Automatic Consistency Assessment for Query Results in Dynamic Environments

Jamie Payton
Christine Julien
Gruia-Catalin Roman

TR-UTEDGE-2007-005

© Copyright 2007
The University of Texas at Austin

Automatic Consistency Assessment for Query Results in Dynamic Environments

Jamie Payton
Department of Computer Science
University of North Carolina at Charlotte
payton@uncc.edu

Christine Julien
Department of Electrical and Computer Engineering
The University of Texas at Austin
c.julien@mail.utexas.edu

Gruia-Catalin Roman
Department of Computer Science and Engineering
Washington University in Saint Louis
roman@wustl.edu

ABSTRACT

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 the 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 state of 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.

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 and sensor 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. In military scenarios, commanders and soldiers can rely on information in the surrounding network to create a real-time operational picture or to locate support services.

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 or sensor 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 *ACID properties*. 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 sensor networks, which are often designed to provide access to streaming data, is not feasible. In both settings, attempting to strictly adhere to the ACID semantics can make 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 and sensor 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 conditions of its operating environment and adapts its execution accordingly to provide query results. An assessment of the achieved con-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

sistency 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 the result of a query.

This paper is organized as follows. Section 2 introduces our model of query execution. A range of consistency semantics is introduced in Section 3. In Section 4, we present an adaptive, self-assessing protocol for query execution that provides varying degrees of consistency; an implementation and evaluation of the protocol is discussed in Section 5. Section 6 reviews related work, while Section 7 concludes.

2. MODELING QUERY EXECUTION

In this section, we introduce a model that formally captures query execution in dynamic ad hoc networks. We can use this model to precisely define the query processing guarantees that can be offered in query in such environments. Furthermore, the model can drive the discovery of new semantics that may be beneficial in application development.

We view an ad hoc network as a closed system of hosts. We assume 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 that includes the hosts that are *reachable* from $\bar{h}$. Typically, reachability is defined in terms of physical network connectivity, which can be captured by a relation that conveys the existence of a (possibly multi-hop) network path.

We use a binary logical connectivity relation $\mathcal{K}$ to determine if a direct (one-hop) communication link exists between hosts h_1 and h_2. 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 the third field of a triple—in this case, the host's location—and d refers to a bound on the distance between two hosts to consider them connected. This statement requires that the two hosts h_1 and h_2 are within a physical distance less than d. 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. 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 $\mathcal{K}^*$ gives logical connectivity, and $\downarrow$ denotes projection.

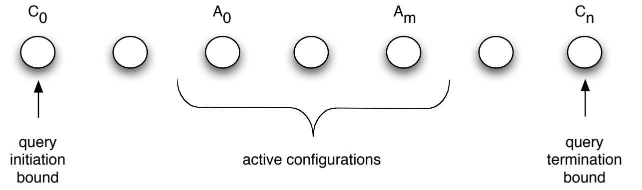

Figure 1: Query bounds and active configurations

The environment evolves as the network topology changes and value assignments occur. We model network evolution as a state transition system where the state space is the set of possible system configurations, and transitions are 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:

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

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:

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

We can now define a configuration change as:

$$\Delta C \equiv \langle value_change \oplus neighbor_change \rangle$$

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.

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 and ending in the configuration that corresponds to the delivery of the result. We call the endpoints the *query initiation bound* (C_0) and the *query termination bound* (C_n), respectively. We define $\langle C_0, C_1, \ldots C_n \rangle$ as the set of configurations over which a query is executed. No configuration before the query initiation bound or after the query termination bound impacts

the query result. Processing time at the query issuer may delay a query's submission to the network. For this reason, the set of configurations that are "active" in the query processing may be a subset of those between the query initiation bound and the query termination bound. All of the configurations, starting with the one in which the first host is interacted with and ending with the configuration in which the last host is interacted with, are the *active configurations*. The end-points of the sequence of active configurations are the *active initiation bound* (A_0) and the *active termination bound* (A_m). Every component of a query's result must be a data element that is part of some active configuration.

Since the query issuer can only interact with reachable hosts, only its effective configurations can contribute to its active configurations. We refer to this refined sequence as *effective active configurations* whose endpoints are the *effective active initiation bound* (E_0) and the *effective active termination bound* (E_m). $E_0, E_1, \ldots E_m$ is the sequence of effective active configurations that correspond to the active configurations $A_0, A_1, \ldots, A_m$, as shown in Figure 2.

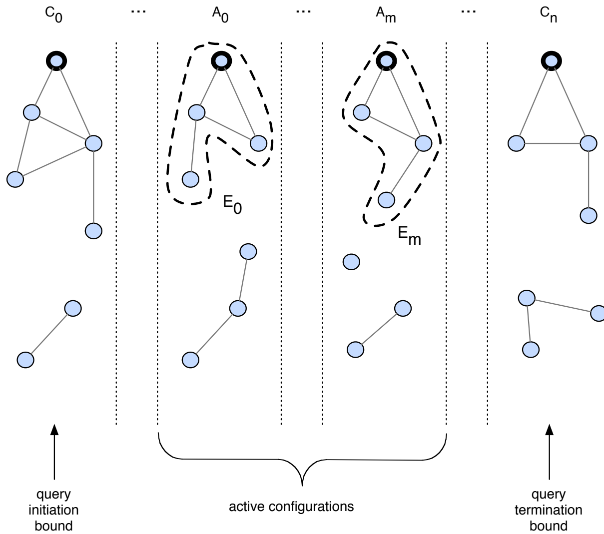

Figure 2: Effective active configurations. Circles are hosts; solid lines represent the logical connectivity relation. Dashed lines show effective active configurations.

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 directly to a straightforward expression of a query's results. In fact, the result itself is a configuration. This perspective on a query's result 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 in at least one of the effective active configurations. Second, only one query result per host should be present in the result set. We formally express these restrictions as:

$$h \in \rho \quad \Rightarrow \quad \langle \exists i : 0 \leq i \leq m :: h \in E_i \wedge \\ \langle \forall r : r \in \rho - \{h\} :: h \uparrow 1 \neq r \uparrow 1 \rangle \rangle$$

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 may be useful to application developers.

3. QUERY CONSISTENCY SEMANTICS

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 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. The first domain entails a military commander who queries an ad hoc network spanning the battlefield to request the identities and locations of assets in the network. 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.

3.1 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 military scenario, a query with IMMEDIATE semantics gives the commander a complete picture of the battlefield at the instant the query returns, allowing him to know which assets are currently present and to issue directives to those assets. 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.

3.2 Strong Guarantees: ATOMIC

Many applications require that a query result provides an exact view of the surrounding environment but may not

require that the results are still available. In our military command and control example, a sequence of results with ATOMIC semantics gives the commander a temporal picture of asset changes and locations. 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.

3.3 Partial Results: ATOMIC SUBSET

In many instances, applications may only need a certain number of resources to complete a task. A military commander may need a certain number of vehicles for a task, and a query that returns the exact relative locations of some subset of the assets available may be sufficient to complete a particular mission. 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.

3.4 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 military scenario, a query of asset locations on the battlefield gives the commander a view of a certain percentage of the available assets, 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.

3.5 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 military commander may have no significant use for weak semantics because they give him no reliable information about his assets. 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.

3.6 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 military scenario, this gives the commander some information about the relative recent availability of some assets. 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 assets, 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:

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

4. A QUERY EXECUTION AND CONSISTENCY ASSESSMENT PROTOCOL

In this section, we present a protocol that can provide any of the consistency semantics introduced in Section 3. 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 such a protocol, we demonstrate the feasibility of implementing the semantics and provide application developers with a flexible mechanism for query execution that has an underlying formal foundation for precise reasoning about query results.

4.1 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 already 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. The first phase precisely identifies the query initiation bound (as defined in Section 2), while 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 such scoping [5, 6, 13]; these solutions can easily be adapted to provide foundational execution support for our protocol.

In the remainder of this section, we provide a more detailed description of this self-assessing query execution protocol. Our protocol assumes the use of a reliable message delivery mechanism. In addition, we assume that each host can detect connection and disconnection of its neighbors using one of the aforementioned scoping approaches.

4.2 Protocol Description

The state variables used by each host participating in the query execution protocol are shown in Figure 3. The figure shows only the state for a single query execution; each query execution has a duplicate set of these variables. To define the protocol's behavior, we use I/O Automata notation [10]. 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 4) has an effect guarded by a precondition. Actions without preconditions are *input actions* triggered by another host. In the model, each action executes in a single atomic step. We abuse I/O Automata notation slightly by using, for example, "send *ParticipationRequest*(r) to *Neighbors*" to indicate a sequence of actions that triggers *ParticipationRequestReceived* on each neighbor.

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-count	– a pessimistic bound on the number of hosts departed below A in the tree
added-count	– a pessimistic bound on the number of hosts added below A in the tree

Figure 3: State Variables for Protocol

4.2.1 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 2, the first flood defines the members of the initial configuration 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 of this flood to receive acknowledgments of participation. Figure 4 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 (or leaf) 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 some other host is the parent). This action is omitted for brevity.

```
ParticipationRequestReceived_A(r)
  Effect:
    if ¬membership then
      membership := true
      parent := r.sender
      request := r
      if (neighbors − r.sender) ≠ ∅ then
        for each B ∈ (neighbors − r.sender)
          send ParticipationRequest(r) to B
          replies-waiting := neighbors − r.sender
        end
      else
        send ParticipationReply to parent
      end
    else
      send CancelParticipationRequest to r.sender
    end
```

Figure 4: The *ParticipationRequestReceived* action

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. Figure 5 shows the actions *NeighborAdded* and *NeighborDeparted* which demonstrates how our protocol handles these changes.

In both actions in Figure 5, 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, and we must simply ensure that the request propagation is handled correctly. In the case of an added neighbor, the new host is simply added to the participation request and becomes a child of this host. In the case of a departed neighbor, this host simply no longer waits for the host's reply. We will revisit the other cases in this figure 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 (or leaf) host initiates the reply process, i.e., the second wave in the first flood, by sending a *ParticipationReply* to its parent. The action handling the reception of this message is shown in Figure 6. When a host receives all of the participation replies it is waiting on, it forwards the participation reply to its parent. When it does this, it aggregates the participant information it has received and passes its parent a list of all of the participants in this subtree.

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 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 the ATOMIC semantic. At the end of the first phase, the reference host sends a *Query* to its participating neighbors to initiate the second flooding phase.

```
NeighborAdded_A(B)
  Precondition:
    connected(A, B) ∧ B ∉ neighbors
  Effect:
    neighbors := neighbors ∪ {B}
    if membership then
      if ¬monitoring ∧ (replies-waiting ≠ ∅) then
        send request to B
        replies-waiting := replies-waiting ∪{B}
      else
        added-count := added-count+1
      end
    end

NeighborDeparted_A(B)
  Precondition:
    ¬connected(A, B) ∧ B ∈ neighbors
  Effect:
    neighbors := neighbors − {B}
    if membership then
      if B = parent then
        [reset state]
      else if ¬monitoring ∧ (replies-waiting ≠ ∅) then
        replies-waiting := replies-waiting −{B}
      else if ¬monitoring then
        departed-count := departed-count+1
        participants := participants − {B}
      else if (replies-waiting ≠ ∅) then
        departed-count := departed-count+1
        replies-waiting := replies-waiting−{B}
      end
    end
```

Figure 5: Actions for handling neighbor changes

```
ParticipationReplyReceived_A(r)
  Effect:
    replies-waiting := replies-waiting−r.sender
    participants := participants ∪ {r.participants}
    if replies-waiting = ∅ then
      if r.requester ≠ id
        send ParticipationReply to parent
      else
        send Query to neighbors ∩ participants
      end
    end
```

Figure 6: The *ParticipationReplyReceived* action

4.2.2 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 7. 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. Leaf hosts initiate the second wave of the second flood to deliver query results. In constructing a query result, a leaf includes its own data value and its *departed-count* and *added-count* variables. As these replies propagate

```
QueryReceived(q)
  Effect:
    if membership ∧ ¬monitoring
      ∧q.sender = parent then
       monitoring := true
       if participants ≠ ∅ then
         replies-waiting := participants)
         send Query to neighbors ∩ participants
       else
         send QueryReply to parent
         [reset state]
       end
    end
```

Figure 7: The ***QueryReceived*** **action**

up the tree, parents aggregate the results and counts 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 of the protocol, the setting of the *monitoring* flag and checking for changes in data during query execution is analogous to the used 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 in turn will impact what consistency semantic the protocol can achieve. As shown in Figure 5, 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 increments the *departed-count* variable. Similarly, if a new host becomes connected "in front of" the second wave, the parent increments its *added-count* variable. Recording this information allows the protocol to determine what guarantee can be provided to the query issuer. For example, in the case where 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 and reporting the remainder of the results. *QueryReply* messages propagate back to the query issuer in a manner similar to the propagation of *ParticipationReply* messages. The action *QueryReplyReceived* is shown in Figure 8.

```
QueryReplyReceived_A(r)
  Effect:
    replies-waiting := replies-waiting−r.sender
    results := results ∪ {r.results}
    added-count := added-count + r.added-count
    departed-count := departed-count + r.departed-count
    if replies-waiting = ∅ then
      if r.requester ≠ id
        send QueryReply to parent
        [reset state]
      else
        [assess query consistency]
        [deliver result to application]
      end
    end
```

Figure 8: The ***QueryReplyReceived*** **action**

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 resulting 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., using *departed-count*), 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 (e.g., using *departed-count* and *added-count*, respectively), 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-count* and *added-count* values from the messages it has received. It aggregates across the messages it has received and determines the query semantic that was achieved. For example, if the values of *departed-count* and *added-count* are both 0, then the query issuer can determine that the query was executed with *atomic* semantics. After making this determination, the host then returns the query results *and* the achieved semantic to the application.

5. A REFERENCE IMPLEMENTATION

We have implemented a prototype of the self-assessing protocol described in Section 4 using the open source OMNeT++ discrete event simulator [15] and its mobility framework extension [8]. Here, we demonstrate the semantics that our protocol can achieve in different situations and provide a performance characterization for the protocol's behavior.

5.1 Simulation Settings

The results below were obtained from running our protocol 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 [2], 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. In all of our simulations, we 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 participating in the network varied from 5 to 100 in multiples of 5. Second, the average speed of nodes in the network varied from 0m/s (completely static) to 30m/s (the speed of a fast moving vehicle on a highway). Finally, we also varied a *time-to-live* (TTL) parameter that restricts the scope of query in terms of the number of hops it could travel. A TTL value of 1 indicates that a query only contacts directly connected neighbors. We varied the TTL from 1 to 3; at a TTL value

of 3, the queried nodes were between 85-100% of the total nodes in the network. Due to space limitations, we report results below only for TTL values of 3.

Metrics. We report results for several metrics. The first two categories (reported in Sections 5.2 and 5.3) demonstrate the protocol's capability of assessing a query's consistency after it has completed execution. These results show which semantics from Section 3 can be achieved under which operating conditions. The results in Section 5.3 specifically look at the qualified semantics (i.e., QUALIFIED SUBSET and WEAK QUALIFIED), and they show what percentage of the nodes contributed to the subsets when those semantics were achieved. The final metrics, reported in Section 5.4, evaluate trends in the protocol's performance with respect to the overhead (the number of bytes transmitted to evaluate a query) and the latency (the time between when a query is issued and when its result is returned). These results serve as a sanity check to ensure that our protocol does not incur significant overheads or delays in reporting the semantics with the result.

5.2 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 9 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. The first is that, 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 the increasing network density and the fact that the nodes must compete to access the shared (wireless) medium. The second notable aspect is that, 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, after all), but instead the networks with higher density suffer because nodes are competing to return their query results, making it appear as though some did not respond at all.

Figure 10 shows the same metric for a high degree of mobility (20 m/s). Here, the percentage of the time in which the ATOMIC semantic can be achieved is even further reduced. However, in comparison to other approaches that simply fail with they cannot 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.

The final chart for this metric, Figure 11, demonstrates the effect of changing speed on the achievable query semantic. In this case, we looked closer at a 30 node network, and plotted the achievable semantic as the speed varied from 5 to 30 m/s. Again, the key observation is that, even in highly dynamic situations, more than 10% of the time, our protocol can provide a query semantic better than best-effort. If the application developer was choosing from existing protocols, in these instances he may be forced to choose one with best-effort semantics. If he instead uses our self-assessing protocol, approximately 10% of the time, he can achieve a better guarantee.

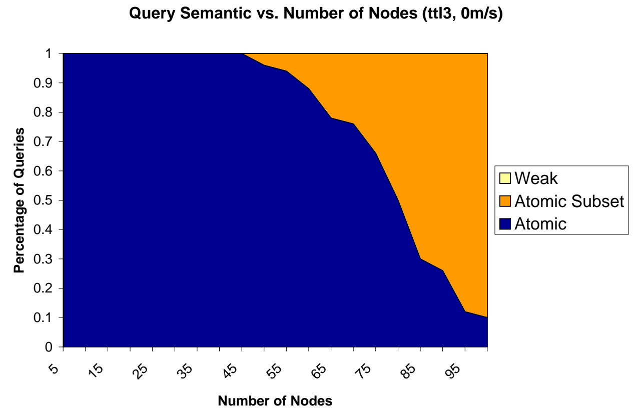

Figure 9: Achieved query semantic vs. number of nodes in a static network.

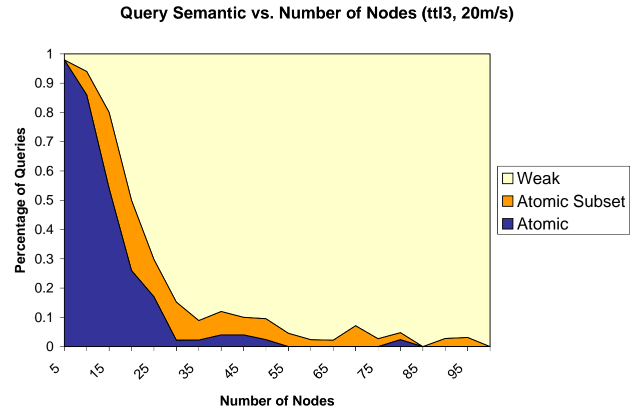

Figure 10: Achieved query semantic vs. number of nodes in a highly dynamic network.

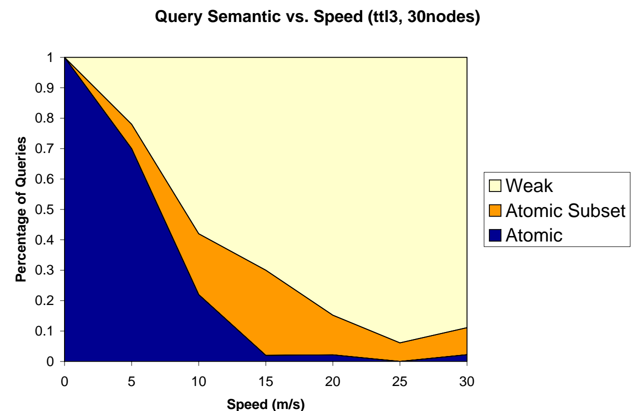

Figure 11: Achieved query semantic vs. speed for a network of 30 nodes.

5.3 Qualifying Query Results

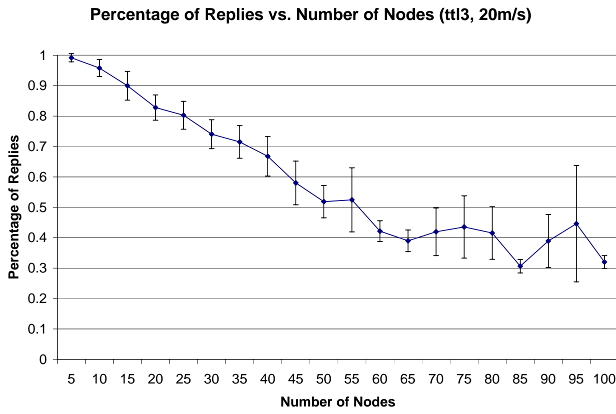

Figure 12: Percentage of nodes replying vs. number of queried nodes for a highly dynamic network.

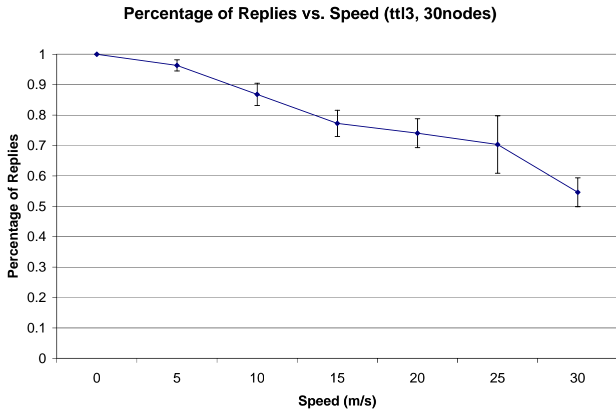

Figure 13: Percentage of nodes replying vs. speed for a 30 node network.

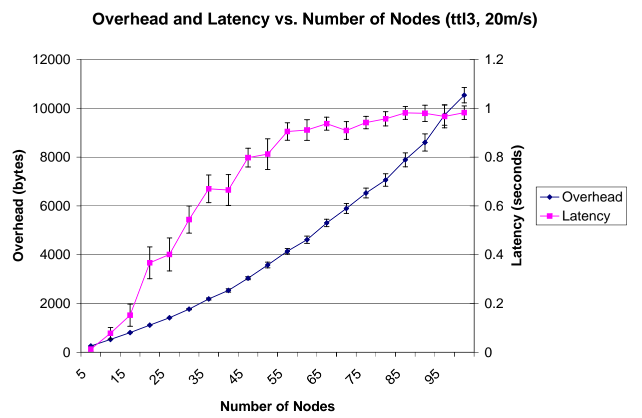

Figure 14: Performance vs. number of nodes for a highly dynamic network.

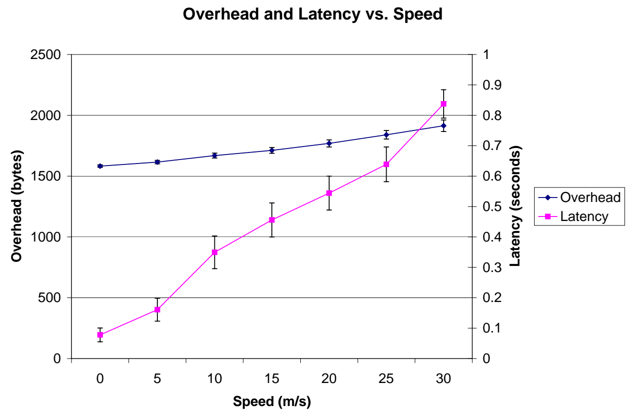

Figure 15: Performance vs. speed for a network of 30 nodes.

The previous section shows results only for the ATOMIC, ATOMIC SUBSET, and WEAK semantics. The two additional semantics presented in Section 3 *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 12 and 13 should be looked at in conjunction with Figures 10 and 11, respectively.

Figure 12 shows that, as the number of nodes increases, the percentage successfully responding to a query also decreases. In combination with Figure 10, when the query result reported has the WEAK semantic (the dark gray space in Figure 10), Figure 12 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 13 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 11 and 13.

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 3.

5.4 Protocol Performance

Figures 14 and 15 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 final step in the evaluation of this protocol would be to compare its performance metrics to a protocol that provides strong consistency semantics and to a protocol that provides best-effort semantics. As the focus of this paper is on the ability of the protocol to self-assess its behavior, we have omitted these results for space and time considerations.

6. RELATED WORK

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 [1]. Several of these strategies focus on issues related to dynamic cache allocation [14] or optimistic replication [7], while others allow applications to explicitly issue *weak operations* that are allowed to operate over potentially inconsistent data [12]. 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 [4, 16] 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 [9] ensure only that committed transactions appear as though executed independently; toggle transactions [3] enable extended execution, relaxing both atomicity and isolation; and the pre-write transaction model [11] focuses almost exclusively on data-availability. These models are generally limited to use in nomadic networks.Because of their reliance on powerful and fixed nodes on the fringe of the network, these weakened transactional models cannot be applied to ad hoc networks. Moreover, the frequent disconnections and reconnections in a mobile ad hoc network could result in significant overhead when employing similar approaches.

7. CONCLUSIONS

This work offers a new perspective on query execution in pervasive computing 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 initial evaluation suggests that this protocol can indeed be useful in dynamic ad hoc networks to deliver a richer, more flexible alternative to traditional transaction query processing.

Acknowledgments

This research was supported in part by ONR-MURI research contract N00014-02-1-0715. C. Julien thanks the Center for Excellence in Distributed 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 supporting parties.

8. REFERENCES

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

[2] 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 *Proc. of the ACM/IEEE MobiCom*, pages 85–97, October 1998.

[3] R. Dirckze and L. Gruenwald. A toggle transaction management technique for mobile multidatabases. In *Proc. of the 7th Int'l. Conf. on Information and Knowledge Management*, pages 371–377, 1998.

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

[5] C. Julien and G.-C. Roman. Egocentric context-aware programming in ad hoc mobile environments. In *Proc. of 10th Int'l Symp. on the Foundations of Software Engineering*, pages 21–30, Nov. 2002.

[6] S. Kabadayi and C. Julien. A local data abstraction and communication paradign for pervasive computing. In *Proc. of the 5th IEEE Int'l. Conf. on Pervasive Computing and Comm.*, 2007. (to appear).

[7] J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Trans. on Computer Sys.*, 10(1):3–25, February 1992.

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

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

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

[11] S. Madria and B. Bhargava. A transaction model for mobile computing. In *Proc. of the Int'l. Database Eng. and Apps. Symp.*, pages 92–102, July 1998.

[12] E. Pitoura and B. Bhargava. Maintaining consistency of data in mobile distributed environments. In *Proc. of the 15th Int'l. Conf. on Distributed Computing Systems*, 1995.

[13] G.-C. Roman, C. Julien, and Q. Huang. Network abstractions for context-aware mobile computing. In *Proc. of 24th Int'l Conf. on Software Engineering*, pages 363–373, 2002.

[14] A. Sistla, O. Wolfson, and Y. Huang. Minimization of communication cost through caching in mobile environments. *IEEE Trans. on Parallel and Dist. Sys.*, 9(4):378–390, April 1998.

[15] A. Vargas. OMNeT++ Web Page. `http://www.omnetpp.org`.

[16] G. Walborn and P. Chrysanthis. Transaction processing in PRO-MOTION. In *Proc. of the ACM Symp. on Applied Computing*, pages 389–398, 1999.