic network.

6.1.3 *Qualifying Query Results.* The previous section shows results only for the ATOMIC, ATOMIC SUBSET, and WEAK semantics. The two additional semantics presented in Section 4 *qualified* the ATOMIC SUBSET and WEAK semantics to further communicate to the application the degree with which the results match the execution environment. That is, QUALIFIED SUBSET and WEAK QUALIFIED both communicate the percentage of the potential responders that successfully replied to the query. Because of its design, any time our protocol can report the ATOMIC SUBSET semantic, it also has enough information to report the qualification that is part of the QUALIFIED SUBSET semantic. The same is true for the pair WEAK and WEAK QUALIFIED. Therefore, Figures 14 and 15 should be looked at in conjunction with Figures 12 and 13, respectively.

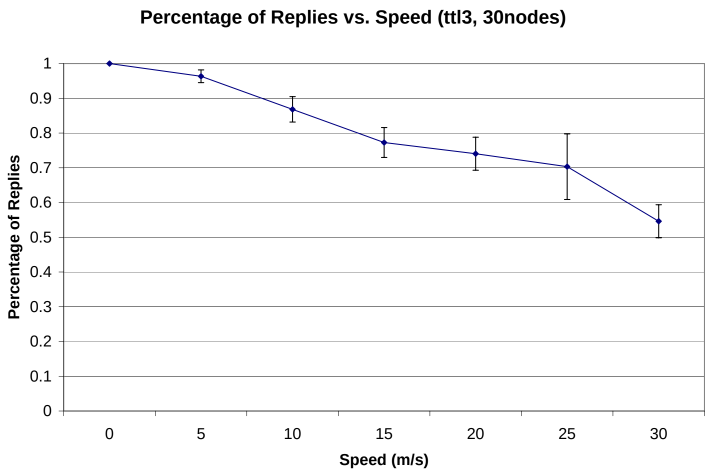

Fig. 15. Percentage of nodes replying vs. speed for a 30 node network.

Figure 14 shows that, as the number of nodes increases, the percentage successfully responding to a query decreases. In combination with Figure 12, when the query result reported has the WEAK semantic (the dark gray space in Figure 12), Figure 14 shows what percentage of the nodes it was possible to contact actually responded. For example, in a network of 85 nodes, every query had the weak semantic, and, on average, the results represented approximately 30% of the results that were available over all of the effective active configurations. Figure 15 shows a similar result: as the speed of the nodes increases, the percentage of results returned drops. The same exercise as above can be performed with the combination of Figures 13 and 15.

While the qualified semantics do not provide consistency results that are strictly stronger than the ATOMIC SUBSET and WEAK semantics, the ability to communicate the percentage of the potential query responders from which results were received provides extra beneficial information to the application, as discussed in Section 4.

6.1.4 *Protocol Performance.* Figures 16 and 17 show the performance of our self-assessing protocol as it varies with both increasing numbers of nodes and speed, respectively. We measured both the query latency (i.e., the amount of time that elapses between the application issuing the query and the results being returned to the application) and the overhead (i.e., the number of bytes sent as part of issuing the query and in control packets to maintain the network). Both the latency and overhead results show that our protocol scales well with both increasing network density (number of nodes) and average node speed. The leveling off experienced by the latency values for increased numbers of network nodes is due to the fact that, at these increased densities nodes begin to have many different paths from the query issuer, and, on average, the paths become shorter, reducing the latency to complete the query.

A next step would be to compare our protocol's performance to one that provides strong consistency semantics and to one that provides best-effort semantics. As the

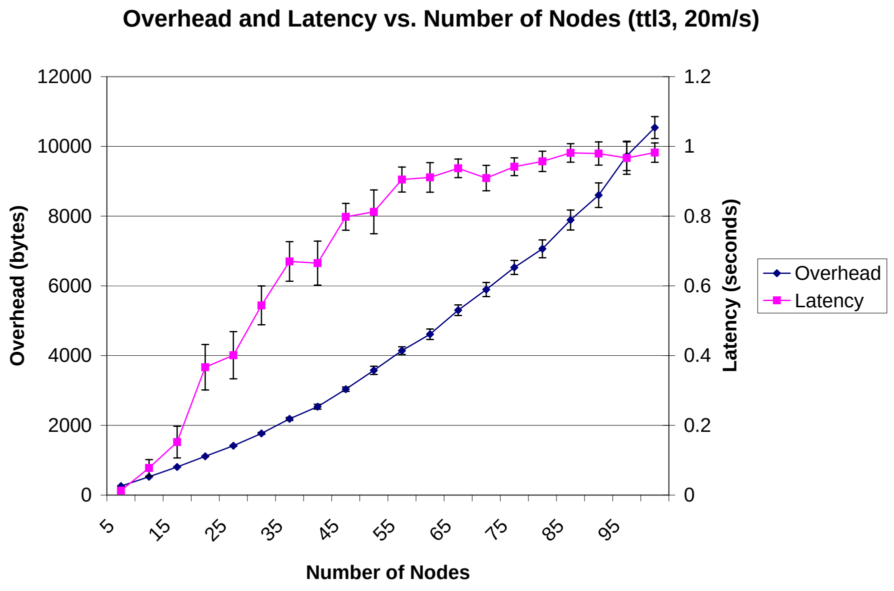

Fig. 16. Performance vs. number of nodes for a highly dynamic network.

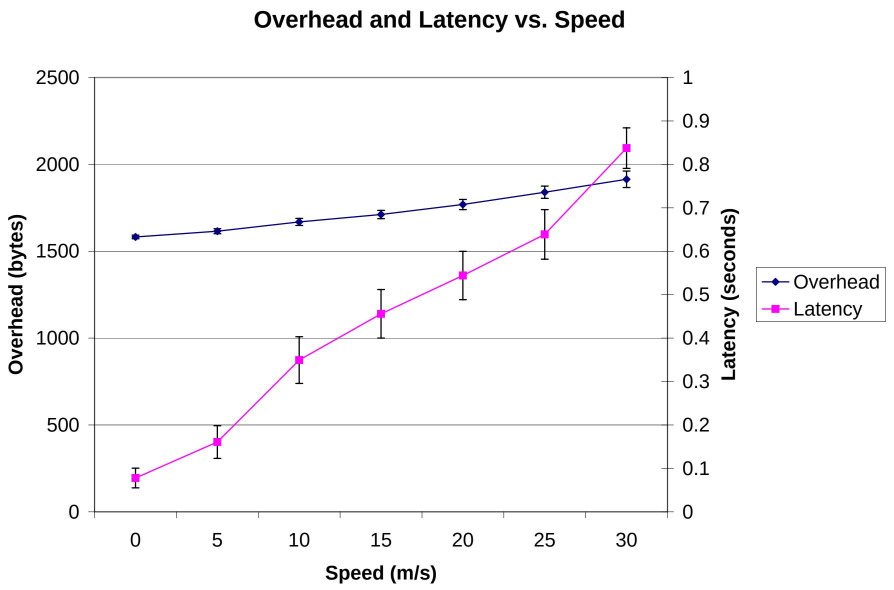

Fig. 17. Performance vs. speed for a network of 30 nodes.

focus of this paper is on the ability of the protocol to self-assess its behavior, we have omitted these results due to space and time considerations. We also plan to apply our approach to one or more specific application domains in the future.

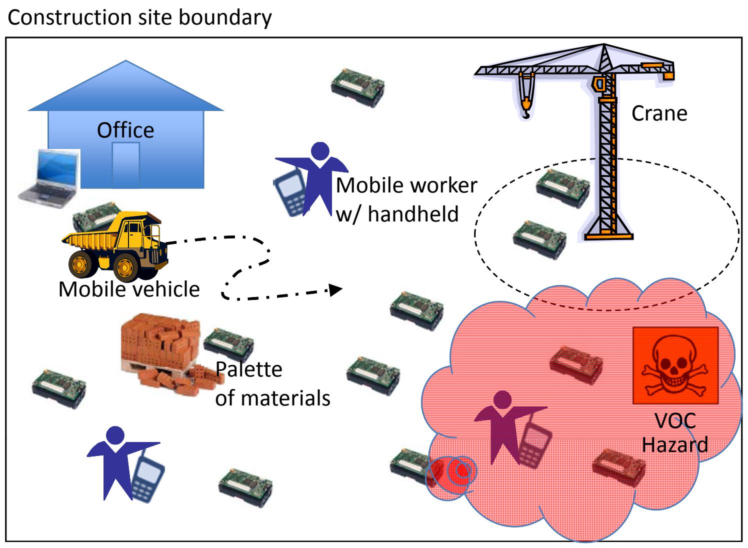

Fig. 18. Mock construction site used for scenario evaluations.

6.2 Application-Driven Simulation

To further demonstrate how an application can use the self-assessing protocol and the consistency semantics it can achieve, we demonstrate its use in a specific application domain for some different queries.

Figure 18 shows the set up for our application-driven simulation. This scenario models a construction site and components likely to be on that site. Within the scenario, we modeled construction workers (who move at walking speeds and carry handheld devices that can issue queries), mobile vehicles which may be outfitted with communication devices, sensors scattered throughout the construction site, and a crane, also outfitted with sensors. Devices in this scenario move in application-specific manners depending on the type of device. The majority of sensors scattered about the site are stationary. The device attached to the vehicle (the dump truck in Figure 18) move at vehicular speeds (in our experiments, these vehicles moved at approximately 30 mph). Devices attached to the crane moved within the circle defining the cranes swing (the dashed ellipse in Figure 18) and according to the physical movement allowed by the components of the crane. Devices carried by workers move at reasonable walking speeds (approximately 4 mph in our experiments). Devices attached to materials move rarely.

Using this scenario, we created and evaluated the consistency of two different queries representative of those likely to be issued on a construction site. The first query is issued periodically from a worker's handheld device and queries nearby sensors for the strength of a gaseous VOC (volatile organic compound) leak. The second query is issued from the (fixed) office computer, disseminated to the entire site, and returns a count of the available bricks on the job site.

Figure 19 depicts the result of issuing a query in a local area requesting the concentration of a hazardous gas. While the previous evaluations considered how

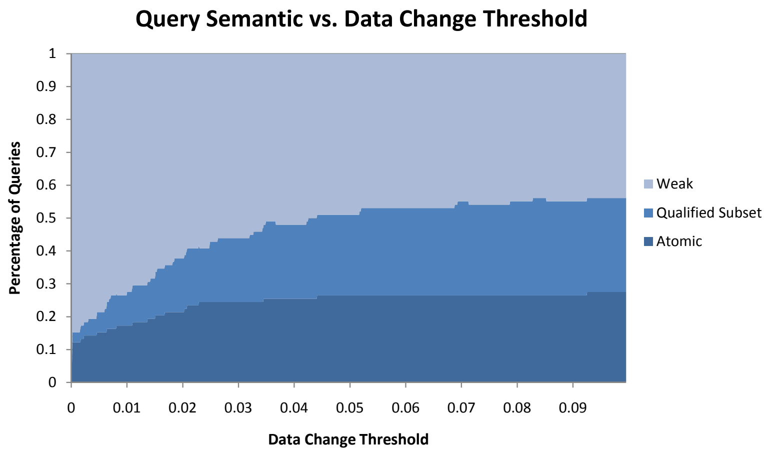

Fig. 19. Queries for hazardous conditions on the job site.

network dynamics impact the consistency of query results, this example allows us to assess the impact of changing data. In this case, the data change is exemplified by the dissipation and deterioration of the gaseous cloud. To get a sense for how the rate of changing data impacts our self-assessing protocol, we vary the sensitivity of our hazardous gas sensors. In the results shown in Figure 19, we varied the sensitivity of our sensors from 0 up to 10%. A sensitivity of $k\%$ indicates that the concentration of the sensed gas must to change by $k\%$ for a sensor to detect a change in concentration. Therefore, for the same data sets, small values for the data change threshold in Figure 19 will represent cases in which the sensors detect a high degree of changing data, while larger values represent cases in which the sensors will not detect as much change in the data. As the figure shows, the percentage of atomically consistent queries decreases as the tolerance for change decreases, but, even in highly dynamic scenarios, our protocol can still achieve a good degree of consistent queries.

Figure 20 shows the result of issuing queries from the office for available materials (in this case bricks). The query is distributed over the entire site, whose dynamics are as stated above. Because the rate of data change in this case is much slower in comparison to the movement of the devices in the network (in the depicted example, bricks are consumed at a rate of 10 per minute), the x-axis in the figure shows changing network density. As can be seen in the figure, even in this application-based example, increasing network density also results in decreased consistency. This is because the query is more wide-ranging, increasing the likelihood of network dynamics during query execution.

6.3 Applying Consistency Information to Decision Making

An essential aspect of our self-assessing query execution protocol is the ability to offer a range of query consistency semantics, providing the strongest semantics achievable at run time and delivering information about the achieved semantics with the query's result. This flexible approach to query processing can support

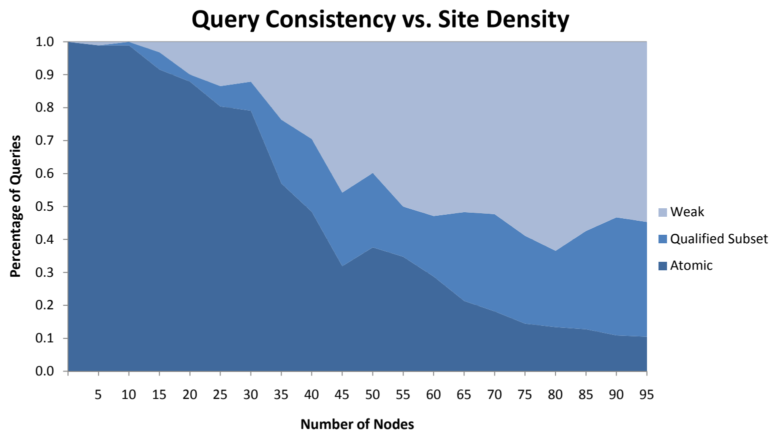

Fig. 20. Queries for available bricks on the job site.

the development of query-based applications in a way that has not been possible until now, since it provides applications with information about the query's consistency semantics that can benefit application-level decision processes. When using any other existing approach to supporting query execution in dynamic networks, an application developer must decide in advance which kind of query semantics, transactional or best-effort, will suit the application's purpose and selects a single query execution protocol that provides results with those semantics. With our self-assessing protocol, however, the application developer can avoid making an advance selection of consistency semantics; instead, the developer can use the results returned by a self-assessing query as well as their reported semantics to determine what action to take, whether it be to employ a different query strategy or to alter other application-specific behavior.

Consider, as an example, a safety monitoring application for use in the instrumented construction site introduced in Figure 18. This application would allow a construction site supervisor to check for the presence of a gaseous volatile organic compound (VOC) leak; the application will take appropriate action on behalf of the supervisor, ordering the evacuation of the construction site if necessary and initiating VOC containment procedures if possible. To implement this application, a developer can use a query to check chemical sensors distributed across the construction site and use the results to determine if there is a VOC leak present in the area. With our approach, this safety monitoring application can be designed to be adaptive and autonomous, using a decision-making strategy that is based on the query results as well as their associated semantics. Below, we outline a possible decision-making strategy for an adaptive construction safety monitoring application.

Ideally, the construction site supervisor wants to check for the possibility of a VOC leak across the entire construction site. In order to take actions that are safety-critical (e.g., issuing an "all clear" signal or ordering an evacuation of the site), the supervisor must have high confidence in the results that are returned by a query. When conditions are relatively stable in the construction site network, our query execution protocol can achieve reasonably strong consistency semantics. However, if the network is constantly changing throughout execution of the initial query, the returned query results will likely not be strongly consistent enough for use in making a sound decision. Therefore, we design the application so that it uses a query execution protocol to issue an initial query across the entire construction site, and if the result is returned with WEAK semantics, then we adapt the application's query strategy to attempt to elicit more strongly consistent results. We observe from the results presented in Section 6.1.2 that reducing the scope of a query in a highly dynamic environment can lead to more strongly consistent results; therefore, we design our application to use a divide-and-conquer strategy in which several queries are reissued by the application, each with a reduced scope that focuses on an area in the construction site that is at a high risk for VOC leaks.

Given this general strategy for adaptation, let us explore this application and its use of query results for adaptation in more detail. The supervisor's application will find one of two categories of query results: either there are no dangerous VOC readings in the area or there are. Consider the first case in which the application finds that the query does not return any dangerous VOC readings. If such a result is returned with reasonably strong semantics (i.e., ATOMIC, ATOMIC SUBSET, or QUALIFIED SUBSET), the construction site supervisor can be reasonably confident that no leak exists. Therefore, the application makes the decision to generate an "all clear" report and deliver it to the supervisor. However, if the supervisor's query results are associated with WEAK semantics, then he should not have a high level of confidence in the results; it is possible that dangerous VOC readings exist, but were missed by the application's query due to the drift of the VOC leak into the surrounding atmosphere, the mobility of network nodes, or rapid changes in the values of the VOC readings performed by chemical sensors. Since the supervisor is concerned with the safety of every person on the site, he wants more information to determine if a leak exists. Therefore, the application will make the decision to issue another query on the supervisor's behalf, reducing the scope of the query to focus on areas that are at high risk for chemical leaks in hopes of acquiring a result with stronger consistency semantics.

The adaptive construction safety monitoring application also takes action when a dangerous chemical reading is found. The application will always issue an alert to evacuate the site when a dangerous chemical reading is found. However, the consistency semantic associated with that result can also be used to determine if additional action can be taken. If the query result containing a dangerous chemical reading is associated with ATOMIC semantics, then the application can use the results returned by the query to determine the relative locations of the dangerous chemical readings since they are directly comparable; this location information can be used to begin VOC leak containment procedures. However, if the semantic associated with the query's result is not atomic, then some chemical readings may have been missed and the results are not directly comparable; more accurate information is needed to estimate the severity and location of the leak to begin containment procedures. In this case, the application will reissue the query over an area estimated to surround the initial dangerous VOC readings.

```
Protocol API
   void issueQuery(Query q, QueryListener ql, Scope s);

QueryListener Interface
   void queryCompleted(Result r, QuerySemantic qs);
```

Fig. 21. The Self-Assessing Query Protocol Programming Interface

This construction safety monitoring example demonstrates that the use of query semantics associated with the query results can aid in an application's decision-making process, through adaption of the application's query strategy or its application-specific behavior. To promote the development of such applications, we provide a simple application programming interface (API) for our query execution protocol. We believe that this API provides a simple way to incorporate the use of query results and their associated semantics in the design of an adaptive application's decision-making processes. Below, we briefly describe the programming interface applications can use to access our protocol and then show by example how this interface is exercised using the construction safety monitoring application example.

Figure 21 shows the two key elements of our API. Applications access the protocol through the `issueQuery` method. Using this method, an application can dispatch a request (encapsulated in a `Query` object whose details are omitted for brevity). Upon issuing a query, an application designates a responsible `QueryListener`, i.e., an object that has implemented the `QueryListener` interface also shown in Figure 21. The final component of an application's request is a `Scope`. This object instructs the protocol as to how widely to distribute the query. The initial implementation we have used for this paper uses a simple hop count scope restriction; more sophisticated implementations of scope restriction mechanisms [Julien et al. 2008; Kabadayi and Julien 2007] can be easily integrated into the protocol.

When the two phases of the query complete, a result is returned to the application that issued the query. This occurs through the `queryCompleted` callback in the `QueryListener` interface. The application receives two things. The first is the result of the query; in our simple initial implementation, the result is just a set of all of the values collected from the nodes queried. This could also be extended with more expressive aggregation mechanisms. The application also receives an object that encapsulates the consistency semantic associated with the query execution. To a first approximation, this query semantic is a simple enumeration; in the qualified semantics cases, this enumeration is coupled with the fraction associated with the subset semantic.

To demonstrate the use of this API, we return to our construction safety monitoring example introduced earlier in this section, providing a simple and straightforward implementation for the application. A code snippet highlighting how the construction safety monitoring application uses the query API is shown in Figure 22. The application constructs a `Query` object that asks for VOC readings from chemical sensors. When the construction site supervisor clicks a button to check for VOC leaks, the application submits this query to be executed over the entire construction site using the `issueQuery` method. A `ChemicalQueryListener`

```
public class ConstructionSafetyMonitor implements ActionListener{

  private Query q;
  private ChemicalQueryListener cql;

  ConstructionSafetyMonitor(){
    q = ..new query object that requests VOC sensor readings
    cql = new ChemicalQueryListener(this);
  }

  ...ConstructorSafteyMonitor builds a graphical user interface...
  ...a button click results in a searchForChemicals over the entire site

  public void ActionPerformed(ActionEvent ae){
    if (ae.getSource() == chemicalCheckButton) {
      Scope site = ...new scope that defines entire construction site
      searchForChemicals(site);
    }
  }

  void searchForChemicals(Scope s){
    issueQuery(q, cql, s);
  }

}
```

Fig. 22. The Construction Safety Monitoring Application

object, which embodies the strategy previously described for handling the results of a query for VOC readings, is provided as a parameter. As outlined previously, the application's decision-making processes uses the query results and their associated semantics to decide whether to reissue the query over a new scope (or scopes), issue an evacuation order, or contain a VOC leak. The code snippet for the `ChemicalQueryListener` class in Figure 23 highlights how the query consistency semantic attached to the result of a query is used to adapt the behavior of the construction safety monitoring application. As this example demonstrates, the query protocol programming interface can be used to facilitate the development of adaptive applications based on query results and their associated semantics in a simple and straightforward manner.

7. CONCLUSIONS

This work offers a new perspective on query execution in pervasive environments. The novelty of our approach lies in the ability to formally express varying degrees of consistency semantics in a dynamic ad hoc network. We have introduced several new notions of consistency and captured them using our formal model. To realize these query semantics, we have developed a self-assessing protocol that can determine the achievable consistency *during query execution* and report the assessment. Our evaluation suggests that this protocol can indeed be useful in dynamic ad hoc networks to deliver a richer, more flexible alternative to traditional transactional query processing. In addition, we have demonstrated that our approach is useful

```
public class ChemicalQueryListener implements QueryListener {
...variables and constructor omitted for brevity...

  public void queryCompleted(Result r, QuerySemantic qs){
    ChemicalResult cr = (ChemicalResult) r;
    if(!cr.containsDangerReading()) {
      //no danger found
      ...
      if(qs.getSemantic() == WEAK){
        //low confidence in result
        //need to look more closely at high risk areas
        Scope[] riskAreas = new scopes that focuses on risk areas
        for(i=0; i<riskAreas.length; i++)
         safetyMonitor.searchForChemicals(riskArea[i]);
      }
    } else {
      //dangergous reading has been found
      } if(qs.getSemantic() == ATOMIC){
         //result yields exact knowledge of VOC presence
         //evacuate site and initiate containment
         safetyMonitor.issueAlert();
         safetyMonitor.startContainment(...);
      } else{
        //alert workers, find out more about VOC presence to aid in containment
        csm.issueAlert();
        Scope interestArea = new scope focusing on area where dangerous readings were found
        csm.searchForChemicals(interestArea);
    }
  }

}
```

Fig. 23. The ChemicalQueryListener Class

to real industrial applications and can benefit their decision making processes.

Acknowledgments

This research was supported in part by ONR-MURI research contract N00014-02-1-0715 and NSF Grant CNS-0626777. C. Julien and V. Rajamani thank the Center for Excellence in Distributed and Global Environments for providing research facilities and a collaborative environment. The conclusions herein are those of the authors and do not necessarily reflect the views of the supporting parties.

REFERENCES

Barbara, D. 1999. Mobile computing and databases: A survey. *IEEE Transactions on Knowledge and Data Engineering 11,* 1 (January/February), 108–117.

Bawa, M., Gionis, A., Garcia-Molina, H., and Motwani, R. 2004. The price of validity in dynamic networks. In *Proceedings of ACM SIGMOD*. 515–526.

Broch, J., Maltz, D., Johnson, D., Hu, Y.-C., and Jetcheva, J. 1998. 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*. 85–97.

Dirckze, R. and Gruenwald, L. 1998. A toggle transaction management technique for mobile multidatabases. In *Proceedings of the 7th International Conference on Information and Knowledge Management (CIKM'98)*. 371–377.

Dunham, M., Helal, A., and Balakrishnan, S. 1997. A mobile transaction model that captures both the data and movement behavior. *ACM-Baltzer Journal on Mobile Networks and Applications 2,* 2 (October), 149–161.

Julien, C. and Roman, G.-C. 2002. Egocentric context-aware programming in ad hoc mobile environments. In *Proceedings of 10th International Symposium on the Foundations of Software Engineering*. 21–30.

Julien, C., Roman, G.-C., and Huang, Q. 2008. Sicc: Source-initiated context construction in mobile ad hoc networks. *IEEE Transactions on Mobile Computing 7*, 401–415.

Kabadayi, S. and Julien, C. 2007. A local data abstraction and communication paradign for pervasive computing. In *Proceedings of the 5th Annual IEEE International Conference on Pervasive Computing and Communications*. 57–66.

Kistler, J. and Satyanarayanan, M. 1992. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems 10,* 1 (February), 3–25.

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

Lu, Q. and Satyanarayanan, M. 1994. Isolation-only transactions for mobile computing. *Operating Systems Review 28,* 2 (April), 81–87.

Lynch, N. and Tuttle, M. 1989. An introduction to I/O automata. *CWI-Quarterly 2,* 3, 219–246.

Madria, S. and Bhargava, B. 1998. A transaction model for mobile computing. In *Proceedings of the International Database Engineering and Applications Symposium*. 92–102.

Pitoura, E. and Bhargava, B. 1995. Maintaining consistency of data in mobile distributed environments. In *Proceedings of the 15th International Conference on Distributed Computing Systems*.

Roman, G.-C., Julien, C., and Huang, Q. 2002. Network abstractions for context-aware mobile computing. In *Proceedings of 24th International Conference on Software Engineering*. 363–373.

Si, W. and Li, C. 2004. RMAC: A reliable multicast MAC protocol for wireless ad hoc networks. In *Proceedings of the International Conference on Parallel Processing*. 494–501.

Sistla, A., Wolfson, O., and Huang, Y. 1998. Minimization of communication cost through caching in mobile environments. *IEEE Transactions on Parallel and Distributed Systems 9,* 4 (April), 378–390.

Vargas, A. 2008. OMNeT++ Web Page. `http://www.omnetpp.org`.

Vellambi, B., Subramanian, R., Fekri, F., and Ammar, M. 2007. Reliable and efficient message delivery in delay tolerant networks using rateless codes. In *Proceedings of the 1st International Workshop on Mobile Opportunistic Networking*. 91–98.

Walborn, G. and Chrysanthis, P. 1999. Transaction processing in PRO-MOTION. In *Proceedings of the 1999 ACM Symposium on Applied Computing SAC'99*. 389–398.