ADAPTIVE NETWORK PROTOCOLS TO SUPPORT QUERIES IN DYNAMIC NETWORKS

by

Dingxiang Liu

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2011

ABSTRACT

DINGXIANG LIU. Adaptive network protocols to support queries in dynamic networks. (Under the direction of DR. JAMIE PAYTON)

Recent technological advancements have led to the popularity of mobile devices, which can dynamically form wireless networks. In order to discover and obtain distributed information, queries are widely used by applications in opportunistically formed mobile networks. Given the popularity of this approach, application developers can choose from a number of implementations of query processing protocols to support the distributed execution of a query over the network. However, different inquiry strategies (i.e., the query processing protocol and associated parameters used to execute a query) have different tradeoffs between the quality of the query's result and the cost required for execution under different operating conditions. The application developer's choice of inquiry strategy is important to meet the application's needs while considering the limited resources of the mobile devices that form the network. We propose adaptive approaches to choose the most appropriate inquiry strategy in dynamic mobile environments. We introduce an architecture for adaptive queries which employs knowledge about the current state of the dynamic mobile network and the history of previous query results to learn the most appropriate inquiry strategy to balance quality and cost tradeoffs in a given setting, and use this information to dynamically adapt the continuous query's execution.

ACKNOWLEDGMENTS

First of all, I would like to express my deepest appreciation to my Ph.D. advisor Dr. Jamie Payton. As my advisor, she has provided me continuous encouragement and insightful suggestions which are invaluable for me to go through the challenges of research. Her excellent expertise and advice played an important role in this dissertation and her vision has led me throughout my graduate research. This dissertation would not be possible without her guidance and encouragement.

I would also like to express my sincere thanks to Dr. Teresa Dahlberg, Dr. Yu Wang, and Dr. Jun-tao Guo for serving on advisory committee.

Next, many thanks go to former and current members of Networking Research Laboratory. Thanks for the discussions and help.

Finally, I also deeply thank my parents for their endless love and encouragement in my study. My special gratitude goes to my wife who provided tremendous support, understanding, and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

The widespread availability and adoption of mobile computing devices has resulted in new approach to the way that we interact with digital information. The availability of small, battery-operated sensors with environmental sensing and wireless communication capabilities has provided us with a relatively low-cost mechanism to monitor intruders, study wildlife, and provide emergency rescue services [78, 81]. The evolution of PDAs, smart phones, and touch tablets and wireless communication technologies have prompted researchers to imagine pervasive computing applications that take advantage of information shared by people and devices within their surroundings.

A wide array of pervasive computing applications can be supported through the use of mobile ad-hoc networks. A mobile ad-hoc network (MANET) [30, 37] is a collection of autonomous mobile hosts equipped with wireless communication capabilities that form opportunistic network connections as mobile devices come within communication range of one another. Mobile nodes are free to move arbitrarily in any direction and organize themselves in an arbitrary fashion, resulting a highly dynamic network topology. In a MANET, there is no centralized administration of the network. Instead, every node serves as a router, and the transitive nature of communication is leveraged to route a message across multiple nodes to its final destination.

Specifically, when a node in a MANET transmits a message, the message is received by all hosts within its transmission range due to the broadcast nature of wireless communication and omni-directional antennae. We call these "one-hop" neighbors of the sending node. For our work, we assume a bi-directional communication model [97], in which the one-hop relationship is symmetric, which means that if node A can send a message to node

B, then node B can send a message to node A. To enable communication across a network, ad hoc routing protocols have been developed that rely on the transitive nature of communication to route messages along a path of one-hop neighbors. Figure 1.1 illustrates how communication of a message works in a MANET. The colored circles represent the transmission range of each node and the connections between nodes denote wireless connectivity between the nodes. For example, node B is in the transmission range of node A and they can exchange messages directly. If node A wants to send a message to node C, an ad hoc routing protocol treats all intermediate network nodes along a path as routers, enabling node A to send a message through node B to reach node C.

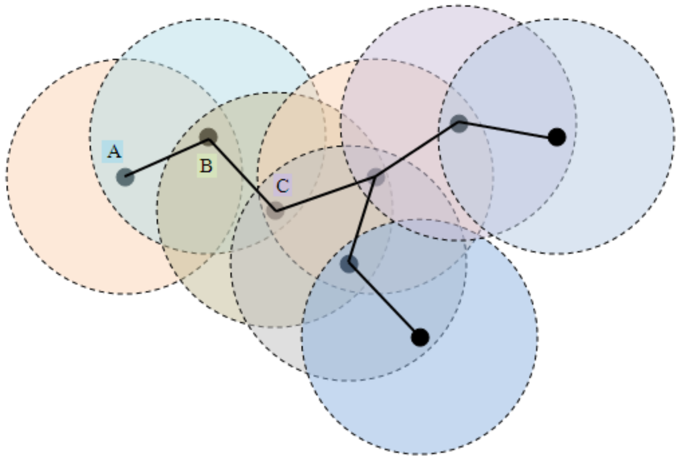


Figure 1.1: Multi-hop Communication in a MANET

Given these characteristics, mobile ad-hoc networks can be set up at any place and time without any established infrastructure, which makes these networks highly flexible. A number of pervasive computing application scenarios for mobile ad-hoc networks have been envisioned and implemented:

Tactical networks. Historically, ad-hoc wireless networks have primarily been used for military related applications [36, 49] due to the dynamic nature of military communications and operations. For example, small mobile devices can be deployed in inhospitable terrain to collect important location and environmental data for surveillance of the enemy and to share operational information. Furthermore, soldiers equipped with mobile devices can communicate with each other without reliance on access to a fixed communication infrastructure to help them navigate in battlefields.

Emergency rescue services. Efficient communication and collaboration between teams from different organizations is important for the successful rescue and recovery in disasters and emergency scenarios. However, the entire communication infrastructure may be destroyed by natural disasters (e.g., earthquake). The ad-hoc networking technology is a desirable approach for the replacement of fixed infrastructure and forms a network quickly [45, 51].

Vehicular networks. In vehicular service applications [83, 84, 91], a Vehicular Ad-hoc NETworks (VANET) is formed by considering moving vehicles as participating nodes in a network. Similar to MANET, each participating vehicle can be treated as a wireless router or node. Therefore, a transportable internet is established which achieves intelligent inter-vehicle communications and results in improved road safety and traffic navigation.

Despite domain-specific and device differences, the above applications share an essential trait: the need to collect information from a set of networked nodes in a distributed fashion. Query processing is a promising approach to acquire information from such networks [10, 53, 55, 99], because it hides the complex network communication details associated with identifying data owners and using multi-hop network communication protocols to acquire their information in a highly dynamic network. Different query processing protocols provide different guarantees about the quality of results, and have different costs associated with them in terms of message overhead and therefore energy consumption. Rather than requiring application developers to choose a single query protocol that balances this quality versus cost tradeoff, we propose an approach that learns the optimal protocol and parameters to apply for the current operating conditions in terms of quality and cost, and uses this knowledge to dynamically adapt the query.

The remainder of this dissertation is organized as follows. Chapter 2 is a literature review of related work, introducing query processing approaches in classical database systems, as well as distributed databases. In chapter 3, we introduce the motivation of study which we work on and briefly present our adaptation strategies for query processing. Then, we illustrate regression models to provide fidelity-based adaptation strategies for continuous queries in chapter 4. Chapter 5 describes an adaptation approach which is used to take a perspective on learning dynamic changes and supporting adaptive query strategy in dynamic environments. In chapter 6, we provide automatic adaptation component which implements the proposed adaptation algorithms in order to simplify the application development tasks. In the final chapter, we summary the proposed approaches and conclude the study work.

CHAPTER 2: RELATED WORK

Classical database systems are known for their ability to process queries efficiently. In a traditional database management system (DBMS) [33], the data structures and actual data are stored in permanently centralized storage. The users are shielded from the details of physical data storage and a logical query language, such as SQL [34, 57], which is used for querying the database. In most cases, the whole query processing has several steps [39]. First, user queries are submitted to the database management systems. After receiving the queries, a query processor converts those queries into system readable contexts, such as algebra expressions, query trees, and query graphs, which can be understood by the query processing engine. Then, the query optimization engine applies related rules to translate the those contexts into more efficient representations in order to improve the query performance. Finally, the execution engine handles those representations to answer the queries.

Since network technology has matured businesses with geographically dispersed facilities, more and more work relies on distributed data processing approaches. Therefore, another research trend in database systems is distributed database system (DDBS) [7, 48, 65], which is the integration of a distributed database (DDB) and a distributed database management system (DDBMS) [74]. A DDB is a collection of multiple interrelated databases distributed over a computer network while DDBMS is the software that manages the DDB. Such systems are aimed at handling several geographically separated data stores, which are connected by a communication network. Replication of data makes the entire system more robust to failures and can improve database reliability. However, this architecture has an impact on how the data can be maintained effectively. For example, the database management will become more complex and extra approaches should be designed in order to maintain consistency across replicated distributed data sources, especially in the face of network failure.

Due to the convergence of portable devices and wireless communication networks, the research community has investigated mobile databases [3, 85]. With mobile database systems, the users of the system are mobile while the database itself is on a wired network. The mobile users can disconnect from the wired network, and still work in "disconnected" mode on their mobile devices, and the database system will be synchronized upon reconnection. Although the development and deployment of mobile databases brings more convenient accessibility of data, mobile database systems have unique challenges. The first is to ensure the data availability for mobile clients. The second is guaranteeing the consistency of replicas for collaborative work and while considering the limited memory available in mobile devices. The third challenge is recovering from terminated transactions because mobile devices do not stay connected to the network continuously and they may move and disconnect from the previous ones. Previous research has made progress to address these challenges. For example, four per-session guarantees were described in [90] to provide data consistency for mobile clients. However, these approaches assume that a mobile device temporarily disconnects from a wired network only to reconnect later, and disconnection is the exception, which is not the case in a MANET. In addition, solutions for mobile databases rely on the use of a centralized server and assume the use of resource-rich nodes, making such solutions unsuitable for MANETs.

The idea of mobile databases has since inspired research on distributed query processors for data acquisition in opportunistically formed sensor networks, which share many of the same characteristics as mobile ad-hoc networks. In many of the early approaches [10, 53, 55, 99], the entire sensor network is viewed as a single database table and users are able to issue SQL-style queries using a connection to this resource-rich node to acquire information. There are two types of queries: snapshot queries and continuous queries. A snapshot query is expected to execute through the network and return values that satisfy the query criteria at a particular instant in time. A continuous query delivers a streaming result that reflects changes to data items meeting the query's criteria over time. Continuous queries are frequently used in mobile ad-hoc networks and sensor networks to monitor changing conditions of the surrounding environment. We focus on continuous queries in this dissertation.

In general, two approaches have been introduced to realizing the use of continuous queries for data acquisition in sensor networks: push-based and pull-based continuous queries. In a push-based continuous query model, nodes push the data every time when the value of a store attribute is changed (e.g., in a sensor network, a reading of an environmental phenomenon results in the generation of a new sample) to a collector node (i.e. a base station) that puts it in a centralized database on the wired network. Queries are then executed over this database. In this approach, a centralized database is maintained and used to resolve queries. When environments are highly dynamic, a large number of updated values must be propagated across a network and stored in the database; this has a significant cost in terms of network communication and energy consumption on the resource-constrained devices. Conversely, in the pull-based model, nodes generate readings, but data is "pulled" on-demand by a query that is disseminated using a distributed protocol to all available nodes, which execute the query against local data and send the results back to the query issuer across the multi-hop network.

Well-known (and widely used) query processing systems like Cougar [99], TinyDB [55], and TAG [53] embody the pull-based model to query processing. In [10], Bonnet et al. introduced the notion of viewing a wireless sensor network as a database, and the initial version of the Cougar system was described. The Cougar system [99] attempts to borrow approaches from central warehousing and apply them to a distributed sensor network. In Cougar, a pull-based model of continuous query processing is used; a query is issued with at a specified periodic rate. TAG and its extended version, TinyDB, also use a pull-based model of continuous query processing. TinyDB is a widely used acquisitional query processing framework that aims to reduce power consumption, and to give greater control over where, when, and in what order the sensor nodes are interrogated as part of the execution of a pull-based query. Users of TinyDB submit queries via a powered base station, which optimizes the query to reduce power consumption and then sends the query to the network of sensor nodes. TinyDB then employs a simple in-network aggregation technique to collect responses in sensor networks, storing them at a centralized database.

The above solutions have been successfully applied for use in certain applications for sensor networks. However, these methods require a query issuer interact with a collector that is known in advance and reachable at any instant, which is often unreasonable in a mobile ad-hoc network. Furthermore, the solutions for distributing queries and replies are targeted for static sensor networks, and are not appropriate for use in mobile ad-hoc networks in which the network topology is highly dynamic. Finally, those approaches do not provide adaptive mechanisms to adapt the query strategies, which is important because the conditions in mobile ad-hoc network are constantly evolving. For example, nodes may initially be moving quickly, but may begin to move slowly (e.g., cars on a highway that encounter rush hour traffic), which may mean that the periodic issue of the pull-based continuous query should be updated to acquire information as the data changes more slowly.

The work in this dissertation focuses on the use of pull-based adaptive continuous queries for mobile ad-hoc networks. In this dissertation, we approximate a continuous query as a sequence of snapshot queries. Any node in the network is able to issue a query. There is no centralized collector or storage node. Snapshot queries are assumed to be issued at a periodic rate over the data currently available in the network, and the results are returned over multi-hop paths to the query issuer.

We adapt the model of approximated continuous queries introduced in [75], which defines a continuous query as a sequence of snapshot shot queries issued issued using an *inquiry strategy*, which defines the snapshot query protocol used to issue the query and the rate at which the snapshot queries are issued. As noted by Rajamani et al. [75], an important concern in choosing the inquiry strategy is the the tradeoff between the quality of query processing achieved by a particular inquiry strategy and its associated execution cost. There are some works on measuring quality of query processing in wireless networks. In [79], quality is defined as the confidence of the accuracy of a query result. The confidence is given as the probability of satisfying values located within the acquired precision bound. In [98], authors introduce a quality metric that describes the response time for a query's processing. This dissertation presents two definitions of quality that can be used to assess the appropriateness of an inquiry strategy to support a continuous query's execution. The first, defined in chapter 4, uses a definition of quality of snapshot queries to define the quality of the continuous query. In chapter 5, we provide a second definition of the quality of query result that attempts to measure how well the approximate persistent query reflects the ideal query result.

A query's environment changes over time, and query processing should adapt to these changes [25]. Early works [2, 54] on adaptive continuous query processing typically focused on optimization of the ordering of query operators. In [2], StreaMon is an adaptive engine to automatically adapt join orders for input streams in response to evolving conditions. The adaptive join algorithm is called Multi-way streaming Join (MJoin) [93], which produces a faster output rate by joining multiple inputs at the same time. This approach is used to deal with more than two inputs in streaming environments. Continuously adaptive continuous queries (CACQ) [54] uses eddy query processing framework [1] to achieve continuous adaptivity by changing the order of query operations. None of the above approaches provides general support for dynamically adapting a continuous query based on application requirements in terms of quality and cost of the query's execution.

Recent work [77] indicates the need for adaptive continuous queries and studies the problem of how to adapt continuous query execution. The method in [77] contains two main parts: the use of an introspection strategy to determine the quality of execution and an adaptation strategy that describes how to adapt execution. Both are based on application-specified data quality metrics and thresholds to adapt the query protocols. According to the results of applying the introspection strategy to a history of a continuous query's result, the adaptation is triggered if the value of introspection is not satisfactory. The activated adaptation policies change the continuous query's execution in an attempt to meet the application's needs in the current operational environments. For this approach, a significant amount of knowledge about the relationships between inquiry strategies and the nature of the environment is necessary for application developers in order to create adaptation rules that satisfy the application's requirements in terms of cost and query quality.

The goal of this dissertation is to provide adaptive continuous queries that achieve the optimal tradeoff between quality and cost of execution. In this dissertation, we introduce a general approach to application-transparent adaptation that employs machine learning algorithms to determine $when$ and how to adjust the suitable inquiry strategy for dynamic conditions. In the next chapter, we firstly describe the motivation of adaptive query processing in dynamic pervasive computing environments. Then the corresponding adaptation strategies are introduced in chapter 4 and chapter 5, which learn the properties of operational environments and adapt inquiry strategies to match the evolving conditions.

CHAPTER 3: ADAPTIVE QUERY PROTOCOLS

Queries are a popular abstraction used to discover and collect information from a wireless network. There are two types of queries in wireless sensor and mobile ad-hoc networks: one-time snapshot queries and continuous queries. A one-time snapshot query executes through the network and returns data values from some or all of the nodes that satisfy the application query criteria at a particular time. For example, a driver can find available parking space in a parking lot by collecting parking information from all sensors which monitor the parking situation. The snapshot query is suitable in this application because the driver needs the information at an exact time.

At the logical level, a continuous query is a long-lived operation which provides a continuously updated view of all changes that occur in a dynamic network over time. For some applications, continuous queries are necessary. For example, in habitat monitoring applications, a continuous query may be used to track the path of an animal as it moves through an area. Building such applications with continuous query is costly. Constant monitoring requires the construction and maintenance of a distributed data structure, which can be expensive in terms of communication cost when the environment is highly dynamic. To address this issue, a framework has been proposed in [40] to express a continuous query as a sequence of snapshot queries; the continuous query's result can be viewed as the integration of the snapshot queries' results. We adopt this view in our work on adaptive continuous query processing, and review the model below.

3.1 A Review of Approximated Continuous Queries

In [40], a continuous query $\mathcal{C}$ is modeled as a sequence of non-overlapping snapshot queries, whose results are denoted as $\langle \rho_0, \rho_1, \ldots, \rho_n \rangle$. A continuous query's *inquiry strat-*

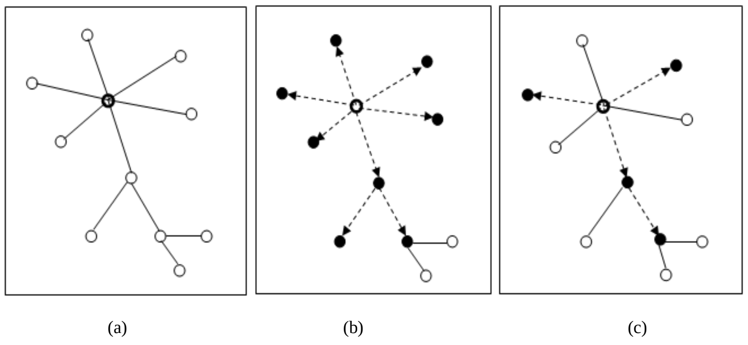


Figure 3.1: Query protocols. The query issuer is depicted with a darker boundary. (a) A sample network. (b) A flooding query protocol. (c) A probabilistic protocol (50%).

egy defines the query processing protocol used to execute a single snapshot query and the pattern of invocations of the snapshot query (e.g., frequency of issue and parameters of the snapshot query protocol). Individual snapshot queries can be processed in many different ways, and the different styles of communication employed by the different protocols can result in radically different qualities of query results. The result of a continuous query at stage i (i.e., including ρ_i, which is the result of the i^{th} component snapshot query q_i), is obtained by applying an *integration strategy* over a windowed history of previously issued snapshot query results. Integration strategy defines how to combine the results of component snapshot query results into a continuous query result. Below, we review details originally presented in [40] about inquiry strategies and integration strategies that will motivate and be relevant to our approach to adaptive continuous query processing.

A number of query processing protocols have been developed to support information collection in dynamic pervasive computing environments. Each of these protocols offers different guarantees about the quality of their execution.

Two examples are flooding queries and probabilistic queries as shown in Figure 3.1 in which the dashed lines denote sent the messages and solid lines represent available connections. The former broadcasts the query to all of its one-hop neighbors. Every recipient of the new query will in turn propagate the message to their neighbors until the network boundary is reached. By this way, the protocol likely reaches all nodes in the network and gains the most information. Figure 3.1(b) shows an example of flooding query. All nodes within a restricted radius (e.g., two hops) around the query issuer receive the query. However, the disadvantage of this kind of query is high cost in terms of message overhead and energy consumption, because the large number of nodes expected in network deployments may generate and deliver huge volumes of data.

In order to cope with this problem, probabilistic query protocols have been introduced [40], which have different features and capabilities to suit different application domains. Probabilistic query mode distributes the query propagation to a random subset of neighbors. Figure 3.1(c) illustrates a probabilistic query example where each node within two hops of the query issuer receiving a query retransmits it to the subset of neighbors (50%). When a persistent inquiry strategy employs the probabilistic query protocol, a selected subset of nodes are reached probabilistically, which reduces overhead because only some nodes are executed. However, it comes with added complexity, and yields less deterministic results than a flooding query protocol.

Clearly, different query protocols have different properties that impact their selection for use in approximating a continuous query; different query protocols provide different degrees of quality and have differing costs of query execution. If an application developer knows the nature of the environment in advance and does not expect the character of the environment to change, then the developer can use this information to select a single snapshot query protocol and frequency of issue to construct a continuous query to meet the application's quality and cost requirements. However, we do not make this assumption. Instead, we expect that the character of the environment can and will change rapidly and unexpectedly: nodes may suddenly join the network, impacting its density; data values may start rapidly changing in response to a physical phenomenon; autonomous mobile nodes may start moving in response to a perceived change in the environment. Therefore, our goal is to provide an adaptive approach to continuous query processing that changes the inquiry strategy to meet quality and cost requirements given the current operational environment.

3.2 A Review of a Framework for Adaptive Continuous Queries

In [40], a general framework has been presented to support for dynamic adaptive continuous query processing in mobile and pervasive networks. The authors first introduce the concept of *integration strategies*, which define how to combine the previous snapshot query results into a continuous query result. For example, an additive integration strategy integrates snapshot query results so that the continuous query result includes only information for nodes that have joined the network or presented a new value since the execution of the continuous query began. The framework also introduces the concept of *introspection strategies*, which are applied to the history of snapshot query results to assess the quality of the continuous query's execution. Finally, the framework introduces *adaptation strategies*, which describe when and how to adapt the inquiry strategy to meet the application requirements based on the results of introspection. These adaptation strategies are application-specific, and are defined by the application developer using knowledge about the relationships between the environment, the results of query execution, and the non-functional requirements of the application.

The work in this dissertation applies this framework to create adaptive continuous queries that use the history of snapshot query results to automatically learn the optimal inquiry strategy that balances quality and cost tradeoffs given the current environment conditions. Because the notions of introspection strategies and adaptation strategies are important to the work presented here, we provide a more in-depth review in the remainder of this section.

3.2.1 Integration Strategies

The framework presented in [40] includes a set of integration strategies which are used to combine a windowed history of snapshot query results. An integration strategy is simple a function, f, applied to a history of snapshot query results; the result of applying an inquiry strategy at stage i in the continuous query's execution is denoted as $\pi_i = f(\rho_0, \rho_i, \ldots, \rho_i)$. To illustrate the concept, we review several examples of integration strategies that have been proposed: cumulative integration, additive and departure integration.

Cumulative Integration. This is one of the simplest integration strategies, which combines all available results over time.

The statement of cumulative integration is presented as:

$$\pi_i = \pi_{i-1} \cup \sum_{j=0}^{n} \rho_j \tag{3.1}$$

One example of cumulative integration is illustrated in Figure 3.2. In the figure, the query issuer is in white color. During the sequence of queries, one node departed ($\rho_0 \mapsto \rho_1$) and one node was added ($\rho_1 \mapsto \rho_2$). The right side of the figure shows the integration results.

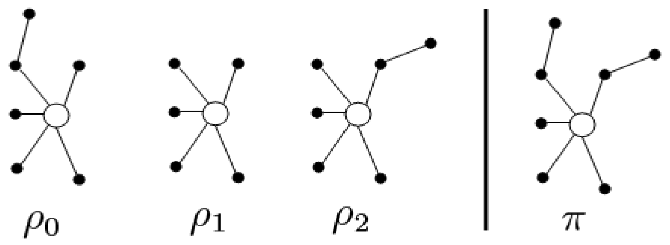


Figure 3.2: Cumulative Integration Example

Additive and Departure Integration. The cumulative integration captures a "growing" view of query results that existed over time. In contrast to cumulative integration, *additive and departure integration* determines the changes between the first query result and current query result. The additive integration shows the added nodes by comparing the current result and staring results, which can be defined as:

$$\pi_i = \rho_i - \rho_0 \tag{3.2}$$

The Figure 3.3 shows the example of additive integration. By comparing current query result (ρ_2) with the start of query (ρ_0), two additional nodes have been added.

Similar to additive integration, the departure integration presents the departed nodes between the start query instance and current query result. The function of departure integra-

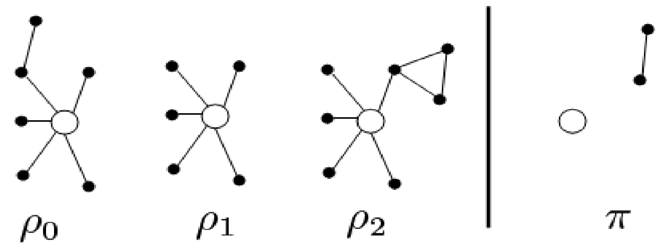


Figure 3.3: Additive Integration Example

tion is described by:

$$\pi_i = \rho_0 - \rho_i \tag{3.3}$$

For the example in Figure 3.4, two nodes have departed since the first query instance.

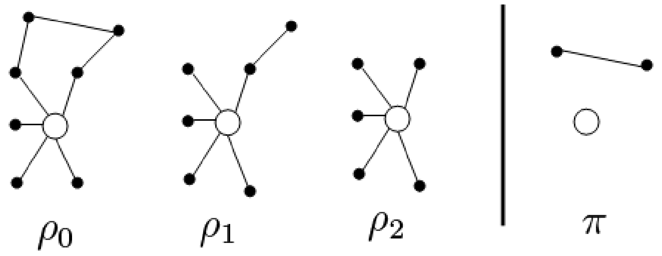


Figure 3.4: Departure Integration Example

Clearly, the ability to capture a quality representation of the state of the world with an approximated continuous query will impact the integration of results. This is a concern that we attempt to address in measuring the quality of the continuous query and adapting its execution. Next, we discuss introspection strategies, which are parts of the framework that allow for examination of the quality of the continuous query by examining a history of its component snapshot query results.

3.2.2 Introspection Strategies

An introspection strategy is applied to assess an application-specific definition of the quality of the continuous query's execution, which can later be used to dictate when to adjust the inquiry strategy. In the framework presented in [40], an introspection strategy is a function that is applied to the integrated result of a continuous query which describes the quality of the continuous query's execution. Using this information, one can determine suitability of current inquiry strategy by comparing historical query results to the desired results. Generally, the comparison can be defined as a distance function:

$$d = g(\gamma, \ell(i, j)) \tag{3.4}$$

where γ is the desired property and ℓ is the union function over a bounded window of query results from i and j. One sample expression of ℓ is:

$$\ell = \bigcup_{k=i}^{j} \rho_k \tag{3.5}$$

According to various applications' requirements and networking environment, the general introspection expression can be in different functions. For example, in order to evaluate the variability of environments, we can use set difference operator [40]. The metric of additive changes is:

$$d = \frac{|\ell_p|}{|\ell_p - \ell_0|} \tag{3.6}$$

This metric can be used to measure how much query results have been added during the query execution.

3.2.3 Adaptation Strategies

The introspection strategies can be used to assess whether the current inquiry strategy is suitable for the current environmental conditions in terms of satisfying the application's requirements. If not, the current inquiry strategy needs to be adjusted and the new inquiry strategy should be adopted. An adaptation strategy is specified by the application develop to dictate $when$ and how to adapt the inquiry strategy, typically using an introspection strategy to help guide this decision. The general formalization of an adaptation strategy is provided in [40] as follows:

$\langle\langle\tau, frequency\rangle, d, \delta^{+/-}, \langle\tau^*, frequency^*\rangle\rangle$

where $\langle\tau, frequency\rangle$ is the current inquiry strategy, d is the value of applying an introspection strategy, $\delta^{+/-}$ is the threshold which is used to be compared with d. The new inquiry strategy $\langle \tau^*, frequency^* \rangle$ will be activated if the threshold is reached. δ^+ means the new inquiry strategy will be triggered when the d is higher than δ. Similarly, the δ^- indicates that new inquiry strategy will be executed if the d is lower than δ.

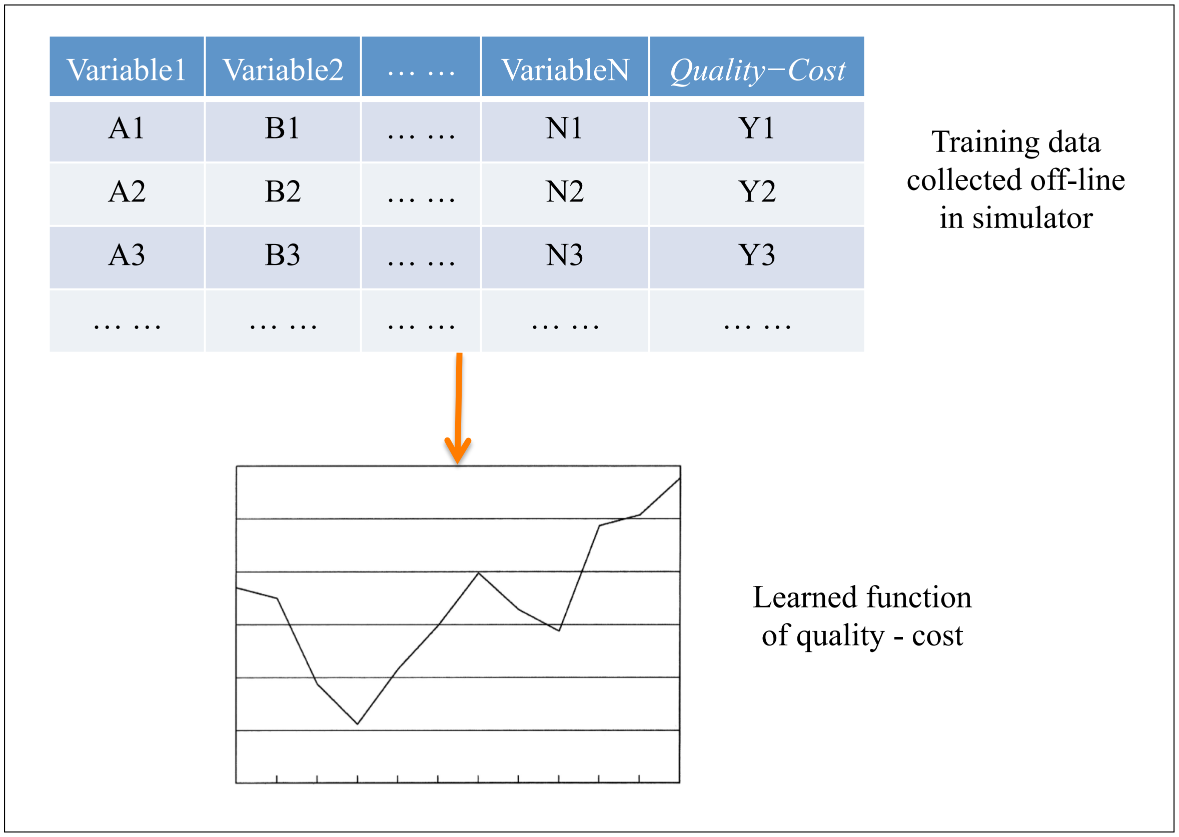
In the approach presented in [40], the application developer is responsible for creating application-specific introspection and adaptation strategies. While this is a flexible approach that will serve a wide variety of applications, it requires the application developer to have a deep understanding of the relationship between the conditions of the operating environment, the parameters of the inquiry strategy, and the achieved quality of query execution. The goal of this dissertation is to reduce this burden on the programmer. Instead, we seek to develop an approach which learns the most appropriate way to adapt the query to optimize for the quality and cost tradeoff, and uses that information to automatically adapt the execution of the query.

3.3 A Learning-based Approach to Adaptive Continuous Query Processing

In this dissertation, we seek to develop an approach which learns the most appropriate way to adapt the continuous query to optimize for the quality and cost tradeoff, and to use that information to automatically adapt the execution of the query.

In our new framework, we treat the adaptation of query as the process of finding the inquiry strategy that optimizes the tradeoff between quality and cost. In other words, we treat this as a numerical optimization problem in which we are trying to maximize the value of $F - \alpha C$, where F is a function that defines the quality of a continuous query's execution, α is an associated weight that indicates the importance of cost, and C is a function that defines its cost. One way to define the quality of the query's execution is to compare the results of a query to the state of the world at the instant that the query is issued. Network and environmental dynamics that occur during query processing will likely render it impossible for a distributed query protocol to return a perfect reflection of the state of the world, and the selection of the inquiry strategy will also impact how well the query result reflects the ground truth. Ultimately, an oracle view could be used to define the value of a function F

that describes the quality of a continuous query's execution, but this oracle view could not be computed online and therefore cannot be used in practice to help the continuous query adapt its execution. Therefore, we must learn an approximation of the quality function; we call this approximation $\hat{F}$.



Variable1	Variable2	… …	VariableN	*Quality−Cost*
A1	B1	… …	N1	Y1
A2	B2	… …	N2	Y2
A3	B3	… …	N3	Y3
… …	… …	… …	… …	… …

Training data collected off-line in simulator
Learned function of quality - cost

Figure 3.5: Using A Simulator to Collect Training Data for $\hat{F} - \alpha C$

We apply an instance-based learning approach to learn the function $\hat{F} - \alpha C$. Together, Figure 3.5 and Figure 3.6 illustrate this approach. As shown in Figure 3.5, we use the oracle view of the network simulator to collect training data. Each instance in the training data set represents the result of a snapshot query's execution as part of a continuous query in a given environmental scenario. Each instance includes information about the environment that can be detected by each node (such as node speed, battery life remaining, number of neighbors, etc.), the inquiry strategy used to issue the query, and the actual query results. We can use the simulator's oracle view to compute the quality of the query. We apply machine

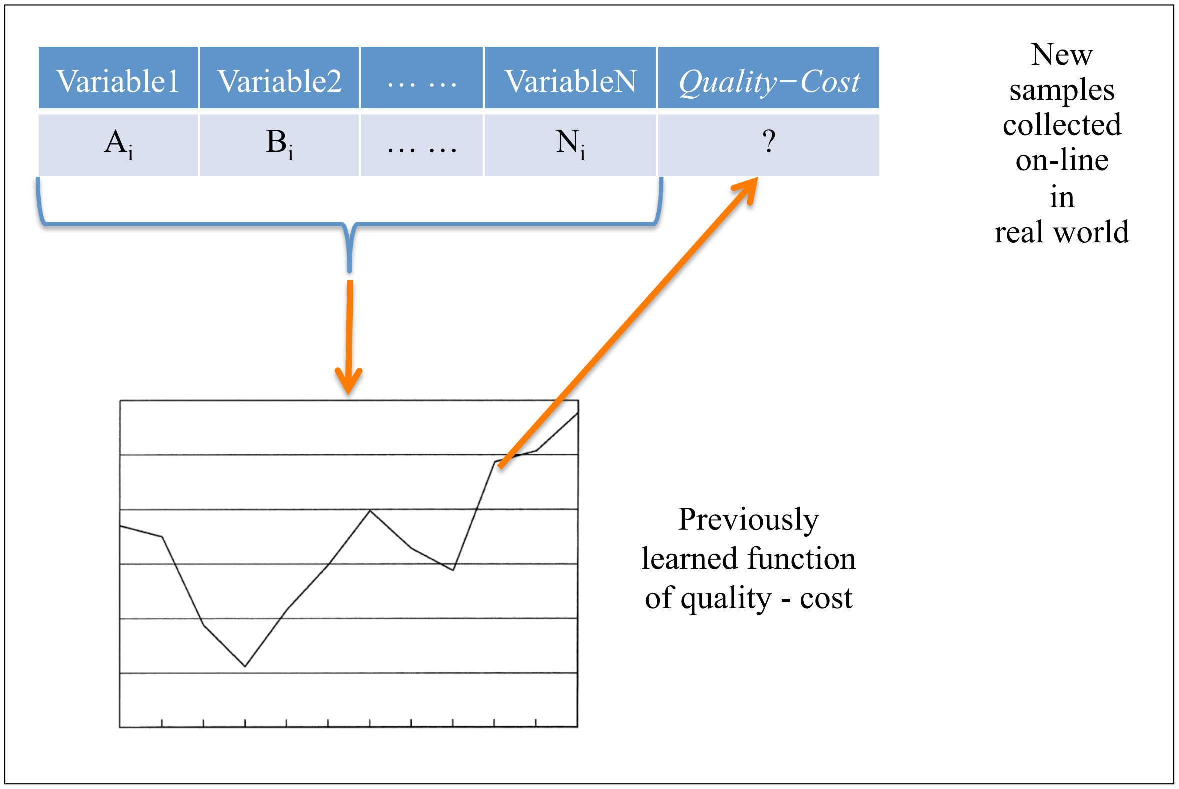

Variable1
Variable2
… …
VariableN
Quality−Cost
A_i
B_i
… …
N_i
?
New samples collected on-line in real world
Previously learned function of quality - cost

Figure 3.6: Using the Learned Function to Apply $\hat{F} - \alpha C$ to New Online Sample

learning algorithms to learn an approximated function of quality from this training data. This learning step takes place off-line, on a resource-rich computing device that runs the network simulator. Figure 3.6 illustrates the use of this function by a node operating within a MANET to the results of a live continuous query that is issued over the network. This results in values of the $\hat{F} - \alpha C$ function for some set of inquiry strategies; the continuous query is adapted by choosing the inquiry strategy with the highest $\hat{F} - \alpha C$ value.

In recent years, machine learning techniques have been used for a wide variety of problems in the domains of wireless sensor networks (WSNs) and mobile ad-hoc networks (MANETs), such as network routing [86], failure detection [17, 23], localization [63], vehicular ad-hoc networks (VANETs) [89], and so on. Some examples of widely used machine learning algorithms are reinforcement learning [8,42,87,94], swarm intelligence [15, 16,26,43,44,82], Bayesian networks [61], radial basis functions [60,69], heuristics [46,47], and so on. In [28], the survey of machine learning algorithms for data routing problem in

WSNs and MANETs is provided and the guide of applying those algorithms to applications in ad-hoc networks is presented. One commonly used algorithm is Q-learning [4, 95, 96] in which the agents learn how to act optimally in Markov Decision Process (MDP) [6, 73] problems without a model of the environment. For example, the authors in [11] describe Q-Routing algorithm for the packet routing. The algorithm is based on the reinforcement learning technique of Q-learning to discover efficient routing paths without the knowledge of network topology in advance. The approach in [29] employs Q-Learning approach on clustering and data aggregation to address the problem of energy expenditure and maximize network lifetime in WSN. Another popular machine learning algorithm is Radial Basis Function (RBF) [35, 60, 69] network which is an artificial neural network. Radial basis function has met with success in a large diversity of applications including speaker normalization and identification [31, 80], data mining [8, 13, 70], recognition of radar targets [32] and so on.

In the following two chapters, this dissertation presents two different approaches to define quality, and evaluates the use of those definitions of quality in our learning-based framework to adapt a continuous query. We then present a middleware solution that allows application developers to use these approaches to learning-based adaptive continuous query processing and provides the ability to create new definitions of quality and cost, or to apply different machine learning techniques for use in their applications.

CHAPTER 4: FIDELITY-BASED ADAPTATION

As we begin this exploration, we make a simple but important observation: the quality of the continuous query is impacted by the quality of its component snapshot queries. In this chapter, we explore an approach that uses the *fidelity* of snapshot queries to define the quality of a continuous query's execution. Snapshot query fidelity, introduced in [66,68], is a measure of how well the results of a snapshot query matches the state of the world at the moment that the snapshot query is issued. Since the snapshot query executed in distributed fashion over dynamic network, then changes occur during execution that may prevent query results from the matching ground truth. Therefore, query fidelity is an important measure for applications that use queries to collect information that will be used to make decisions; information about the fidelity of the query's execution can be used to enhance decision process and can aid in the development of more reliable applications. In this chapter, we propose a learning-based adaptation strategy that uses an assessment of snapshot query fidelity for component snapshot queries to learn the optimal way to adapt a continuous query's processing to balance quality and cost tradeoffs in the current conditions of the dynamic operational environment.

4.1 Modeling the Execution Environment

In previous work, Payton et al. [66, 68] proposed a model to express the dynamic execution environment in order to make it possible to precisely capture the semantics of query execution. We adapt their model in this work, and review it below.

A node in the network is represented by a tuple (ι, λ, ν), where ι is a unique node identifier, λ is the the node's location, and ν is the node's data value. The global state of a network is considered as a set of node tuples, which is represented as C. Connectivity can be described by applying a symmetric, reflexive, and transitive relation on the configuration. It is possible to apply a projection to a configuration to find the set of hosts that are reachable from a particular reference host; this projection is called an effective configuration. The network is not static, so it is not sufficient to talk about representing a network as a single configuration; instead, the evolution of the network is represented as a sequence of configurations. A configuration change occurs when a node's data value is changed (e.g., a variable assignment occurs) or when the topology of the network changes (e.g., when any node in the network gains or loses a one-hop neighbor).

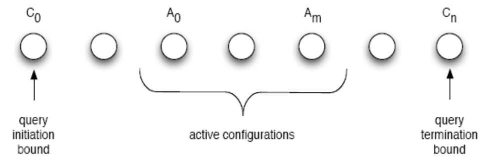


Figure 4.1: Query bounds and active configurations [66]

Figure 4.1 illustrates the use of this model of the environment to describe the execution of a snapshot query using a distributed query processing protocol. The execution of a single snapshot query takes place over a sequence of configurations. C_0 is the starting configuration when the query is issued, and C_n is the ending configuration when the query result is delivered. All the configurations between C_0 and C_n are the active configurations which exist at some point during query execution. A snapshot query's result, ρ, is the set of host tuples that were collected during execution. Every element of a snapshot query's result must have existed in one of the configurations C_0 to C_n, and every element must have been reachable from the query issuer. Since the snapshot query's result is a set of host tuples, this is essentially the same as a "virtual" configuration. Therefore, it is relatively straightforward to talk about how well the query reflects the state of the world; one simply needs to compare the query result (a configuration) to the sequence of configurations that existed during query execution.

Extending this model, a continuous query is modeled as a sequence of snapshot queries which are expressed as $\rho_0 ... \rho_i$ where ρ_0 is the result of snapshot query with the sequence number of zero. We will use this extended representation in the remainder of this dissertation. In the next section, we review previous work that uses the model above to define the fidelity of snapshot queries. We will then show how we can use these definitions of snapshot fidelity to define the fidelity of a continuous query.

4.2 Fidelity of Queries

In order to describe the relationship between the state of the network and a snapshot query's result, definitions of snapshot fidelity have been introduced in [66, 68]. This set of fidelity definitions ranges from a very accurate reflection of the state of the network to a definition that provides a weaker guarantee about the accuracy of query results. Below, we review the definitions of snapshot query fidelity.

With ATOMIC fidelity, all of the results returned should come from the same configuration:

$$\text{ATOMIC} \equiv \exists i : 0 \leq i \leq m \wedge \rho = C_i \tag{4.1}$$

Here, this fidelity definition captures the fact that the relationships among the items returned by the query are important; all of the responses should have existed at the same point in time (i.e., within a single configuration) to give an accurate picture of the network state a a single point in time. A query protocol that provides this kind of guarantee is valuable to applications that require a precise view of entities within the environment to make decisions. ATOMIC fidelity says that the results returned by a query are *comparable*; their values co-existed in time (within a single configuration), and so can be integrated to get a correct view of the world. For example, a military commander may need to know the relative locations of all of his troops and assets before making a strategic decision.

Query protocols that are guaranteed to achieve ATOMIC fidelity sometimes fail or are extremely costly to execute in highly dynamic environments. For applications that do not require such strong guarantees about their results, a more relaxed definition of fidelity may be suitable. The definition of ATOMIC subset fidelity provides a slightly more relaxed definition. This definition of fidelity states that all of a query's results are part of one single configuration (i.e., they are comparable), but not necessarily all results that are present within that configuration are captured by the query. The formalization of the ATOMIC subset is:

$$\text{ATOMIC SUBSET} \equiv \exists i : 0 \leq i \leq m \wedge \rho \subset C_i \tag{4.2}$$

The authors also provide an extended definition of ATOMIC SUBSET called QUALIFIED SUBSET, which provides information about the quantity of results that existed in a configuration that were not captured by the snapshot query. Because all of the results existed in the same configuration with this definition, they are still comparable and can be integrated (e.g., as a sum) correctly.

The weakest definition of fidelity states simply that the results returned by a snapshot query existed during some configuration. This definition of WEAK fidelity is captured as:

$$\text{WEAK} \equiv \rho \cup_{i=0}^{m} C_i$$

Variants of this definition can provide a qualification of how many results that ever existed during the snapshot query's execution were captured with the snapshot query's results. Snapshot query results that were collected by a query that achieved WEAK fidelity are *non-comparable*; their results did not exist at the same point in time, and their integration would result in an incorrect answer. Consider the example of a military commander that is attempting to count the amount of ammunition available on trucks at a site, and during the execution of his query, there was a transfer of ammunition between trucks. If the query gets the value of Truck A before the transfer and the value of Truck B after the transfer, there is strong potential for an incorrect count of the amount of ammunition.

In order to determine the fidelity degree of a continuous query, it seems reasonable to assume that the fidelities of its component snapshot queries can be used in some way. In fact, this is an intuitive practice to explore given our model of a continuous query. As pointed out in the work of Julien et al. [41], since the component snapshot queries are combined into a continuous result by applying an *integration strategy*, it follows that the application of an integration strategy would be impacted by a single snapshot query with *non-comparable* fidelity (i.e., any fidelity that is not ATOMIC or ATOMIC subset, in which snapshot query results are guaranteed to have co-existed at a point in time). During integration, the continuous query will be considered to have a reduction in quality if there is any WEAK snapshot in the window. This is a conservative definition, because one WEAK snapshot within the window will result in the continuous query being incomparable after aggregation. In [41], $d_{fidelity}(q)$ is defined for the fidelity of the continuous query (q) based on the fidelity of the past w snapshots:

$$d_{fidelity}(q) = \frac{1}{w} \sum_{i=k-\eta}^{k} S_i \tag{4.3}$$

where k is the current snapshot and S_i is the fidelity of q's i^{th} snapshot. S_i is equal to one if the snapshot is comparable; otherwise, S_i is zero.

4.3 Fidelity-based Adaptation with a Learning-based Approach

In a mobile ad-hoc network, nodes may move at any time and the data that is sampled is often rapidly changing. The fidelity of the query results can be significantly impacted when these kinds of changes occur in the middle of a query's execution. The mobility of nodes and evolving network topologies can a query to miss information or to return inconsistent (i.e., non-comparable) results. In addition, different inquiry strategies result in different fidelity. For example, inquiry strategies with higher frequency have higher fidelity value of query results than those inquiry strategies using lower query frequency. However, the inquiry strategy with higher frequency has associated larger overhead costs.

Compounding this issue is the fact that during query execution, the environmental conditions may change over time. These changes can impact the query fidelity achieved. Figure 4.2 demonstrates the effect of changing speed on the achievable query fidelity in a dynamic ad-hoc network as the speed of nodes varied from 5 m/s to 25 m/s when the query frequency is equal to 0.5s. It is notable that the fidelity of continuous query is decreased with the increase of network mobility. This is the result of incomplete messages and dynamic topologies because of high mobility.

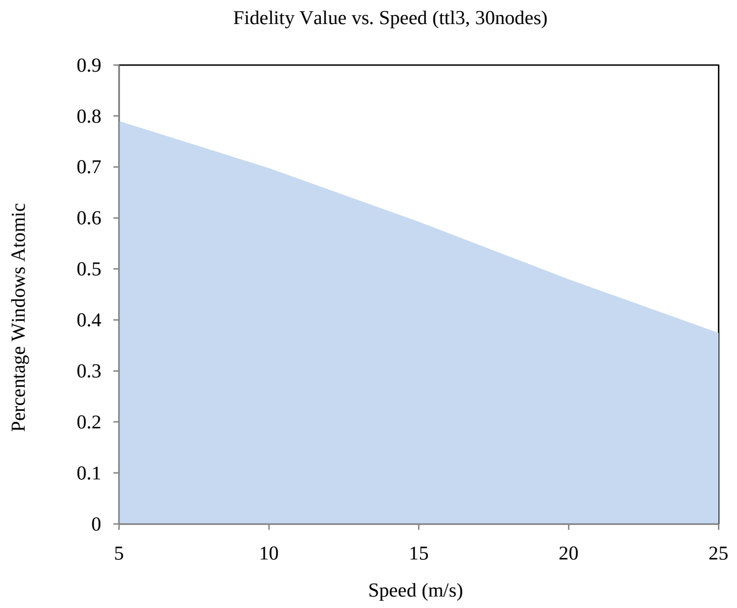


Figure 4.2: Fidelity-based introspection vs. speed for a network of 30 nodes

Since the operational environment is dynamic, one important and challenging issue in continuous queries is to provide adaptive algorithms in response to the evolving conditions. In [41, 76], researchers defined fidelity-based adaptation rules based on the application's requirements. Fidelity-based introspection is employed to evaluate the continuous query's quality. When the query results do not meet the application requirements, the related adaptation rules will be triggered. Therefore, the query strategy will be changed. A generic representation of a fidelity-based adaptation rule is:

$<< \tau, frequency >, d_{fidelity}, \delta^{+/-}, < \tau, frequency^{*} >>$

where $< \tau, frequency >$ is the continuous query strategy, $d_{fidelity}$ is the value of the continuous query's quality, and $\delta^{+/-}$ is the threshold which is used to evaluate $d_{fidelity}$ and trigger the new query strategy $< \tau, frequency^{*} >$. δ^{+} means the adaptation will be triggered when the $d_{fidelity}$ is higher than δ. In the same way, the δ^{-} indicates that new query strategy will be executed if the $d_{fidelity}$ is lower than δ.

Similar adaptation approaches and policies can be defined according to different application requirements. The main limitation of this approach is the fact that it requires the application developer to have significant knowledge about the interplay between the inquiry strategy and conditions of the operational environment and how both of those concepts map to definitions of quality and costs. Our goal is to provide applications with the benefit of adaptive continuous queries that are sensitive to cost and quality tradeoffs without requiring an application developer to have knowledge of the inner workings of continuous query execution.

In this section, we show how we employ a learning-based approach to capture the relationship between dynamic environments and the properties of inquiry strategy (e.g., fidelity and cost). We can then use this information to adapt the inquiry strategy to achieve the optimal tradeoff between quality and cost given the current environmental conditions.

A. Learning Model and Parameters

In the remainder of this chapter, we apply multiple linear regression to adapt continuous queries to match changing conditions in dynamic environments. We use the multiple linear regression model to capture the relationships between the fidelity of continuous query and dynamic conditions. The goal of multiple regression analysis is to learn the association between a dependent variable and several independent variables. Then, the model can be used to estimate the value of the dependent variable given value of independent variables. More specifically, a multiple linear regression model [20] describes the relationship between response variable Y and a set of n explanatory variables, $X_1, X_2, ..., X_n$. Regression analysis can be used to learn changes of the response variable when one or more explanatory variables changes. In the real world, linear regression analysis is widely used in economics [50] and finance areas in order to find trends.

In this chapter, we use regression model to capture the relationship between properties of continuous query and environmental conditions. In our application of multiple linear regression, the regression model can be employed for automatic adaptations when environmental conditions are changed. For example, when the speed of nodes has been increased, the query interval can be shortened in order to maintain the previous achievable query fidelity. Using this information, the continuous query's inquiry strategy will be selected to optimize the quality and cost tradeoffs in light of the changing conditions.

A general multiple linear model of the form relating Y and a set of n explanatory variables (X_i) is given by:

$$Y = \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \varepsilon \tag{4.4}$$

where $\beta_1, \beta_2, \cdots, \beta_n$ are constants referred to as the model partial regression coefficients, and ε is a random disturbance. In this application, there are six explanatory variables ("regressors"): the number of nodes, query frequency, maximum speed of nodes, average speed of nodes, speed variances of nodes, and query window size (η). Y is represented as the fidelity of the continuous query ($\hat{d}_{fidelity}$):

$$\begin{aligned}\hat{d}_{fidelity} &= \beta_1 * log_e(NumberofNodes) + \beta_2 * log_e(queryFrequency) \\ &+\beta_3 * log_e(MaxSpeedofNodes) + \beta_4 * log_e(AvgSpeedofNodes) \\ &+\beta_5 * log_e(SpeedVarianceofNodes) + \beta_6 * log_e(windowSize) + \varepsilon\end{aligned} \tag{4.5}$$

All the above variables can impact the fidelity of the continuous query. For example, with the increasing of number of nodes, the network becomes more dense and collisions of messages becomes more likely; this may lower the fidelity because the messages are not received. For query frequency, a higher query frequency can achieve higher fidelity. This is because it is more likely that if one snapshot query achieves comparable fidelity, it is likely that a query that is successively issued close in time to the first is likely to achieve comparable fidelity as well; in other words, a high query frequency increases the likelihood that a query issuer that encounters environmental conditions that are favorable to achieving comparable fidelity for one query is more likely to do so for quickly issued successive queries. Fidelity measures how well the query results match the state of dynamic networks.

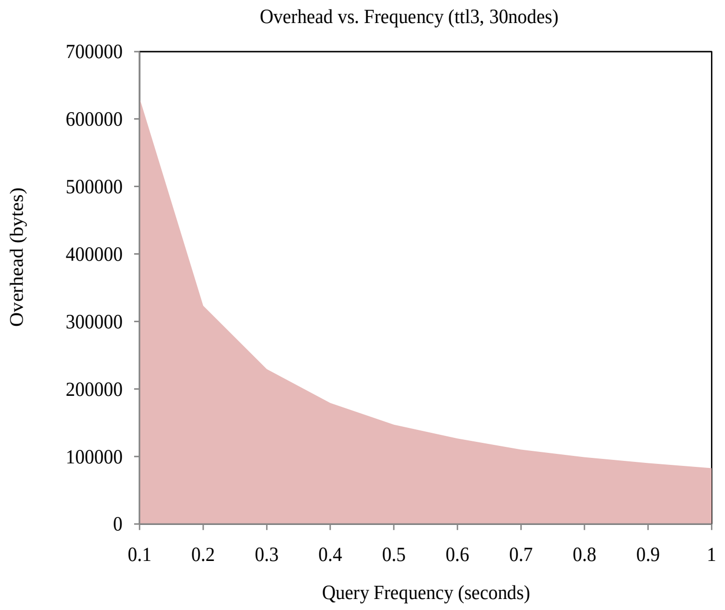


Figure 4.3: Overhead vs. frequency for a network of 30 nodes

Speed information (e.g., max speed, average speed, and speed variances) of node is an one aspect of the dynamic environment. As expected, it is more difficult for continuous queries to provide atomic measurement of the environment while the environment becomes more dynamic. Therefore, the increasing of speed will result in lower fidelity of continuous queries.

B. Defining Overhead of Execution

The cost of execution is measure by the message overhead (in bytes) for messages exchanged as a result of a query's execution. This cost is relevant to networks of mobile devices because communication of message incurs significant drain on the battery. Figure 4.3 explores how query frequency is related to execution overhead. As expected, a higher query frequency has higher overhead.

Using the model and input parameters described in the previous section, regression analysis is employed to learn the relationship between query frequency and overhead. In this application, the criterion variable is query overhead. In addition, the set of explanatory variables (X_i) contains two elements: the "number of nodes" and "query frequency". The formula can be expressed as:

$$f_{overhead} = \beta_1 * (NumberofNodes)^a + \beta_2 * (queryFrequency)^b + \varepsilon \qquad (4.6)$$

where the definition of β_1, β_2, and ε are similar to the Equation 4.5. a and b are the power for explanatory variables.

C. An Overview of the Adaptive Strategy

Continuous queries using high frequency are more likely to have a high percentage of successive snapshots that achieve comparable fidelity. However, a high frequency of issue requires exchanging more messages. Our adaptive approach selects the inquiry strategy that finds the optimal tradeoff between quality and cost.

With respect to the fidelity and overhead in the continuous query, we introduce an adaptive algorithm which assesses the quality of snapshot query executions in the current environment and determines when and how the snapshot queries should be issued to achieve the optimal tradeoff between quality and cost of the continuous query's execution. In general, the optimization function can be expressed as:

$$g(d, c) = d_{fidelity} - \alpha * C \qquad (4.7)$$

where d is the fidelity value, C is the overhead of execution, and α is the parameter that indicates the weight of cost. This means that the suitability of a continuous query is an expression of required fidelity semantic against the weighed associated execution cost. In this application, C can be calculated by Equation 4.6. Hence, the particular formalization of tradeoff is: $d_{fidelity} - \alpha * f_{overhead}$.

4.4 Evaluation

With the aim of collecting data for assessing the regression model and evaluating the adaptation ability, the continuous queries on varying numbers of nodes at varying speeds are executed in a simulated mobile ad-hoc networking environments. We use an implementation of a two-phase self-assessing snapshot query protocol [68] to issue snapshot queries and acquire fidelity labels for them. We use the OMNeT++ network simulator [92] and its mobility framework extension [52] to issue a continuous query that is constructed as a sequence of snapshot queries and to collect training data for our learning-based adaptive approach to continuous query execution. Using the simulator, we execute a continuous query in a MANET environment where a set of nodes moves in a rectangular area with the size of $1200 \times 1000\ m^2$. At the beginning, a number of nodes are randomly deployed in the space to form an ad-hoc network. In our simulations, the nodes move according to the "random waypoint" model [12], in which each node chooses a destination in the space randomly and moves to the selected destination at a given speed between 0 m/s and 30 m/s. When a node reaches the destination, it pauses for a pause time, chooses another random destination, and repeats the process. We run our simulations with a pause time of 0s to represent a more dynamic network.

We evaluate query protocols and the adaptive strategy under varying environmental conditions. Specifically, we varied three simulation parameter settings: number of nodes, degree of mobility, and query frequency. First, the number of nodes is varied from 25 to 85 at increments of 5. As more nodes are placed in the space, the network density increases, since the area size is constant. The density of the network is expected to impact fidelity, since network congestion will result and cause messages to be lost. Second, the average speed of nodes varies from 0 m/s (completely static) to 30 m/s (high degree of mobility) in multiples of 5. The final variable is query frequency. We control the query frequency by adjusting the interval time between two successive query processes. When the interval is short, the query frequency becomes high. In the simulations, there are ten different query intervals from 0.1s to 1.0s at increments of 0.1.

In this section, over 3000 records collected from simulated continuous queries are used to train the regression model. Then, the trained model is applied to several testing sets for different goals of evaluations. We demonstrate that regression models can estimate fidelity value and overhead cost in different situations. Furthermore, the performance of adaptive strategies is explored by comparing with non-adaptive strategies. Finally, our proposed automatic adaptive strategy is compared with application-specific strategy.

A. Learned Estimate of the Achieved Fidelity

Figure 4.2 illustrates different fidelity value for the continuous query in dynamic environment (e.g., varied speed). According to six explanatory variables in the Equation 4.5, the environmental settings include six explanatory variable parameters: number of nodes, query frequency, specified maximum speed of nodes, average speed of nodes, speed variances of nodes, and query window size.

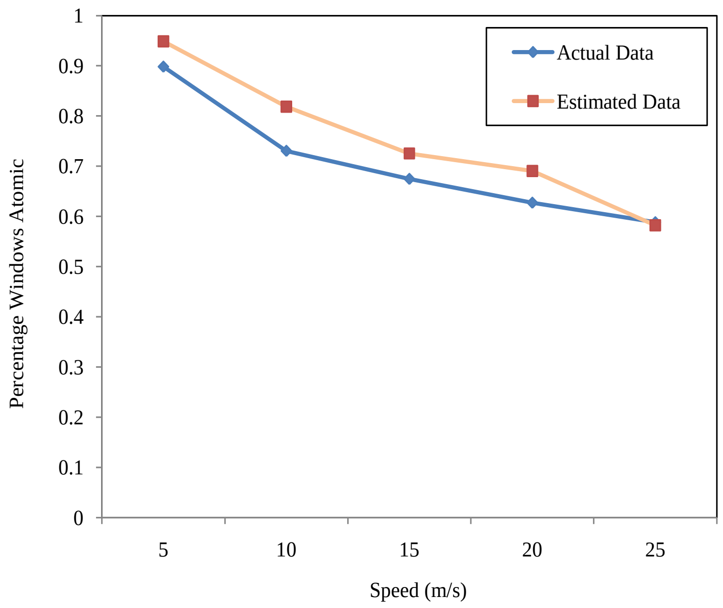


Figure 4.4: Performance of regression model about various speed (number of nodes = 30, query frequency = 0.5s, query ttl = 3 hops)

In order to demonstrate the ability of the linear regression model, Figure 4.4 shows the comparison between actual $d_{fidelity}$ and estimated value calculated by the regression model with respect to one of the dynamic aspects in the networking; in this case, the speed of the nodes. It demonstrates that the estimated value is very close to the actual value, especially when the speed is not lower. Furthermore, it is obvious that higher mobility results in lower percentage of windows atomic. This means that it is more difficult for continuous queries to achieve the atomic measure when the environment becomes more dynamical because of missing data and incomplete information.

B. Learned Estimate of the Cost of Execution

In general, the query strategies have two important aspects: quality of query results and cost of execution. In this application, *Percentage Windows Atomic* presents the quality of the query strategy, which is analyzed in previous section. In this part, we evaluate the performance of regression model which provides the overhead estimation.

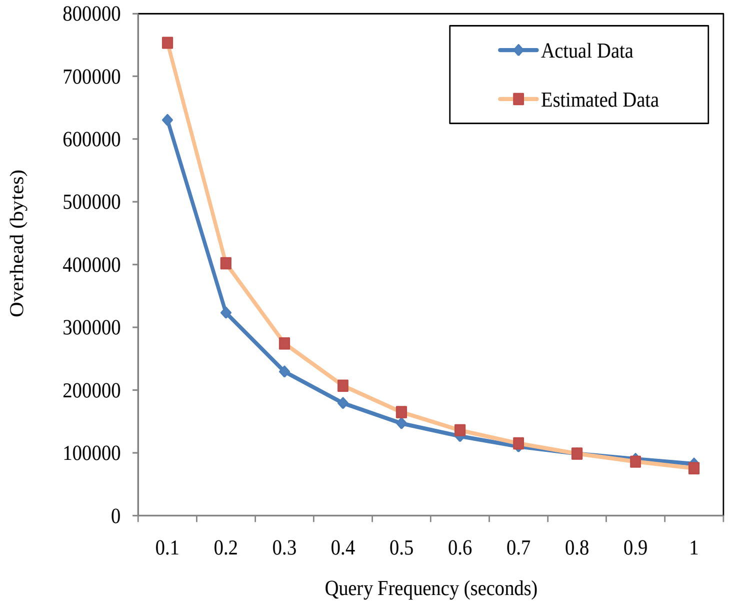


Figure 4.5: Performance of regression model about various frequency (number of nodes = 30, max speed of nodes is 20 m/s, query ttl = 3 hops)

From Figure 4.5, it is notable that the estimated overhead is very close to the actual overhead cost from simulation results. It demonstrates that the proposed regression mode can capture the relationship between variables of networking environments and approximate the overhead cost of query strategies.

C. Evaluation of Fidelity-Based Adaptation with a Learning-based Approach

Because the goal of our adaptation algorithm is to adjust the previous query strategy and determine the suitable query strategy under changing environmental conditions, the most important aspect of our evaluation is to assess the ability of our proposed algorithm to adapt to consider the tradeoff between quality and cost. We first compare our approach to a non-adaptive strategy. We then compare our approach to an adaptive approach that uses an application-specified adaptation strategy to control adaptation.

First, we compare our adaptive approach to a non-adaptive continuous query. This non-adaptive continuous query is issued as a sequence of snapshot queries at an given (and unchanging) query frequency with a given (and unchanging) query protocol. Consider a situation in which a user wants to maintain the fidelity value at a stable level in order to provide comparable query results. If the networking conditions are static, the goal can be achieved by this simple approach (e.g., keeping the same query frequency and query protocol). However, in dynamic operational environment, evolving conditions make it not possible to achieve the same atomicity requirement.

Figure 4.6 shows how a linear regression model can be used to adapt a continuous query when the environmental condition (e.g., speed) is changed. Results are shown for a network of 30 nodes. The speed of the nodes is changed over time from 5m/s to 30 m/s. The window size of the continue query in this application is 2. In order to provide high quality results, an application requires the fidelity value ($d_{fidelity}$) be maintained at the level of 80%. The non-adaptive approach selects a particular query protocol (here, a simple flooding protocol) and query frequency (here, a query is issued every 500 ms). For the non-adaptive approach, the value of fidelity goes down with the increase in speed because it is more difficult for successively issued snapshot queries to achieve comparable fidelity in highly dynamic environments. When the regression model is employed, the model will be triggered when the speed is changed and it will adapt the query frequency to maintain good query results. Figure 4.7 shows the corresponding frequency when the mobility of nodes is updated.

Continuous query strategies that obtain strong fidelity semantic usually have associated high cost. In some cases, applications may be able to tolerate a weak fidelity semantic in order to reduce the execution cost of continuous queries. This requirement motivates another kind of scenarios. In this situation, with the consideration of quality and cost, the

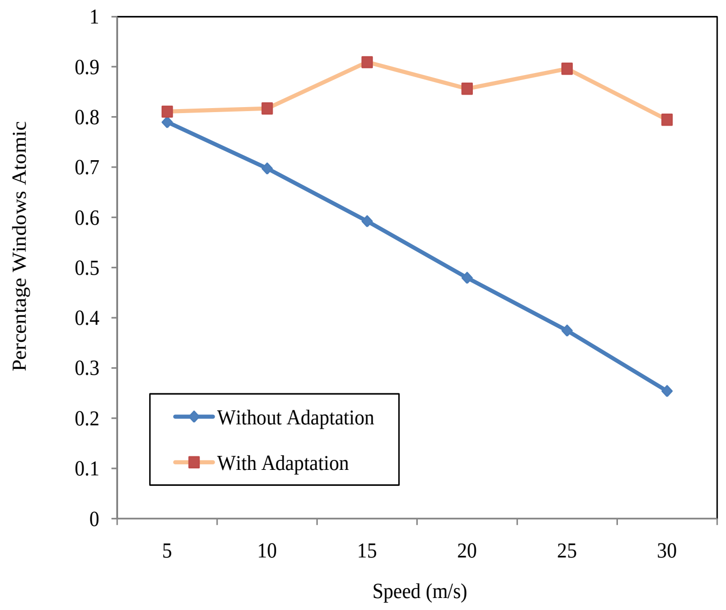


Figure 4.6: Comparison of $d_{fidelity}$ for Adaptive and Non-adaptive Continuous Queries

goal of those applications is to reach the optimal tradeoff between percentage windows atomic and execution overhead.

Figure 4.8 illustrates the performance of using our learning-based adaptive strategy for continuous queries. We varied the scaling to increase the influence of the cost of execution from 0.5 to 2 in steps of 0.5 during the time between 50s to 200s of the simulation. The plot compares the tradeoff of adaptive strategy and non-adaptive strategy. The solid line shows the results of employing our adaptive strategy to determine the continuous queries. It is notable from Figure 4.8 that the adaptive strategy results in substantially improved ability to consider the tradeoff between quality and cost (here, measured as $d_{fidelity} - \alpha * C$), especially when the weight is increased.

Figure 4.9 highlights the comparison of frequency between the adaptive strategy and non-adaptive strategy. With the increasing weight of cost, the adaptive strategy will lower the query frequency in order to reduce query processing overhead. However, the strategy without adaptation will maintain the same query frequency in regardless of cost.

Figure 4.10 and Figure 4.11 show the impact of query frequency on the quality and cost

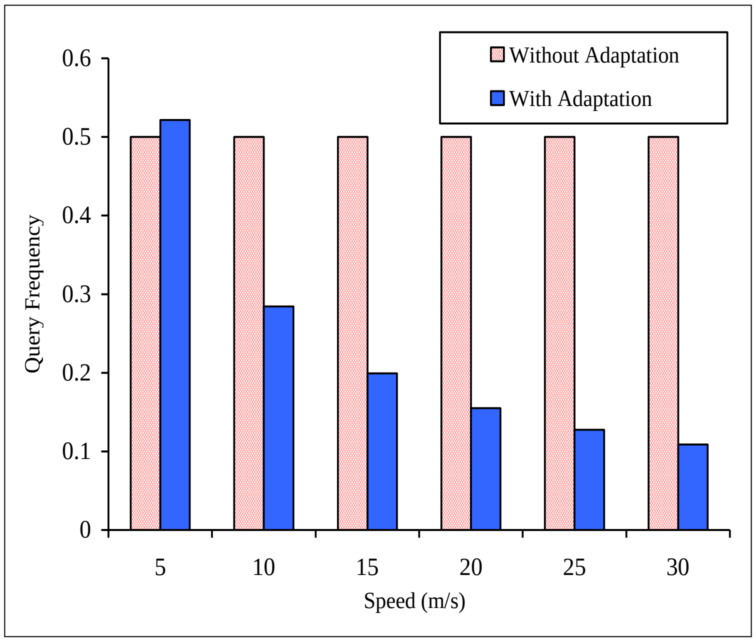


Figure 4.7: Comparison of Query Frequency for Adaptive and Non-adaptive Continuous Queries

of continuous queries. From Figure 4.10, there are two notable issues. Firstly, it is possible that adaptive strategy can figure out the suitable continuous query strategy which has higher fidelity quality than non-adaptive strategy when the weight of cost is low. Secondly, it is obvious that the fidelity semantic of adaptive strategy decreases with the increasing of scaling factor. This is a result of adapting query frequency to the dynamically operational environments in order to enable the continuous query to maintain the optimal value of tradeoff.

Similarly, Figure 4.11 shows the associated execution cost of two different query strategies. It demonstrates that the adaptive strategy works well for reducing the query cost when the weight of query processing overhead becomes higher.

To this point, we have shown the performance of our proposed adaptation strategy with comparison of non-adaptive strategy. We now compare our learning-based adaptation approach strategy to an application-specified adaptation strategy, which requires the application developer to dictate when and how the inquiry strategy should be adapted. We use the application example in [41], which describes an adaptive strategy for an automobile safety

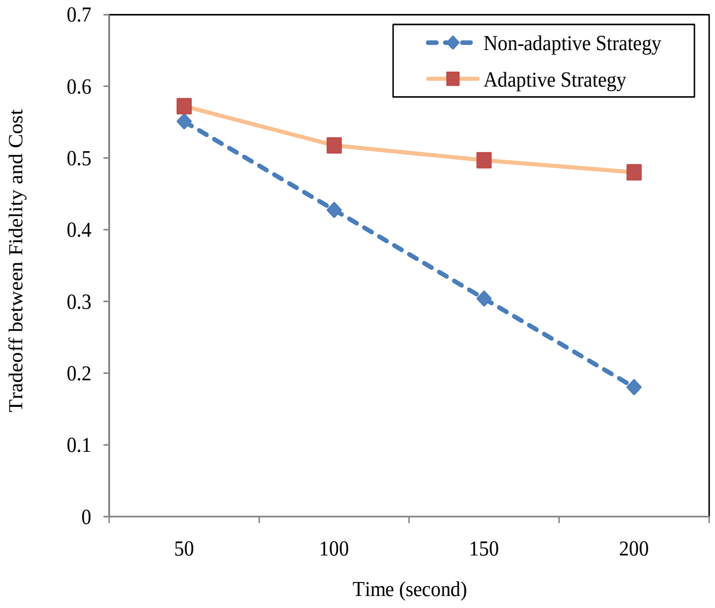


Figure 4.8: Comparison of quality and cost tradeoff for adaptive and non-adaptive approach

monitoring application. In this scenario, a car can issue a continuous query to monitor the environmental situation (e.g., positions of other cars, presence of obstacles) in order to maintain safe driving conditions. Let us assume the speed vehicle is zero when it just starts in the parking lot firstly. After forty seconds, the speed increases to 5m/s. For each interval of forty seconds, the speed will increased by 5m/s. Finally, the speed is 20m/s at the time of one hundred and sixty seconds. The formalization of application-specified adaptation strategy is provided in [41] as:

$\langle\langle\tau, freq > 100ms\rangle, d_{fidelity}, 0.8^{-}, \langle\tau, freq - 100ms\rangle\rangle$

The rule defines that the sampling rate of continuous queries will be increased in order to maintain the required fidelity level ($\delta = 0.8$) when current fidelity value is lower than 0.8.

The results of this rule is shown in Figure 4.12. We can see that the frequency of continuous queries will be changed based on the designed policy to satisfy the application requirement. However, application knowledge is necessary for developers to generate this specific adaptation rule. In Figure 4.13, we illustrate the performance of proposed automatic adaptation strategy, which adjusts the sampling rate of continuous queries without the human-defined adaptation strategy when the environment condition (e.g., speed) is

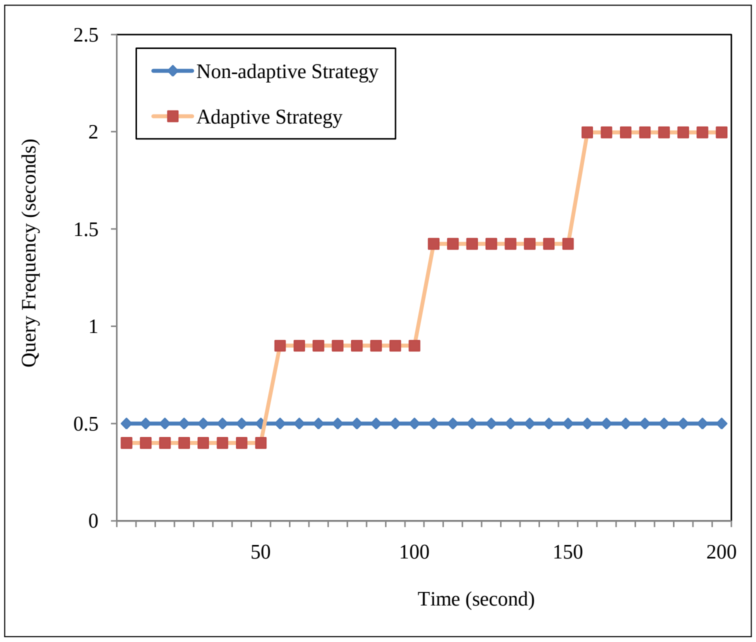


Figure 4.9: Comparison of frequency for learning-based adaptation and non-adaptive strategy

changed.

Figure 4.14 and Figure 4.15 present the performance (i.e., fidelity value and execution cost) comparison between automatic adaptive strategy and application-specific strategy. It is observed that simulation results of those two strategies are very close to each other which demonstrates that the automatic adaptive strategy is able to perform satisfied adaptation tasks without the preliminary knowledge. There are two notable things from Figure 4.15. First, from 40sec to 80 sec, the overhead of using automatic adaptive strategy is the steady while the overhead is increasing for application-specific strategy. This is because that automatic adaptive strategy can determine the frequency of continuous query quickly when the environmental conditions change. However, the application-specific adaptation strategy needs to periodically test the suitability of continuous query by adjusting the frequency. Second, in some cases (e.g., duration between 120sec and 160sec), the overhead of automatic adaptive strategy is slightly more than application-specific strategy. This is because the automatic adaptive strategy provides higher fidelity value.

In order to evaluate adaptation strategies in more dynamic environments, we use a more

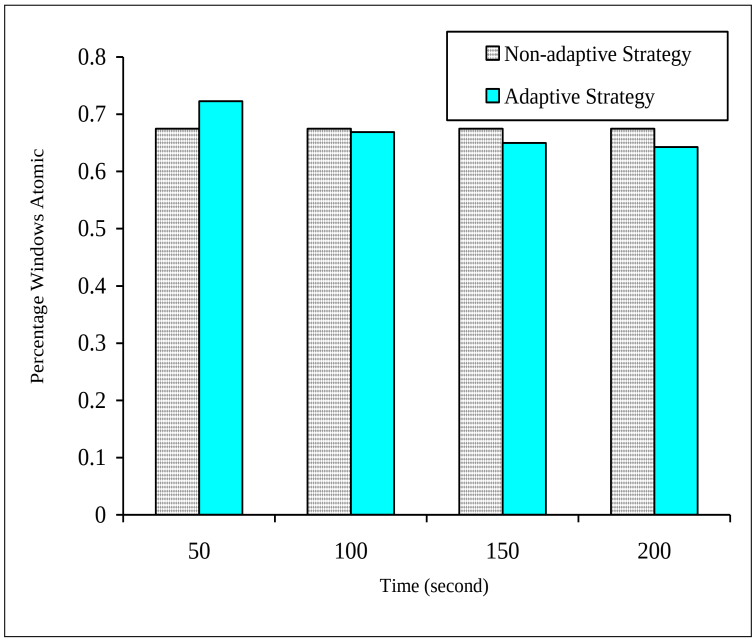


Figure 4.10: Comparison of fidelity for learning-based adaptation and non-adaptive approach

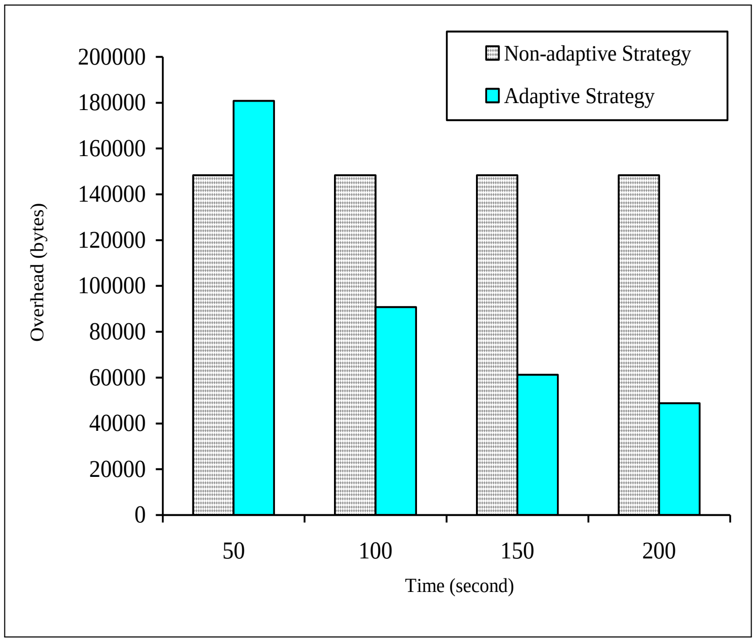


Figure 4.11: Comparison of cost for our learning-based adaptive approach to non-adaptive approach

sophisticated scenario which is from [41]. This scenario is similar as the previous one. However, there are two differences. First, the speed of vehicle will decrease by 5m/s for each forty seconds after the maximum speed, which is 20m/s at the time of one hundred

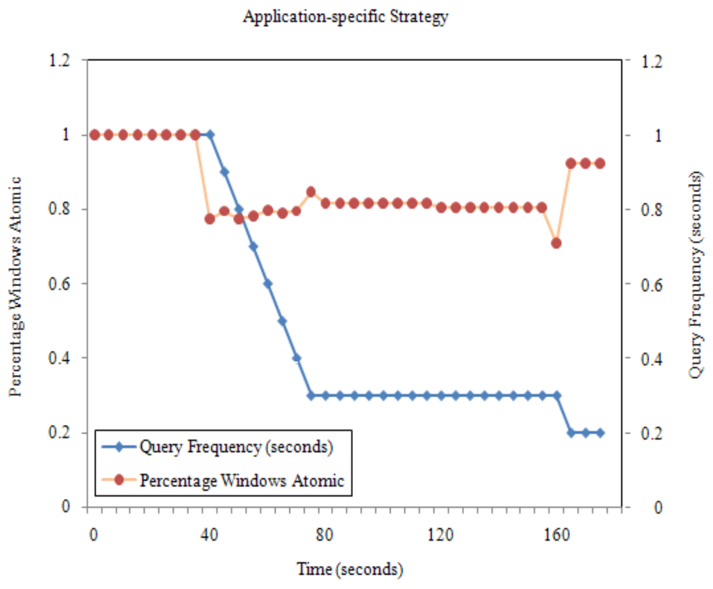


Figure 4.12: Performance of an application-specific adaptive strategy for simple vehicle scenario

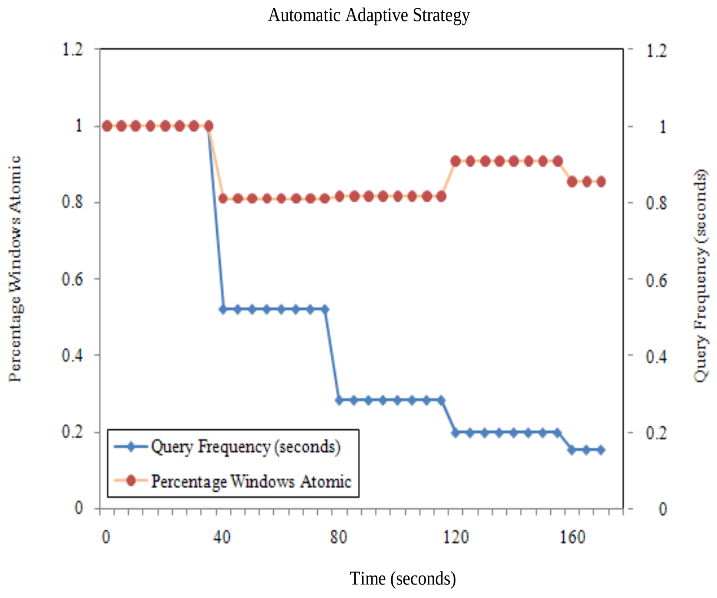


Figure 4.13: Performance of learning-based adaptive strategy for simple vehicle scenario

and sixty seconds. Therefore, the speed will decrease back to 0m/s at the time of three hundred and twenty seconds. The second difference is that the adaptation strategies try to lower the overhead while the fidelity level is maintained after the maximum speed.

Figure 4.16 describes the results of application-specific strategy, which needs to adjust

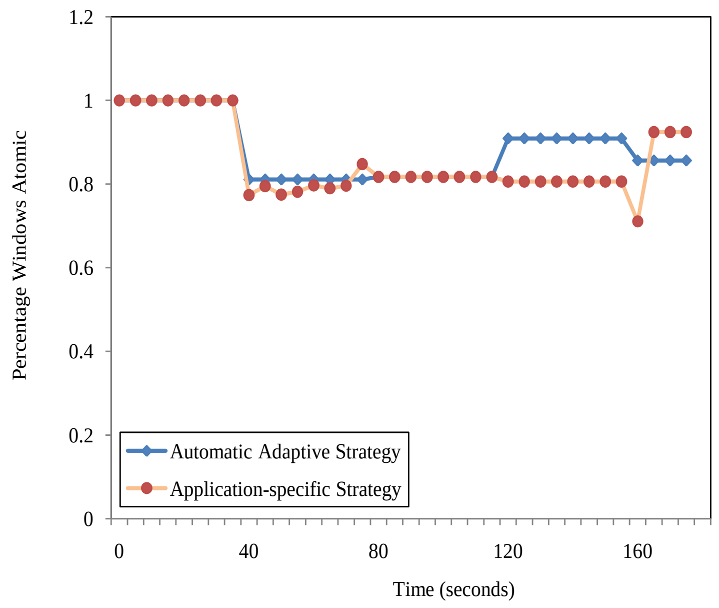


Figure 4.14: Comparison of fidelity for fidelity-based adaptive approach and application-specific adaptation approach for simple vehicle scenario

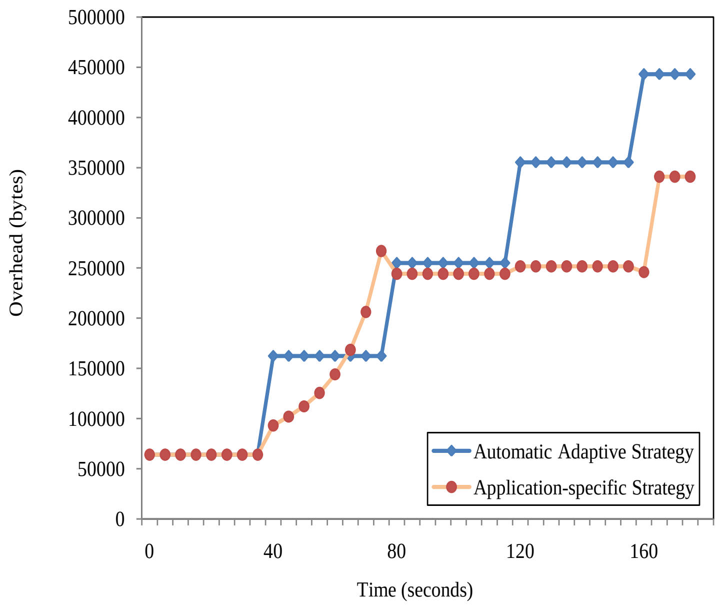


Figure 4.15: Comparison of overhead for fidelity-based adaptive approach and application-specific adaptation approach for simple vehicle scenario

the continuous query frequently in order to maintain the fidelity value and try to lower the execution overhead. The results of automatic adaptive strategy is shown in Figure 4.17. The value of fidelity satisfies the application requirement, which demonstrates that the

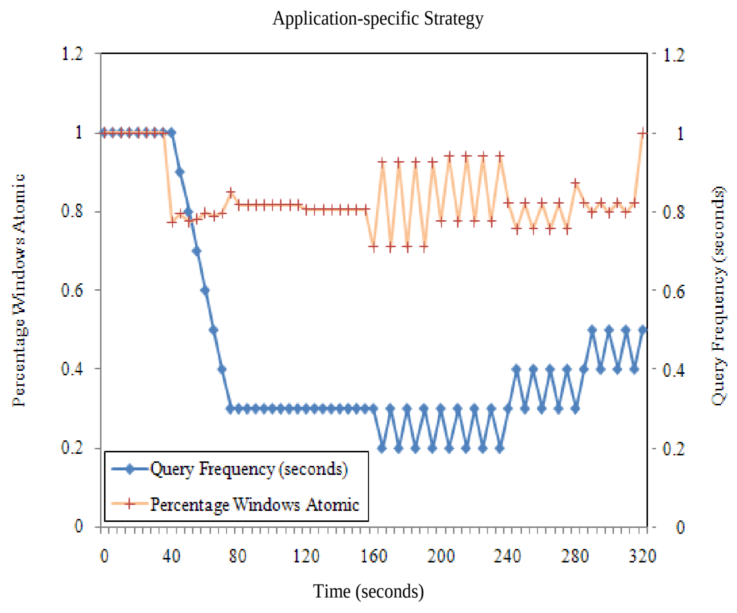


Figure 4.16: Performance of Application-Specific Adaptation for Advanced Vehicle Scenario

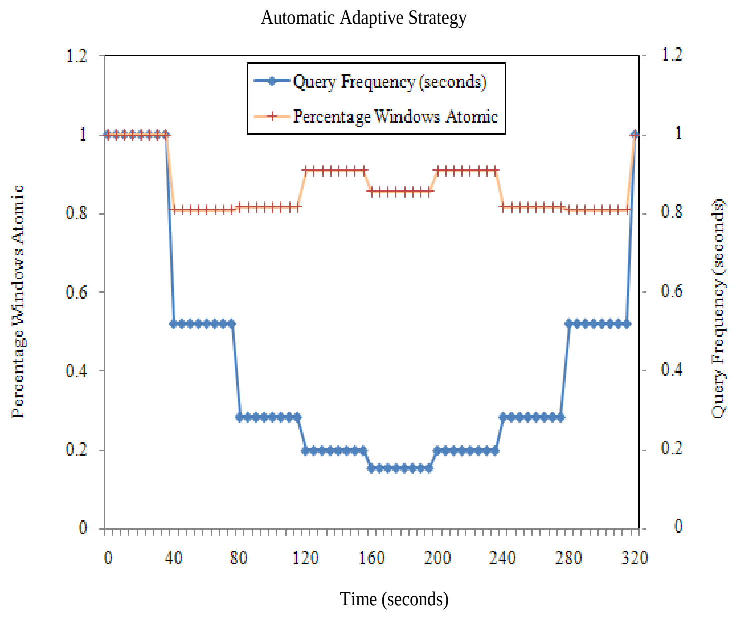


Figure 4.17: Performance of Learning-based Adaptation for Advanced Vehicle Scenario

automatic adaptive strategy does a good job to monitor the evolving conditions and provide suitable query strategies in dynamic environment. In addition, it is observed that the lines of automatic adaptive strategy are steady and have fewer oscillations. The reason is that the automatic adaptive strategy can determine the suitable query strategy directly based on the

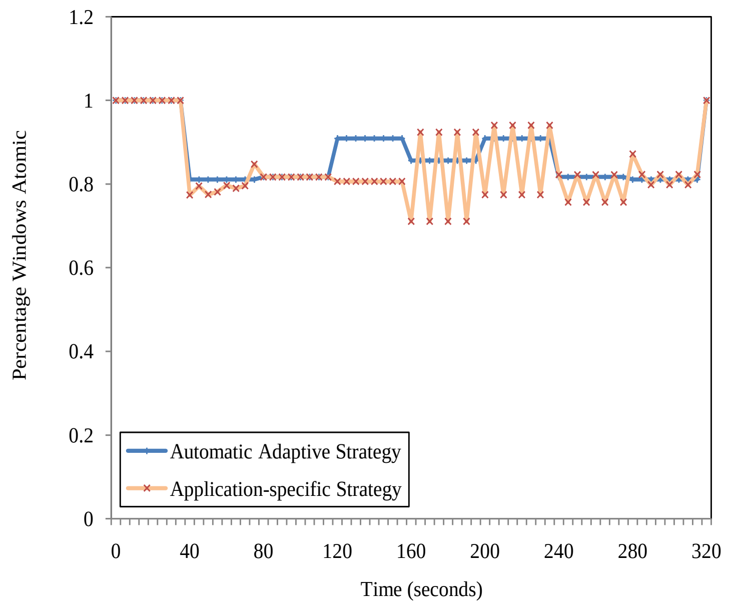


Figure 4.18: Comparison of Fidelity for Learning-based Adaptation and Application-Specified Adaptation for Advanced Vehicle Scenario

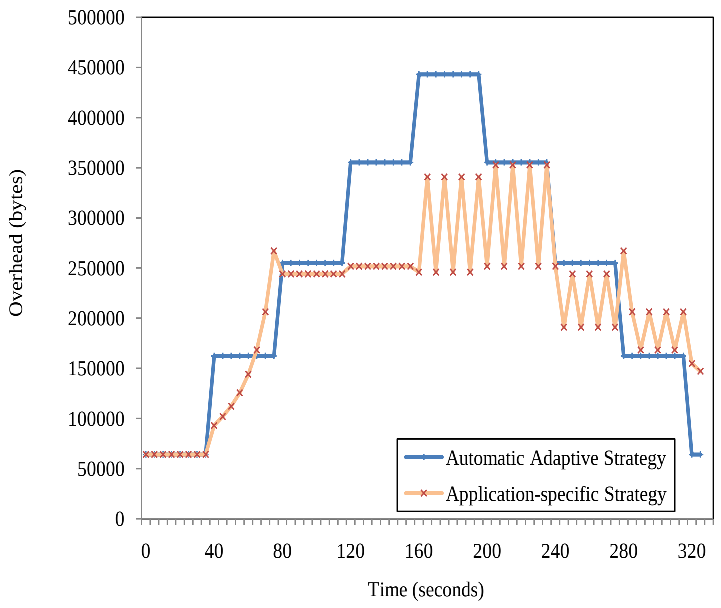


Figure 4.19: Comparison of Cost for Learning-based Adaptation and Application-Specified Adaptation for Advanced Vehicle Scenario

dynamics, while the application-specific strategy needs to periodically test the suitability by changing the query frequency.

Figure 4.18 and Figure 4.19 illustrate the comparison of query quality and execution cost. From those figures, it is notable that automatic adaptive strategy has similar good performance as application-specific strategy, even though the automatic adaptive strategy does not require the insightful knowledge about applications.

4.5 Summary

In summary, the fidelity of continuous query will be affected by the characteristics of executing environments. In MANETs, the network environment is dynamic due to various factors (e.g., evolving topology). Those varying network dynamics give rise to the need of adaptation strategies which should be able to monitor the time-varying network dynamics and adapt queries to satisfy the applications' requirements. In this chapter, we introduce a learning-based adaptation algorithm for governing $when$ and how to adjust continuous queries in order to match the changing conditions in dynamic networks and meet a wide range of requirements' requirements.

The proposed algorithm applies a regression model to capture the relationship between the fidelity sematic and networking environments. Then, the model can be employed for estimating the fidelity value of current continuous queries with the consideration of queries' property and underlying conditions in dynamic environments. Therefore, based on the estimation, the automatic adaptation strategy can adjust the query strategy to meet the applications' requirements. Similarly, with respect to the query execution consumption, similar multiple regression function is proposed to learn the overhead cost of inquiry strategies according to various conditions in operational environments.

The simulation results indicate that our adaptation strategy can be used to learn the query fidelity and execution cost. Our comparison to non-adaptive strategies shows that fidelity-based adaptation using a learned approach provides significant improvement in meeting fidelity and cost tradeoffs. The comparison of our proposed learning-based adaptive strategy to two application-specified adaptation strategies demonstrates that the proposed adaptive algorithm is helpful to determine suitable the continuous strategy for dynamic environments. The performance measurements of the learning-based and application-specified strategies have very similar results, which indicates that the learned adaptive strategy can adequately determine the inquiry strategy that satisfies applications' requirements and reflects the changing conditions in operational environment without requiring the application developer to apply insightful knowledge of relationships between query processing protocols, the conditions of the operational environment, and the resulting quality of the query's execution.

CHAPTER 5: CONTENT-BASED ADAPTATION

The research problem that we explore is how to choose a suitable inquiry strategy in order to adapt a continuous query to the changing environment in such a way that achieves an optimal tradeoff between the quality of the returned results and the cost of the query's execution. In chapter 4, we introduced a learning-based adaptation strategy which uses fidelity of continuous queries to define the quality of the continuous query's execution and uses this definition of quality and a measure of the cost of a query's execution to adjust the inquiry strategy to achieve the optimal quality and cost tradeoff in the current dynamic environment. In this chapter, we extend this approach and explore the use of a new definition of the quality of the query's execution. Here, we employ more information about the contents of the component snapshot queries' results to define the quality of its execution. We then use this definition of quality to learn, or predict, how to choose an inquiry strategy that delivers the optimal tradeoff between quality of query results and execution cost.

5.1 Model Overview

In this section, we provide an extended description of our previously published approach that allows for supporting adaptive continuous queries by learning a general introspection metric that maximizes the quality of query results while minimizing the overhead associated with the query execution [67].

In our approach, we apply an instance-based learning technique to learn how to adapt a continuous query's execution to maximize its quality while minimizing its cost. In order to learn $when$ and how to adapt to a more suitable query protocol, we first define the quality of a query result. In defining the quality of a continuous query's execution, we should consider that it is desirable for our continuous query (comprised of a sequence of snapshot queries) to closely approximate the ideal continuous query. In other words, we want our continuous query to capture as much information as possible about the changes that occur in the environment. We could simply use an inquiry strategy with a high frequency of issue and an inquiry mode that collected information from all nodes (e.g., a flooding query), but message overhead and resource consumption is a concern in networks of mobile devices. Therefore, we want to learn when and how to adapt the inquiry strategy to balance the tradeoff between the "quality" of result (i.e., how well the approximate persistent query reflects the ideal query) and the cost of the query.

We frame this as a numerical optimization problem where the goal is to maximize the difference, $F - \alpha C$, where F is a function that defines this "quality" value, C is the cost of execution using a particular inquiry mode, and α is a scaling constant that reflects the importance of cost. $F(R, R^*, s)$, where R is the set of results (i.e., representations of responding hosts) returned by a one-time query with inquiry strategy s and R^* is the ideal set of reachable results reachable using strategy s, is defined as $|R|/|R^*|$ if s is a flooding strategy and $|R|/(|R^*| * p)$ if s is a probabilistic strategy with probability p. That is, F represents the percent of hosts that, ideally, should have responded.

In reality, however, we cannot compute R^*, and therefore we cannot compute F. So, our approach is to learn a function $\hat{F}$ that is an approximation of F using the history of snapshot query results. To do so, we use an instance-based learning approach to learn $\hat{F}$ and modify our optimization function to $\hat{F} - \alpha C$. We learn this general function offline without requiring the mobile devices to incur the overhead associated with an online learning approach, and then distributed this learned function to mobile devices to use live, within the network, as new queries. Below, we describe the details of this learning-based approach to continuous query adaptation using this new definition of continuous query quality.

5.2 Learning Quality Function

To learn our quality function, $\hat{F}$, which is based on the history of query results, we need to define how to compute the difference between sets of results for successive one-time queries. Each query collects a node's data as well as local properties of the node at the time that the query executed. The result of the i^{th} one-time query, R_i, is a set of hosts that responded to the query. We define a host, h, as a tuple, $(\iota, \nu, \lambda, \omega, \epsilon)$ where ι is a unique node identifier, ν is a data value, λ is the node's location, ω is the node's velocity, and ϵ is a measure of the remaining energy. Our approximation of quality of the query's execution, then, is defined as a function of the difference between the query results for R_i and R_{i-1}.

As a first step, we propose simple metrics for computing the difference between successive query results; we expect that more complex metrics will be developed and can be applied using our method. In our metrics, we compare values of nodes that provided results in successively issued one-time queries. We define M as the set of nodes that contribute results both in R_i and R_{i-1}: $M = \{h_x \in R_i \bigcap h_y \in R_{i-1} : h_x.\iota = h_y.\iota\}$. For each component of the host tuple (excluding the node identifier), we use the variance of the values in M as input to $\hat{F}$. This gives us $\sigma(\nu_M)$, $\sigma(\lambda_M)$, $\sigma(\omega_M)$, and $\sigma(\epsilon_M)$ as parameters. In addition, we calculate a matching score $m = \frac{|M|}{|R_{i-1}|}$ which penalizes for differences between the sets of responding nodes. Other measurements, such as higher-order statistics or domain-specific metrics, may prove to be useful.

To learn $\hat{F}$ by example, we run a series of one-time queries in simulation. For each query, we compute m, $\sigma(\nu_M)$, $\sigma(\lambda_M)$, $\sigma(\omega_M)$, and $\sigma(\epsilon_M)$. We also compute the value of F using the "oracle" view of the network provided by the simulator. To compute the approximation $\hat{F}$, we use radial basis function (RBF) network [60, 71] which is a type of feed-forward network contains three layers: input layer, hidden layer, and output layer.

Figure 5.1 shows the architecture of RBF network. The input layer provides instances and performs no computations. The hidden layer has k nodes with the center c_k. The processing at the middle layer is to perform a nonlinear mapping from the input vector to a higher dimensional space by computing the distance from the input space to the corresponding center and applying related real-valued basis function. The most commonly used real-valued basis function is a Gaussian function. The output layer of RBF network is a linear function $g = (f_1, f_2, ..., f_i)$, which performs a simple weighted sum with combining the resulting scalar value of the hidden layer.

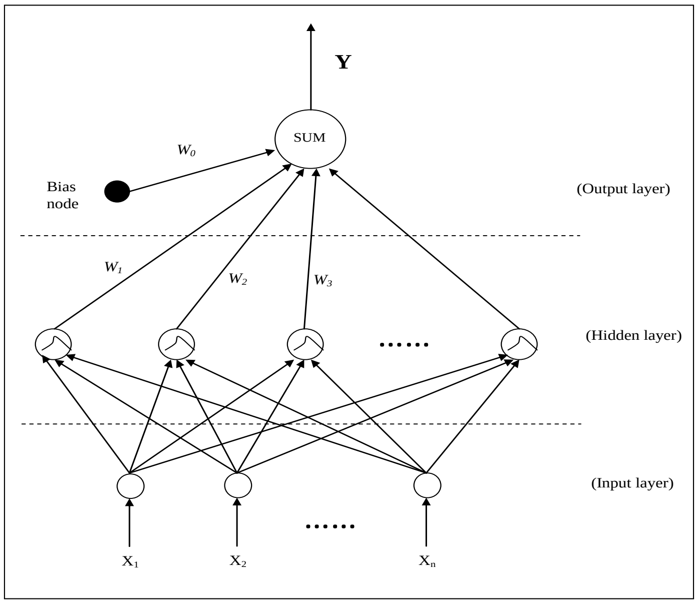


Figure 5.1: RBF network architecture [32]

To compute the approximation $\hat{F}$, we apply RBF network [60, 71] defined as:

$$\hat{F}(\vec{x}, s) = \sum_{j=1}^{N} w_j \phi(||\vec{x} - \vec{z}_j||) + b \tag{5.1}$$

where $\vec{x}$ is a vector whose components are m, $\sigma(\nu_M)$, $\sigma(\lambda_M)$, $\sigma(\omega_M)$, and $\sigma(\epsilon_M)$, s is an inquiry strategy, b is a bias vector, w_j is the real-valued weight of kernel center $\vec{z}_j$, for $j \in 1, 2, ..., N$, and ϕ is a real-valued *basis function*. In our algorithm, we choose the Gaussian function for ϕ: $\phi(r) = e^{-r^2/2\sigma_w^2}$, where σ_w is the average intra-center distance.

As the last step, the learned function $\hat{F}$ is embedded into the node of the query issuer. The query issuer makes decisions regarding $when$ and how to adapt the inquiry strategy to best fit evolving conditions of the environment by selecting a specified query protocol and particular frequency to maximize $\hat{F} - \alpha C$.

5.3 Evaluation

To analyze the performance of our automated adaptation approach, we have developed a prototype implementation of protocols using the OMNeT++ network simulator [92], its mobility framework extension [52], and a battery module component [27]. Query protocols are executed in simulation and information is collected from their executions. We use the view of the network provided by the simulator to compute F-value. For these query instances, we split them into a training set and testing set. In the training set, we employ properties of nodes to define the values of parameters for F-value and to define the cost of the query's execution. Then we apply RBF network to the training instances to learn $\hat{F}$-value and evaluate the performance in the testing set. In this section, we illustrate two of the most important characters of protocols: F-value and cost. Then we use statistical methods to evaluate how well $\hat{F}$ estimates F. Finally, the performance of our adaptation strategy is presented.

A. Simulation Environment

We execute these queries in a MANET environment where a set of nodes moves in a rectangular area with the size of 1200$\times$1000 m^2. At the beginning, a varying number of nodes are randomly deployed in the space to form an ad-hoc network. Then the nodes move according to the "random waypoint" model [12] at a given speed, in which each node chooses a destination in the space randomly and moves to the selected destination at a given speed between 0 m/s and 30 m/s. When a node reaches the destination, it pauses for a pause time, chooses another random destination, and repeats the process. We run our simulations with a pause time of 0s to represent continuous motion.

1) Variables: We evaluate query protocols and the adaptive strategy thoroughly under varying environmental conditions. Specifically, we varied three simulation parameter settings: number of nodes, degree of mobility, and query frequency. First, the number of nodes is varied from 25 to 85 at increments of 10. As more nodes are placed in the space, the network density increases, since the area size is constant. Second, the average speed of nodes varies from 0 m/s (completely static) to 30 m/s (high degree of mobility) in multiples of 10. The final variable is query frequency. We control the query frequency by adjusting the interval time between two successive query processes. When the interval is short, the query frequency becomes high. In the simulations, there are three different query intervals: 0s, 0.5s, and 0.75s.

2) Metrics: Our results are evaluated using the following metrics: (1) Overhead is the number of bytes transmitted when using a particular query protocol. We use overhead to define the energy consumption of a query. (2) P-value indicates a statistical measure for the probability of how much evidence we obtain against the null hypothesis, which is used to measure how well $\hat{F}$-value estimates the ideal F-value. In statistical hypothesis testing, P-value is used to measure the difference between groups, which is ranging from zero to one.

B. Performance Evaluation

1) F-value comparison: Figure 5.2 shows the results of F-value for flooding query. It is obvious that using a higher query rate achieves much higher quality of query results. In addition, as the network density increases, the F-value of flooding protocol decreases. This is most likely due to the increased number of messages dropped due to collisions as many more nodes are competing to access the shared medium to return their query results.

Figure 5.3 presents experimental results where the number of nodes varies from 25 to 85 with a query interval of 0.5s. Several things are notable about these results. First, a flooding protocol is very likely to reach every node in the network and yields the most information about the environment. In all cases, the F-value of the flooding protocol is significantly higher than the probabilistic protocols. Second, the F-value of all probabilistic protocols increases as the network density increases. The reason is that probabilistic protocols cannot reach most of the nodes due to limited connections when the network is sparse. The similar graphs for query intervals of 0s and 0.75s are illustrated in Figure 5.4 and Figure 5.5.

2) Energy consumption comparison: Another aspect of our evaluation is to assess the

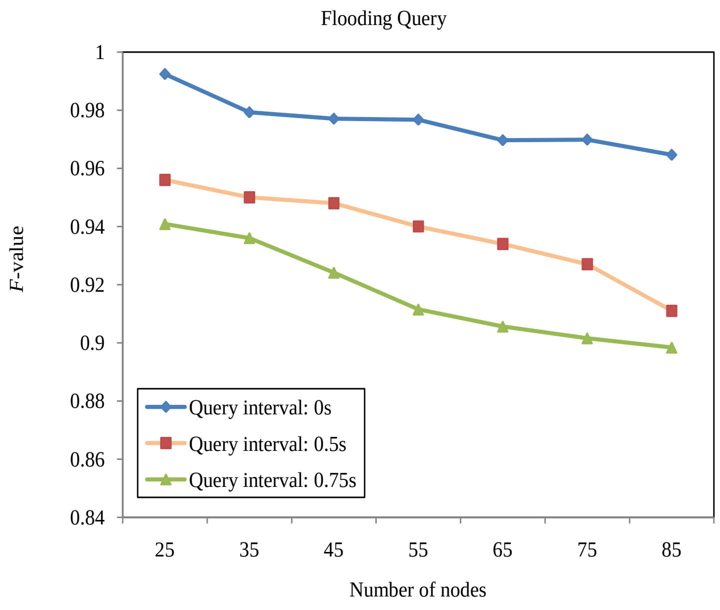


Figure 5.2: F-value vs. number of nodes for flooding query

cost of the query processing. We measure protocol cost as the number of bytes transmitted over the simulation time. We employ overhead to represent the cost of a particular query protocol execution because the energy consumption by the network is dependent on the number and size of transmissions. We show cost comparison by increasing the number of network nodes from 25 to 85, thus increasing network density. Figure 5.6 shows the overhead results of flooding query. It is notable that using a higher query rate results in much higher execution cost.

Figure 5.7 shows one case of how overhead is affected by the network density. As expected, the flooding approach is expensive in terms of the message overhead. Additionally, overhead of all protocols increases as the network density increases, because more messages are sent across the network. The similar trends for query intervals of 0s and 0.75s are shown in Figure 5.8 and Figure 5.9.

3) $\hat{F}$-value estimation: $\hat{F}$-value estimation is also important to the evaluation. It demonstrates that the approach is able to learn a reasonable approximate $\hat{F}$-value of the quality function under different conditions. For each case in Tables 5.1, 5.2, and 5.3, the experiment is based on a training set of over 5000 records, and a testing set of over 500 records.

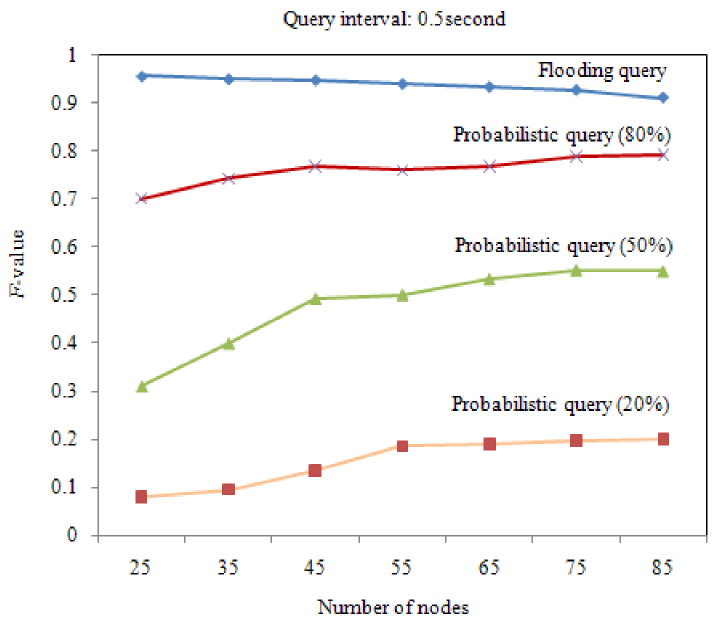


Figure 5.3: F-value vs. number of nodes at a query interval of 0.5s

RBF networks were trained by employing instance based learning algorithm and using these instances in the training set. Then, the RBF network is used to approximate the $\hat{F}$-value in the testing set. We compare the average of actual F-values in the testing set with the approximated $\hat{F}$-value by RBF network. Tables 5.1 shows that their average values are very close, which determines that RBF network performs well. The fourth column of both tables represents P-value (two-tailed t-Test) for comparing F-value and $\hat{F}$-value based on the testing set. A P-value of 0.05 or below is conventionally accepted as the standard to determine a significant difference between two variables. For every row, the P-value is much higher than 0.05, corresponding to a 5% chance. Hence, $\hat{F}$-value is significantly similar to F-value statistically.

To evaluate the adaptation ability of our proposed approach, we need to compare the tradeoff between energy consumption and the contained quality of inquiry results with other non-adaptive query strategies. We conduct such comparisons in the following part.

4) Performance of adaptive strategy: We compare the features of our adaptive strategy with several non-adaptive query approaches. Figure 5.10(a) shows the F-value of the adaptive strategy is not very high. The reason is that the adaptation approach needs to consider

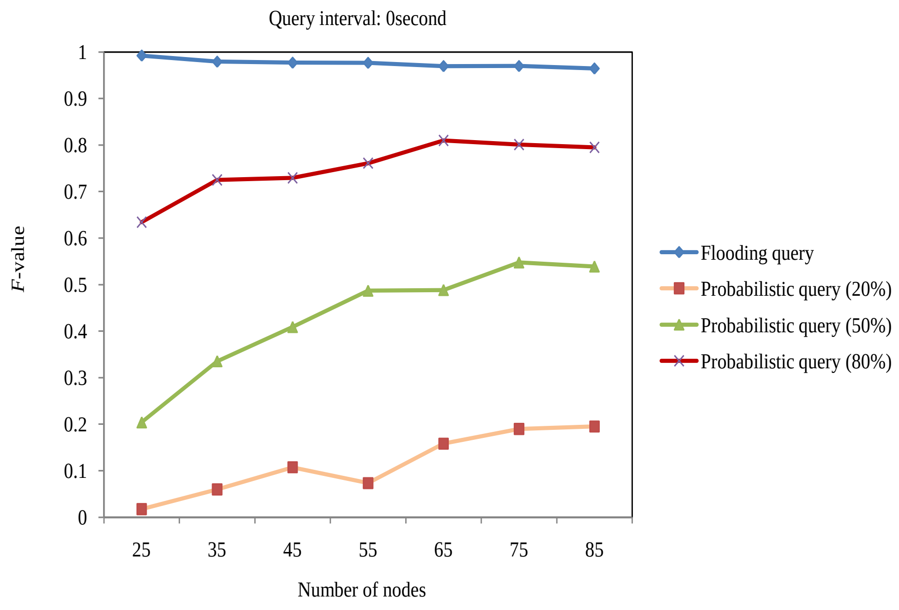


Figure 5.4: F-value vs. number of nodes at a query interval of 0s

Table 5.1: F-value vs. $\hat{F}$-value when query interval is 0s

Number of Nodes	Average of F-value	Average of $\hat{F}$-value	P-value (t-Test)
25	0.425311	0.432090	0.599974
35	0.440666	0.434425	0.576317
45	0.504385	0.496195	0.548671
55	0.511883	0.507329	0.751748
65	0.478845	0.475198	0.796113
75	0.491240	0.488585	0.846708
85	0.465165	0.502068	0.624546

the tradeoff between quality of query results and resource consumption of query processing. At first glance, the flooding query obtains high quality value. However, the flooding approach is associated with the high cost of energy resource. Figure 5.10(b) highlights one advantage of employing adaptation: lowering overhead, especially as the scaling factor of cost increases. In order to evaluate the adaptive ability to maximize $\hat{F} - \alpha C$, we normalize the overhead to a range of [0, 1]. Figure 5.10(c) illustrates comparisons of $\hat{F} - \alpha C$ for different scaling factors. From the results, the adaptive strategy achieves a better tradeoff than other non-adaptive approaches. A similar effect can be observed in Figure 5.11 and

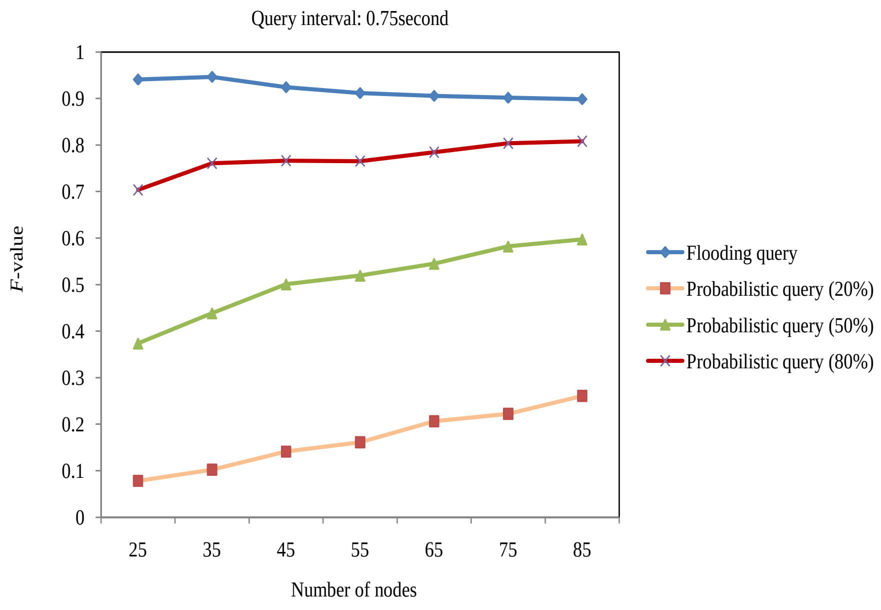


Figure 5.5: F-value vs. number of nodes at a query interval of 0.75s

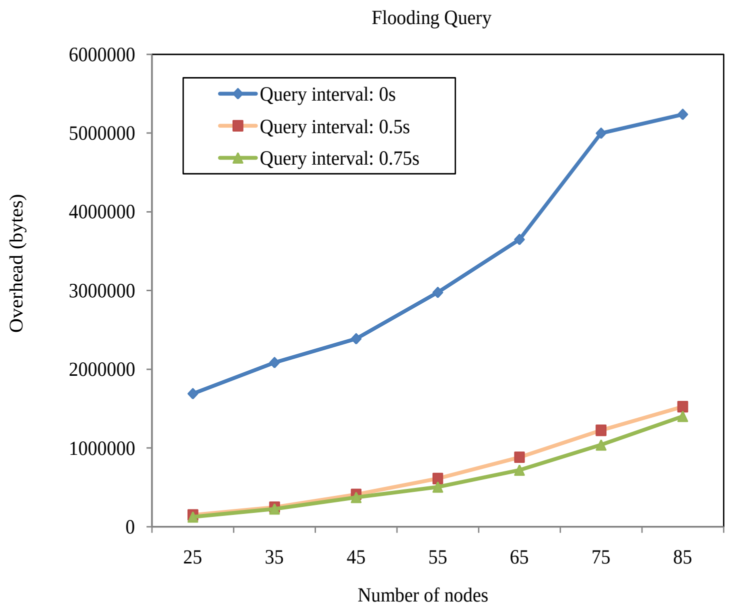


Figure 5.6: Overhead vs. number of nodes for flooding query

Figure 5.12 as the query intervals are increased to 0.5s and 0.75s.

By comparing with non-adaptive strategies, it is clear that employing a mechanism for adaptive strategy is beneficial to query processing, which allows applications to achieve high optimization values by considering performance tradeoffs in query quality and energy

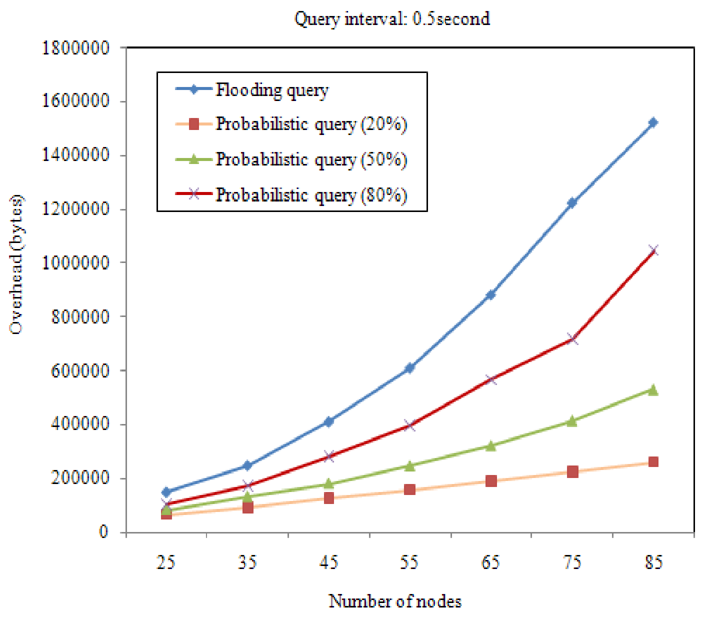


Figure 5.7: Overhead vs. number of nodes when query interval is 0.5s

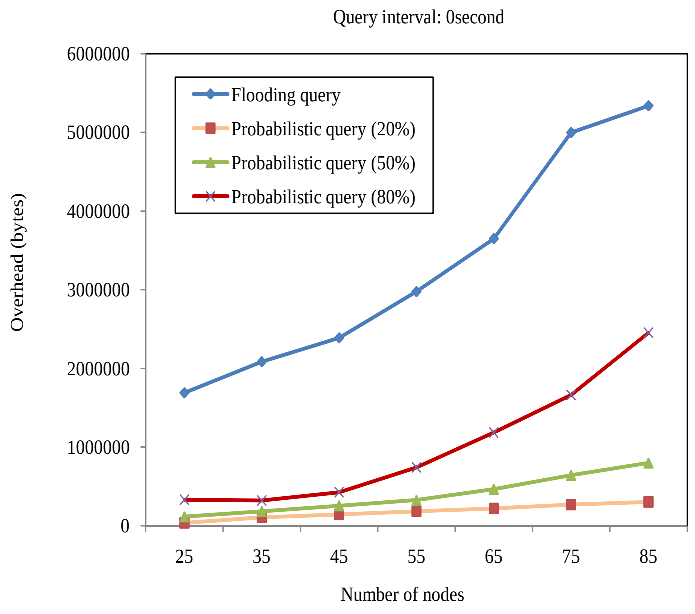


Figure 5.8: Overhead vs. number of nodes when query interval is 0s

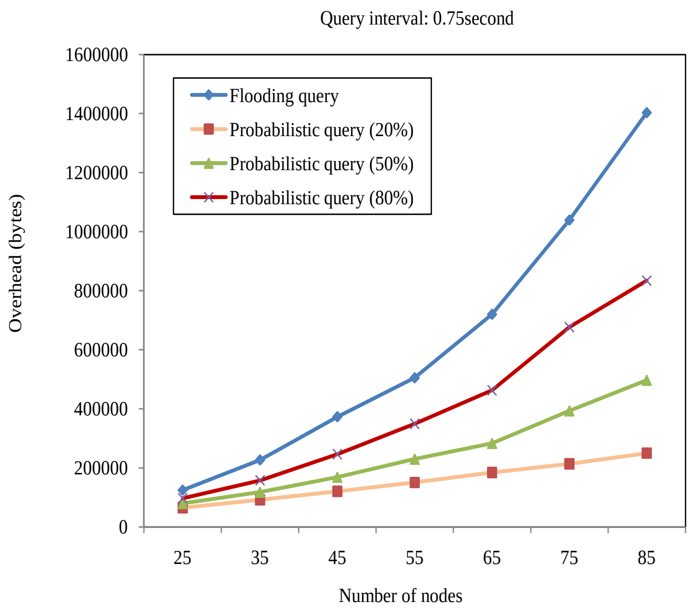


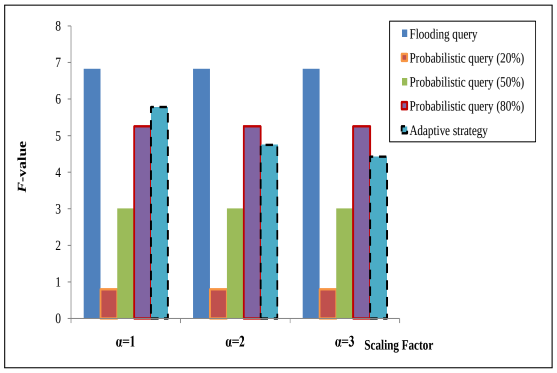
Figure 5.9: Overhead vs. number of nodes when query interval is 0.75s

Table 5.2: F-value vs. $\hat{F}$-value when query interval is 0.5s

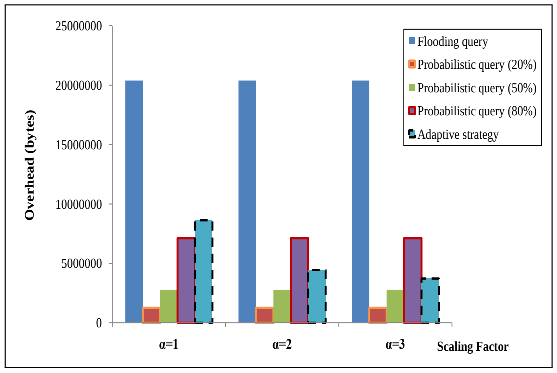
Number of Nodes	Average of F-value	Average of $\hat{F}$-value	P-value (t-Test)
25	0.528400	0.532898	0.783495
35	0.565611	0.554972	0.611345
45	0.550201	0.538628	0.566882
55	0.602899	0.609711	0.614535
65	0.607966	0.612313	0.880611
75	0.620935	0.628241	0.626335
85	0.606296	0.621057	0.618314

consumption.

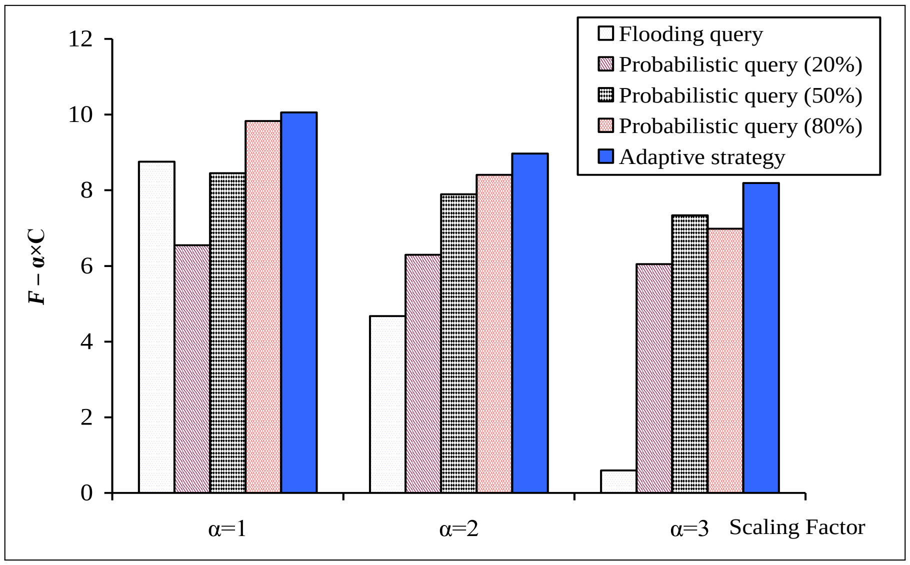
In the previous simulation, we evaluate the performance of automatic adaptation strategy by comparing with non-adaptive strategies. We now show the adaptive ability of automatic adaptation strategy by comparing with application-specific adaptation strategy. Consider an application on executing a continuous query. Initially, the nodes have enough energy. So the application-specific strategy chooses flooding query to achieve higher quality of query results. Then, some energy is used for the query processing. Therefore, the application-specific strategy selects probabilistic query (80%). Finally, with the increase of time, much



(a) Comparison of F-value

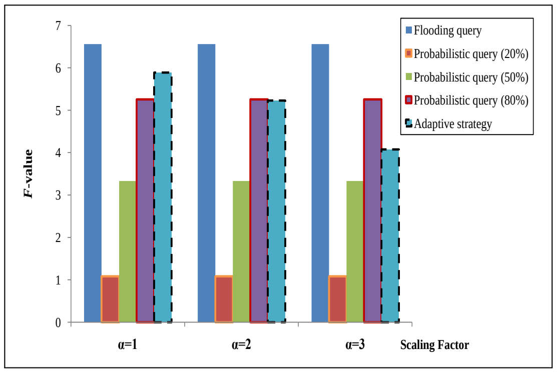


(b) Comparison of overhead

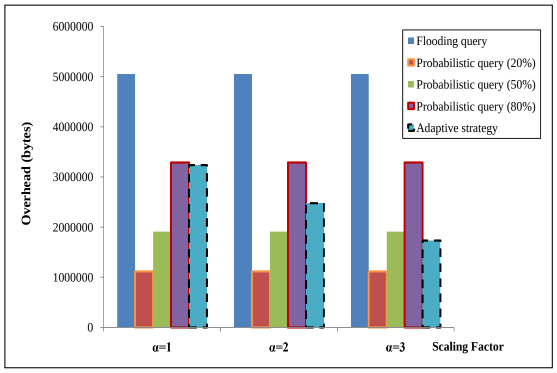


(c) Comparison of $F - \alpha \times C$

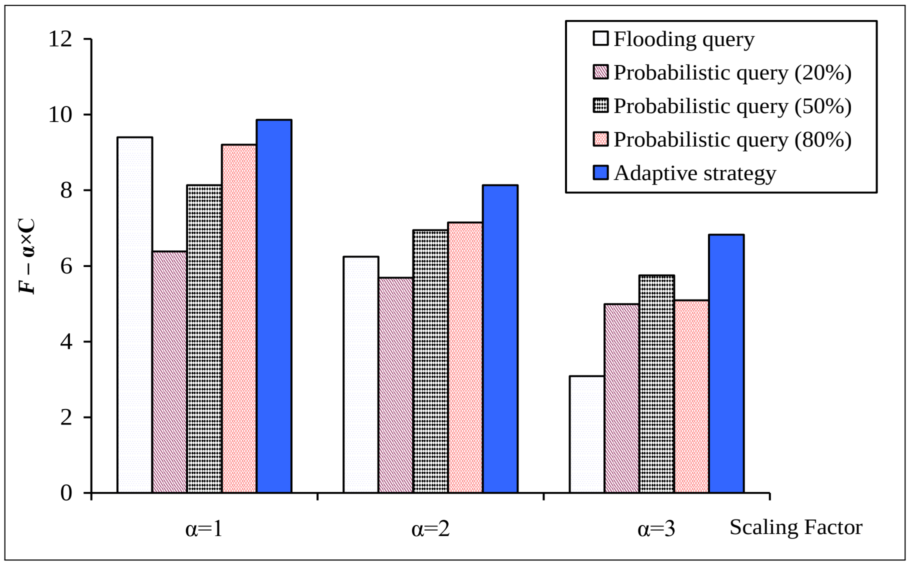
Figure 5.10: Performance under query interval with 0s



(a) Comparison of F-value

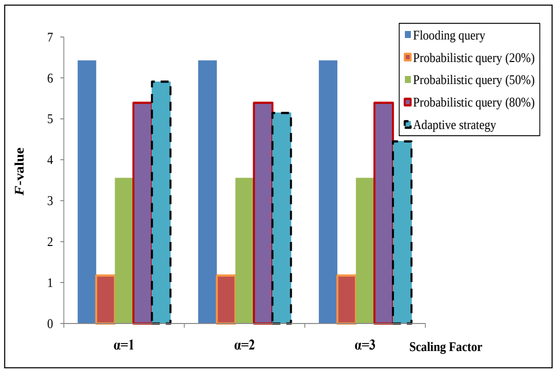


(b) Comparison of overhead

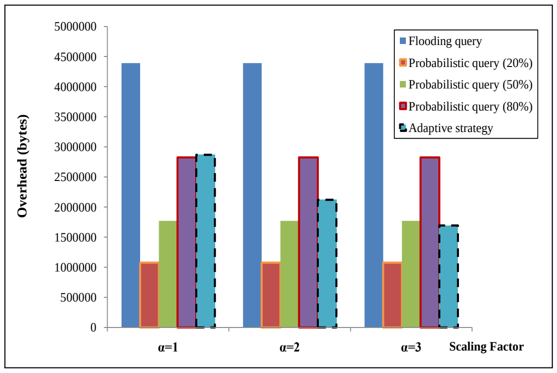


(c) Comparison of $F - \alpha \times C$

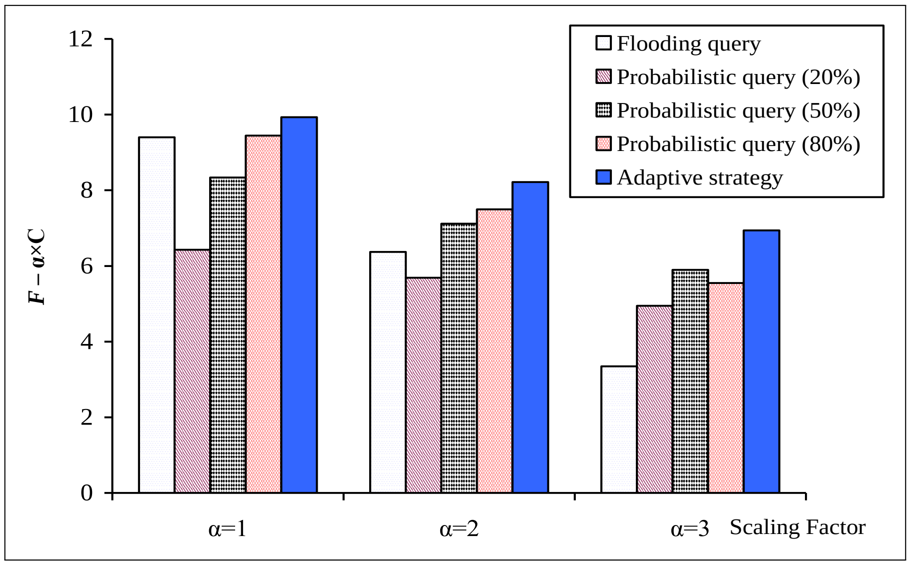
Figure 5.11: Performance under query interval with 0.5s



(a) Comparison of F-value



(b) Comparison of overhead



(c) Comparison of $F - \alpha \times C$

Figure 5.12: Performance under query interval with 0.75s

Table 5.3: F-value vs. $\hat{F}$-value when query interval is 0.75s

Number of Nodes	Average of F-value	Average of $\hat{F}$-value	P-value (t-Test)
25	0.547942	0.536030	0.334459
35	0.538138	0.541385	0.708624
45	0.582139	0.581059	0.926514
55	0.580499	0.584898	0.679015
65	0.603104	0.606496	0.739265
75	0.614598	0.620894	0.514887
85	0.626109	0.632088	0.530319

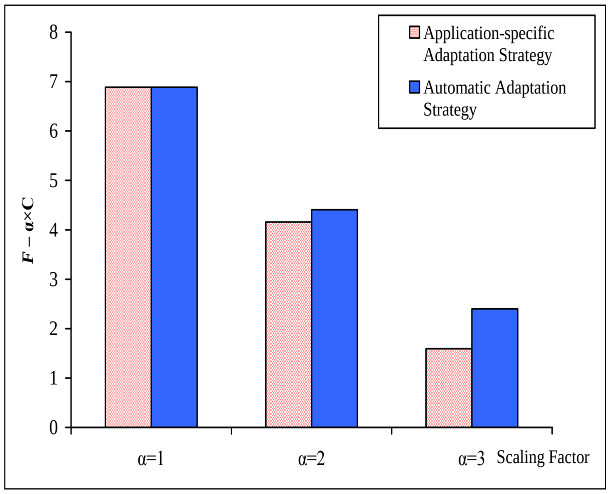


Figure 5.13: An application example (performance of automatic adaptation strategy)

more resource is used. Therefore, the probabilistic query (50%) should be chosen in order to save energy. The Figure 5.13 illustrates the performance of automatic adaptation strategy, which is similar to the application-specific strategy. In some cases, the automatic adaptation strategy performs better because it learns the dynamics and tries to achieve the optimal tradeoff between quality and cost, while the application-specific strategies are static and pre-defined.

5.4 Summary

In this chapter, a second learning-based approach to adaptive continuous processing is presented; this approach introduces and uses a new definition of snapshot query quality, defined by the match function, in order to determine how to adapt for quality and cost tradeoffs in dynamic environments. Our simulation results show the ability of this approach to select a suitable inquiry strategy for changing operational conditions in order to optimize the quality and cost tradeoffs for continuous query execution. Our comparison to a non-adaptive approach shows that the adaptive approach provides more favorable results, in terms of quality and cost, in dynamic environments.

CHAPTER 6: ADAPTIVE QUERY MIDDLEWARE

Inquiry strategies can be employed by applications for retrieving related information or monitoring the surrounding environments. Different inquiry strategies are associated with different degrees of quality and costs, and their applicability for particular environmental conditions may differ. Therefore, one challenging issue is how to choose the most appropriate inquiry strategy for use in a particular environment. Compounding the issue is the fact that the environment changes over time. So, the real challenge is to determine how to determine a suitable query strategy which satisfies the applications' requirements or achieves the optimal tradeoff between the query quality and cost given the current conditions of the execution environment.

Asking application developers to make the connections between inquiry strategies, environmental conditions, the achieved quality of the query, and the cost of its execution introduces a significant burden. The previous chapters of this dissertation proposed a learning-based approach that will automatically adapt the continuous query based on the optimal tradeoff between quality and cost given the environmental conditions. In this section, we propose to provide programming support to application developers to simplify and promote the use of our new learning-based adaptive continuous query approach. We introduce a new component to PAQ [77], which is a middleware to support applications that issue adaptive continuous queries for pervasive computing applications. This new component provides the two previously described approaches to learning-based adaptation, and provides a simple application programming interface (API) that allows developers to extend the middleware to include new learning-based adaptive approaches. These new approaches may use new definitions of query quality, different definitions of cost, may incorporate different environmental factors that impact quality and cost, and may apply different machine learning techniques to learn the optimization of quality minus cost.

6.1 Overview

To develop a diverse set of adaptive querying approaches for use in dynamic networks, software development support is needed. In order to address these issues and support the rapid development of different adaptive inquiry strategies, Rajamani et al. introduced the Persistent Adaptive Query (PAQ) middleware [77]. The PAQ middleware provides programming abstractions that allow application developers to issue continuous queries and to construct application-specific strategies for adapting their execution.

The original version of the PAQ middleware is shown in Figure 6.1.

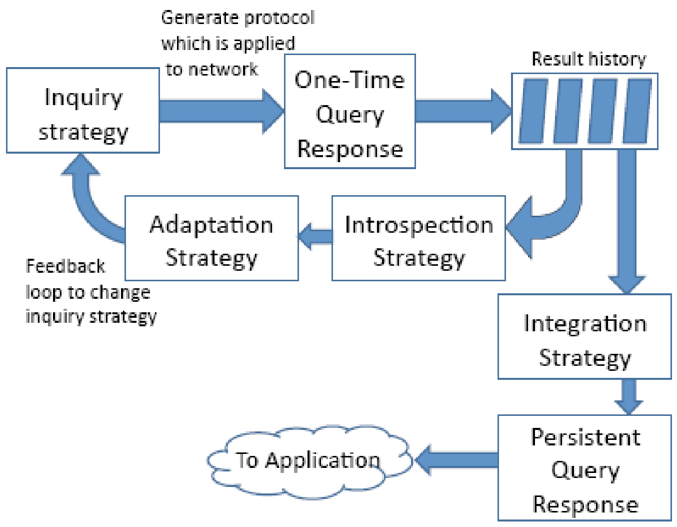
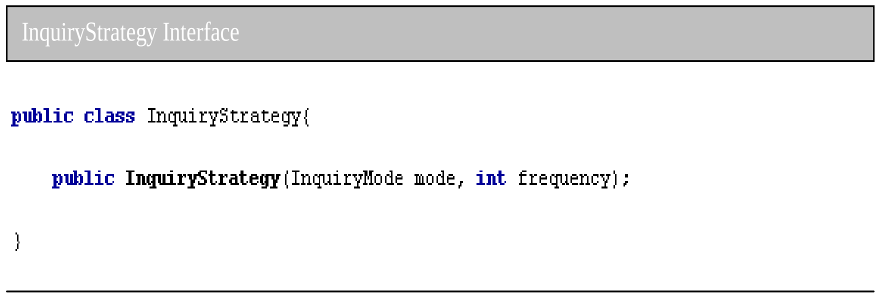


Figure 6.1: PAQ Framework [77]

The inquiry strategy is defined as the composition of query protocol and the frequency of issuing queries, which is expressed as (τ, $freq$). τ is the query protocol which is the implementation of one-time query and $freq$ is the invocation frequency of one-time query. The results of the snapshot queries can be integrated into a continuous query result, or can be evaluated by applying an introspection strategy. PAQ's adaptation strategy component allows an application programmer to specify application-specific adaptation strategies, which describe *when* and *how* the query protocol should adapt, based on the results of applying an introspection strategy to a history of snapshot query results. We extend the PAQ middleware to include a learning-based adaptation component.


InquiryStrategy Interface

```
public class InquiryStrategy{

   public InquiryStrategy(InquiryMode mode, int frequency);

}
```

Figure 6.2: Inquiry Strategy Interface [77]

In the PAQ middleware, the API provides an abstract definition of these various elements of query processing, allowing the developer to easily create new implementations of adaptive continuous queries. This makes the middleware more general, flexible, and reusable. The Figure 6.2 presents the API of inquiry strategies in the PAQ middleware. Several implementation examples of the inquiry strategy interface are shown in Figure 6.3. New inquiry strategies can easily be defined by providing implementations of these abstract classes.

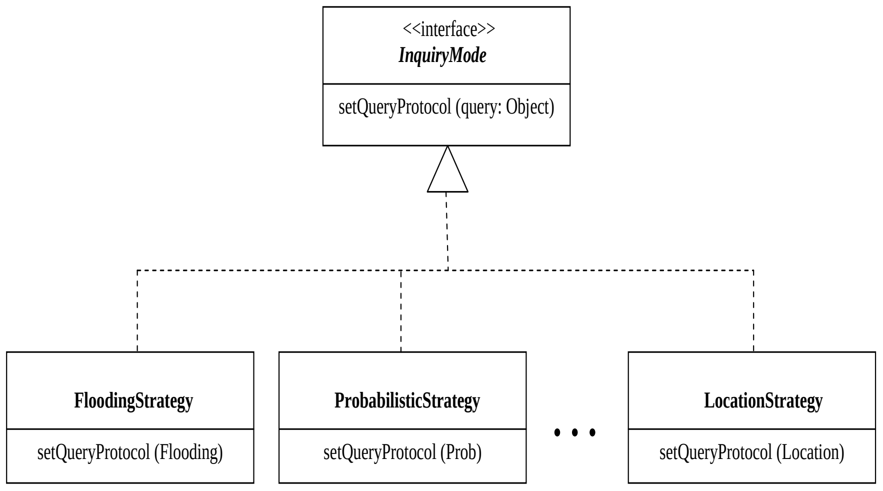


Figure 6.3: Sample Implementations of Inquiry Strategy Interface

In addition, the middleware defines introspection strategies which assess the aggregation results of a sequence of one-time queries; those mechanics can be used to provide evaluation for previous queries. Application-specific thresholds are provided in middleware, which will be triggered for adaptation strategies to adjust the query's processing. As such, the PAQ middleware can serve as a platform to support development and implementation of pervasive computing applications which need to issue queries and collect updated views of the operational environment.

Hence, PAQ middleware is significantly helpful for expressing adaptive querying applications and reducing the burden of programming work. However, it requires the programmers to have significant knowledge about query quality and cost and the relationship to the operational environment in order to generate correct introspection thresholds and create suitable adaptation strategies.

Instead of relying only on application-defined adaptation strategies, we design a new component which encapsulates implementations of the two adaptive algorithms and enables new methods of adaptation. The component will be able to connect to libraries for utilizing existing machine learning methods to evaluate queries' results and determine suitable query strategies across the dynamic networks. Below, we describe our new component to support learning-based adaptive continuous query processing in PAQ.

6.2 Automation Adaptation Component

As discussed in previous section, the PAQ middleware provides several software programming abstractions in order to simplify the development tasks of query processing applications. In addition, the adaptation strategies component includes a set of adaptation policies, which identifies how the inquiry strategy should be altered.

The abstractions provided by the original PAQ middleware are very helpful for query-based applications. However, the query adaptation model requires the application developer to have insights and knowledge about the relationships between inquiry strategies, execution cost, query quality, and the dynamics of the environment. To address this challenge, we design and develop automation adaptation component, which provides services

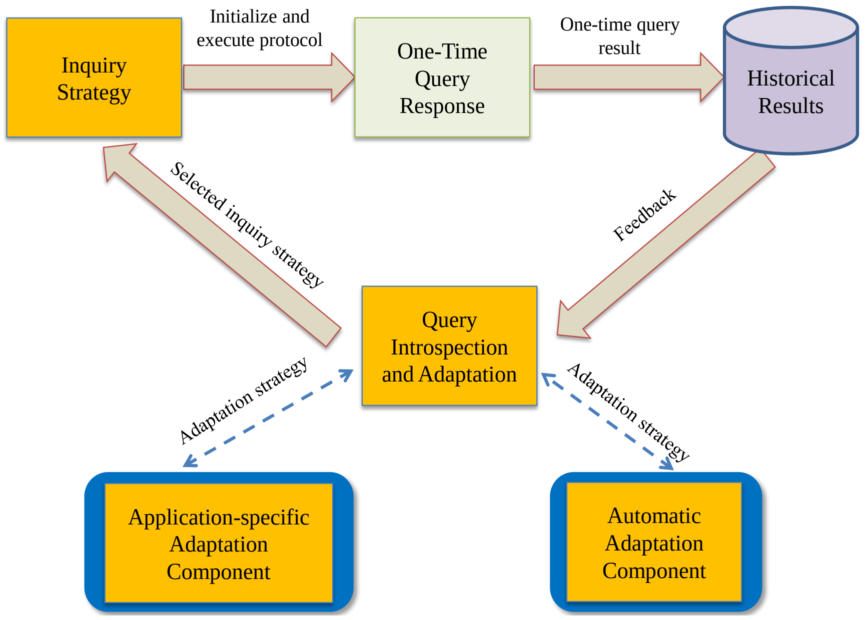
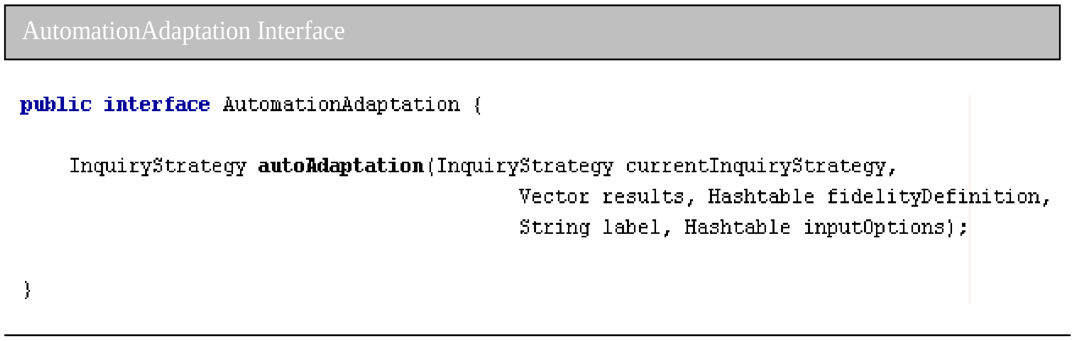

Inquiry Strategy
Initialize and execute protocol
One-Time Query Response
One-time query result
Historical Results
Selected inquiry strategy
Feedback
Query Introspection and Adaptation
Adaptation strategy
Adaptation strategy
Application-specific Adaptation Component
Automatic Adaptation Component

Figure 6.4: The New Framework of PAQ Middleware

for query adaptation model. The new framework of PAQ middleware is shown in Figure 6.4. Based on historical query results, the automation adaptation component applies machine learning algorithms to analyze current environmental conditions and determine the optimal suitable inquiry strategy to address the quality versus cost tradeoff for the next snapshot query as part of the continuous query's execution.



AutomationAdaptation Interface

```
public interface AutomationAdaptation {

    InquiryStrategy autoAdaptation(InquiryStrategy currentInquiryStrategy,
                                   Vector results, Hashtable fidelityDefinition,
                                   String label, Hashtable inputOptions);

}
```

Figure 6.5: The Interface of Automation Adaptation

To allow application developers to extend the middleware, we follow the open-closed principle of software engineering and utilize the interface and inheritance to incorporate new learning-based adaptation approaches to continuous query processing. The interface

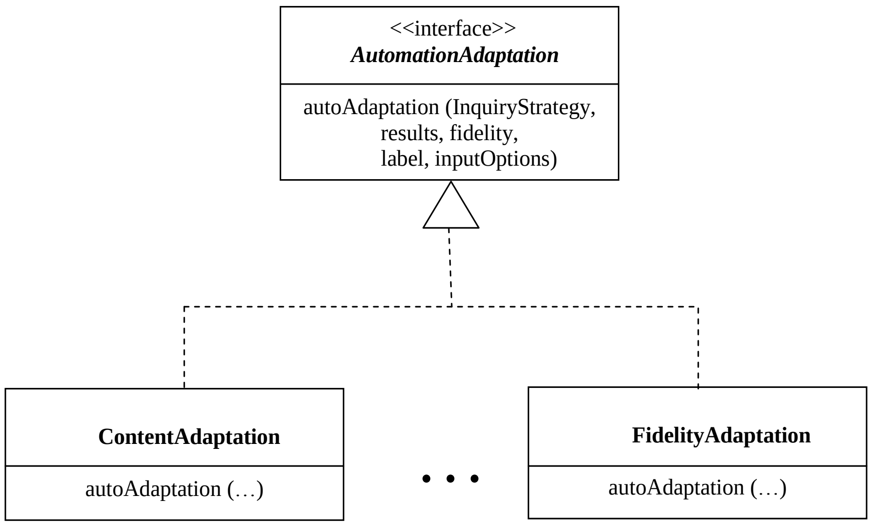


Figure 6.6: Automation Adaptation Interface and Sample Implementations

that new learning-based adaptation components must implement is presented in Figure 6.5.

In addition, the Figure 6.6 presents two implementation examples of automatic adaptation strategies: linear regression approach and radial basis function network. Those adaptive algorithms are introduced in chapter 4 and chapter 5.

The PAQ implementation and the extension presented here has been designed and deployed on the Java SUN SPOT [88] platform, which is a small sensing device provided by Oracle that supports the use of the Java programming language. The SUN SPOT API provides mechanisms for interacting with on-board sensors and for supporting various network architectures and wireless communication protocols. The PAQ middleware architecture deployment on the Java Sun SPOT platform is shown in Figure 6.7. The SUN SPOT base station is connected to a fixed, resource-rich computing device through a USB connection, and the SUN SPOT devices can communicate with SUN SPOT base station via wireless communication. The task of the AutoAdaptLearner in the base station is to learn functions that optimize the tradeoff between query quality and execution cost to implement an adaptive continuous query. The trained adaptation strategies will be deployed into mobile devices which receive the queries and return the related results back to the query issuer.

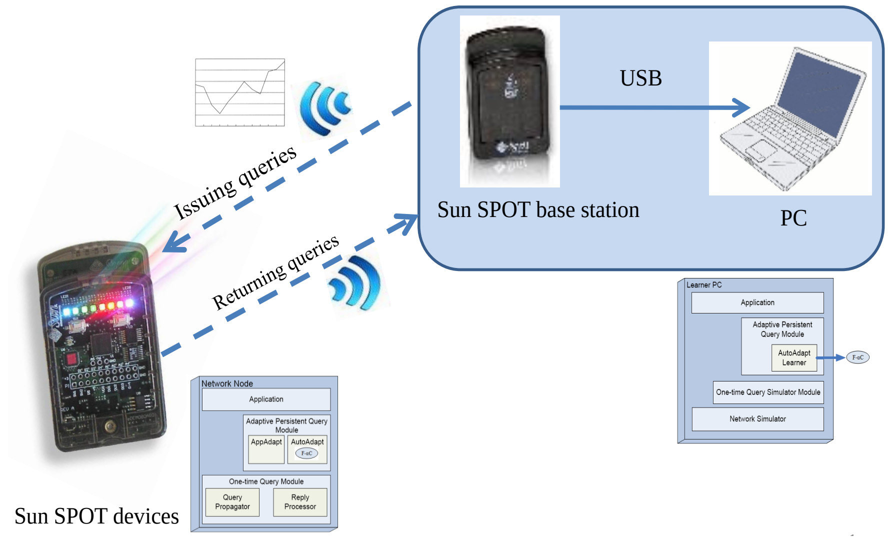


Figure 6.7: PAQ Middleware Architecture on Java Sun SPOT

6.3 Summary

The purpose of this extended PAQ middleware is to simplify the software development tasks and help developers to build pervasive computing applications which entail persistent queries for monitoring in dynamic environments. In order to achieve this goal, PAQ introduces several programming abstractions for persistent query processing and persistent query adaptation. Although persistent query adaptation is very helpful to adjust inquiry strategies to match evolving changes in networks, the insightful knowledge about the relationships between the the resulting quality of query processing, the overhead cost of query execution, and the nature of the operational environment is required for developers to generate suitable adaptation strategies.

In order to address this limitation, we introduce automation adaptation component for PAQ middleware. The component contains a common programming abstract (e.g., interface) which can be used for the implementations of machine learning algorithms. Therefore, the automation adaptation component can provide the services for monitoring the dynamic changes in operational environment and automatically determining $when$ and how to adjust the inquiry strategy during query executions without requiring application developers to have knowledge of the relationship between inner workings of continuous query execution and dynamic environments. Through these features, the middleware can provide an easy-to-use API for application developers and simplify the complex implementations by reducing about 70% of lines of code.

CHAPTER 7: CONCLUSIONS

With the widespread availability and adoption of small wireless devices, a wild range of computing applications have been emerged. In many scenarios, applications are designed to take advantage of information and services within an opportunistically formed wireless network of mobile devices. Those applications can gather related information by issuing queries, which specify how to propagate the inquiry to mobile nodes in the network and how to forward the intermediate results back to the query issuer.

Most of query strategies fall into two categories: one-time snapshot queries and continuous queries. The snapshot queries send out query through the network and retrieve information from corresponding nodes that satisfy the application query criteria at a given time. Through one-time query, the query issuer will have the snapshot information of network at a particular time. This kind of queries is suitable for a wide range of applications. For example, when a driver wants to know which gas station provides the best prices, a snapshot query can be issued over the network and relevant data will be returned back to the driver. However, in some cases, one-time query is not good enough. For example, the drivers may want to continuously monitor the nearby vehicles and surrounding environmental conditions for safety. In this case, a continuous query is needed, which provides a continuously updated view of relevant aspects in a dynamic network.

According to various forwarding and responding functions, different inquiry strategies have different quality of query results and cost of query processing. Therefore, we need to determine suitable inquiry strategy in order to satisfy the applications' requirements. One simple approach is determining the inquiry strategy based on applications' requirements firstly. Then the selected inquiry strategy is used for the whole query processing execution.

This approach should perform well when the requirements and environmental conditions are static. However, in MANET, the networking is dynamic due to evolving network topology and changing operational environment. Hence, the pre-selected inquiry strategy may not be suitable for current networking conditions.

In order to address these challenges, we present automatic adaptation strategies for query processing. In chapter 4, we introduce regression model which learns the relationship between changing conditions, quality of query results, and cost of query executions. Then, the regression model can be used to estimate the query quality or overhead cost when the environmental conditions change. Therefore, the continuous queries can be adjusted to match the evolving changes. We propose another adaptation strategy in chapter 5 to support reasoning networking changes and make adaptive query decision for optimal tradeoff between query quality and execution cost in dynamic environments. The simulation results demonstrate that the proposed adaptation strategies can monitor the dynamics and meet the applications' requirements or achieve optimal tradeoff between quality and cost. Furthermore, a middleware component is provided for simplifying the development tasks of pervasive computing applications.

REFERENCES

[1] R. Avnur and J. M. Hellerstein, “Eddies: Continuously adaptive query processing,” in *ACM International Conference on Management of Data (SIGMOD)*, pp. 261–272, Dallas, TX, May 2000.

[2] S. Babu and J. Widom, “StreaMon: An Adaptive Engine for Stream Query Processing,” in *ACM International Conference on Management of Data (SIGMOD)*, pp. 931–932, June 2004.

[3] D. Barbar, “Mobile Computing and Databases: a Survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, pp. 108–117, 1999.

[4] A. Barto, S. Bradtke, and S. Singh, “Learning to Act using Real-Time Dynamic Programming,” in *Artificial Intelligence*, vol. 72, pp. 81–138, 1995.

[5] M. Bauer, O. Buchtala, T. Horeis, R. Kern, B. Sick, R. Wagner, “Mobile Computing and Databases: a Survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, pp. 108–117, 1999.

[6] R. Bellman, “A Markovian Decision Process,” *Journal of Mathematics and Mechanics*, vol. 6, 1957.

[7] P.A. Bernstein, N. Goodman, E. Wong, C.L. Reeve, and J. Rothnie, “Query Processing in a System for Distributed Databases,” *ACM Transactions on Database Systems*, vol. 6, no. 4, pp. 602–625, December 1981.

[8] A. A. Bhorkar, M. Naghshvar, T. Javidi, and B. D. Rao, “Technical data mining with evolutionary radial basis function classifiers,” *Applied Soft Computing*, vol. 9, no. 2, 2009.

[9] C. Bizer and R. Cyganiak, “Quality-driven information filtering using the WIQA policy framework,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 1, pp. 1–10, January 2009.

[10] P. Bonnet, J. Gehrke, and P. Seshadr, “Towards Sensor Database Systems,” in *Proceedings of 2nd International Conference on Mobile Data Management*, pp. 3–14, 2001.

[11] J. A. Boyan and M. L. Littman, “Packet routing in dynamically changing networks: A reinforcement learning approach,” in *Advances in Neural Information Processing Systems*, vol. 6, pp. 671-678, 1994.

[12] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, “A performance comparison of multi-hop wireless ad hoc network routing protocols,” in *Mobile Computing and Networking*, pp. 85–97, 1998.

[13] O. Buchtala, M. Klimek, and B. Sick, "Evolutionary Optimization of Radial Basis Function Classifiers for Data Mining Applications," *IEEE Transactions on Systems, MAN, and Cybernetics*, vol. 35, no. 5, 2005.

[14] L. Capra, G. S. Blair, C. Mascolo, W. Emmerich, and P. Grace, "Exploiting reflection in mobile computing middleware," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 34–44, October 2002.

[15] G. Di Caro and M. Dorigo, "Antnet: Distributed stigmergetic control for communications networks," in *Journal of AI Research*, vol. 9, pp. 317365, 1998.

[16] G. Di Caro, F. Ducatelle, and L. Gambardella, "AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks," in *Eur. Trans. on Telecommunications*, vol. 16, pp. 443455, 2005.

[17] A. B. Cavalcante and Monika Grajzer, "Fault Propagation Model for Ad Hoc Networks," *IEEE International Conference on Communications (ICC)*, pp. 1–5, June 2011.

[18] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2003.

[19] A. Chan and S.-N. Chuang, "MobiPADS: A reflective middleware for context-aware mobile computing," *IEEE Transactions on Software Engineering*, vol. 29, no. 12, pp. 1072–1085, December 2003.

[20] S. Chatterjee and A. S. Hadi, "Regression Analysis by Example," *John Wiley and Sons*, Fourth Edifion, 2006.

[21] I. Chlamtac, M. Conti, and J. J.-N. Liu, "Mobile ad hoc networking: imperatives and challenges," *Ad Hoc Networks*, vol. 1, no. 1, pp. 13–64, July 2003.

[22] Y. Chen, Q. Zhu, and N. Wang, "Query processing with quality control in the World Wide Web," *World Wide Web*, vol. 1, no. 4, pp. 241–255, 1998.

[23] G. Chin Jr., S. Choudhury, L. Kangas, S. McFarlane, and A. Marquez, "Fault Detection in Distributed Climate Sensor Networks Using Dynamic Bayesian Networks," in *Proceedings of the IEEE Sixth International Conference on e-Science*, Washington, DC, 2010.

[24] A. Deshpande, C. Guestrin, S. Madden, J. Hellersetin, and W. Hong, "Model-driven data acquisition in sensor networks," in *Proceedings of the Thirtieth international conference on Very large data bases*, vol. 30, 2004.

[25] A. Deshpande, Z. Ives, and Vijayshankar Raman, "Adaptive query processing," *Foundations and Trends in Databases*, vol. 1, no. 1, pp. 1–140, 2007.

[26] M. Dorigo and T. Stuetzle, "Ant Colony Optimization," *MIT Press*, 2004.

[27] A. Förster, "*Battery Module 2.0 for OMNeT++ and Mobility Framework*," Web Page. http://www.inf.unisi.ch/postdoc/foerster/downloads.html.

[28] A. Förster, "*Machine Learning Techniques Applied to Wireless Ad-Hoc Networks: Guide and Survey*," In *Proceedings of the third international conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2007.

[29] A. Förster and A. L. Murphy, "*CLIQUE: Role-Free Clustering with Q-Learning for Wireless Sensor Networks*," in Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems (ICDCS), pp. 441-449, 2009.

[30] M. Frodigh, P. Johansson, and P. Larsson, "Wireless ad hoc networking: the art of networking without a network," *Ericsson Review*, no. 4, pp. 248–263, 2000.

[31] C. Furlanello, D. Giuliani, E. Trentin, and D. Falavigna, "Applications of generalized radial basis functions in speakernormalization and identification," IEEE International Symposium on Circuits and Systems (ISCAS), 1995.

[32] D. Huang, "Application of generalized radial basis function networks to recognition of radar targets," International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI) Volume: 13, Issue: 6, pp. 945-962, 1999.

[33] H. Garcia-Molina, J. Ullman, and J. Widom, "Database Systems: The Complete Book," Upper Saddle River, NJ, Prentice-Hall, 2002.

[34] J. R. Groff, P. N. Weinberg, and L. Wald, "SQL: The Complete Reference," 2nd edition Berkeley, CA, McGraw-Hill/Osborne, 2002.

[35] S. Haykin, "Neural NetworksA Comprehensive Foundation," *New York: Macmillan*, 1994.

[36] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "VigilNet: An Integrated Sensor Network System for Energy Efficient Surveillance," *ACM Transactions on Sensor Networks*, vol. 2, pp. 1–38, 2006.

[37] "*IETF Working Group: Mobile Adhoc Networks (manet)*," Web Page. http://www.ietf.org/html.charters/manet-charter.html.

[38] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 2–16, 2003.

[39] M. Jarke, J. Koch, and J. W. Schmidt, "Introduction to Query Processing," in *Proceedings of Query Processing in Database Systems*, pp. 3–28, 1985.

[40] C. Julien, J. Payton, and G.-C. Roman, "Adaptive strategies for persistent queries in dynamic environments," Technical Report TR-UTEDGE-2007-013, The Center for Excellence in Distributed Global Environments, The University of Texas at Austin, 2007.

[41] C. Julien, V. Rajamani, J. Payton, and G.-C. Roman, "Fidelity-Based Continuous Query Introspection and Adaptation," in *Proceedings of the Third International Workshop on Information Quality and Quality of Service for Pervasive Computing (IQ2S)*, March 2011.

[42] L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237285, 1996.

[43] I. Kassabalidis, E. M. A. Sharkawi, R. J. Marks, P. Arabshahi, and A. A. Gray, "Swarm intelligence for routing in communication networks," in *Proceedings of the IEEE Global Tel. Conf. (GLOBECOM)*, IEEE Press, 2001.

[44] J. Kennedy and R. Eberhart, "Swarm Intelligence," Morgan Kaufmann, 2001.

[45] A. Ko, H. Y. K. Lau, "Robot Assisted Emergency Search and Rescue System With a Wireless Sensor Network," *International Journal of Advanced Science and Technology*, vol. 3, February 2009.

[46] S. Koenig, "Agent-centered search," *AI Magazine*, vol. 22, no. 4, pp. 109131, 2001.

[47] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189211, 1990.

[48] D. Kossmann, "The state of the art in distributed query processing," *International Conference on Military Communications (MILCOM)*, pp. 1–7, Nov. 2008.

[49] Y. Lacharit, D. Q. Nguyen, M. Wang, and L. Lamont, "A Trust-based Security Architecture for Tactical MANETs," *ACM Computing Surveys*, vol. 32, no. 4, pp. 418–469, Dec. 2000.

[50] D. E.W. Laidler, "The Demand for Money: Theories, Evidence, and Problems," 4 edition *Addison Wesley*, 1997.

[51] S. Li, A. Zhan, X. Wu, and G. Chen, "ERN: Emergence Rescue Navigation with Wireless Sensor Networks," in *15th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 361–368, 2009.

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

[53] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks," in *5th Annual Symposium on Operating System Design and Implementation (OSDI)*, pp. 491–502, 2002.

[54] S. Madden, M. Shah, J. Hellerstein, and V. Raman, "Continuously adaptive continuous queries over streams," in *Proceedings of ACM SIGMOD*, pp. 491–502, 2002.

[55] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proceedings of the 2003 ACM International conference on Management of Data (SIGMOD)*, pp. 491–502, 2003.

[56] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pp. 88–97, Atlanta, 2002.

[57] J. Melton and A. R. Simon, "Understanding The New SQL: A Complete Guide," Morgan Kaufmann, 1 edition, 1993.

[58] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: A Tiny AGgregation service for ad-hoc sensor networks," in *ACM SIGOPS 36(SI)*, pp. 131–146, 2002.

[59] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "The Design of an Acquisitional Query Processor For Sensor Networks," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 491-502, 2003.

[60] T. M. Mitchell, "*Machine learning*," McGraw Hil, New York, 1997.

[61] R. E. Neapolitan, "*Bayesian Networks*," Prentice Hall, 2003.

[62] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou, "PeerDB: A P2P-based System for Distributed Data Sharing," in *Proceedings of 19th International Conference on Data Engineering*, pp. 633–644, March 2003.

[63] X. Nguyen, M. I. Jordan, and B. Sinopoli, "A kernel-based learning approach to ad hoc sensor network localization," *IACM Transactions on Sensor Networks*, vol. 1, no. 1, pp. 134–152, Aug 2005.

[64] C. Olston, J. Jiang, and J. Widom, "Adaptive Filters for Continuous Queries over Distributed Data Streams," in *Proceedings of the 2003 ACM SIGMOD*, 2003.

[65] M. T. zsu and P. Valduriez, "Principles of Distributed Database Systems," *Prentice-Hall*, 2nd edition, 1999.

[66] J. Payton, C. Julien, and G.-C. Roman, "Automatic consistency assessment for query results in dynamic environments," in *Proceedings of SIGSOFT symposium on The foundations of software engineering*, pp. 245–254, September 2007.

[67] J. Payton, R. Souvenir, and D. Liu, "*An Architecture to Support Learning-based Adaptation of Persistent Queries in Mobile Environments*," Proceedings of the 2nd Workshop on Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS'09), June 2009.

[68] J. Payton, C. Julien, G.-C. Roman, and V. Rajamani, "Semantic self-assessment of query results in dynamic environments," in *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 19, no.4, April 2010.

[69] T. Poggio and F. Girosi, "*Networks for approximation and learning*," Proceedings of the IEEE 78:113-125, 1990.

[70] S. Papadimitriou, S. Mavroudi, L. Vladutu, and A. Bezerianos, "Generalized Radial Basis Function Networks Trained with Instance Based Learning for Data Mining of Symbolic Data," Applied Intelligence, Volume 16, Issue 3, pp. 223-234, 2002.

[71] M. J. D. Powell, "Radial basis functions for multivariable interpolation: a review," In J. C. Mason and M. G. Cox, editors, Algorithms for Approximation. Clarendon Press, Oxford, 1987.

[72] N. Preguia, C. Baquero, J. L. Martins, F. Moura, H. Domingos, R. Oliveira, J. O. Pereira, and S. Duarte, "Mobile Transaction Management in Mobisnap," in *the Proceedings of ADBIS-DASFAA*, pp. 379–386, 2000.

[73] M. L. Puterman, "Markov Decision Processes," *Wiley*, 1994.

[74] Saeed K. Rahimi and Frank S. Haug "Distributed Database Management Systems: A Practical Approach," *Wiley-IEEE Computer Society Pr*, 1 edition, August 2010.

[75] V. Rajamani and C. Julien, "Adaptive data quality for persistent queries in sensor networks," *Technical Report TRUTEDGE-2008-001*, 2008.

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

[77] V. Rajamani, C. Julien, J. Payton, and G.-C. Roman, "PAQ: persistent adaptive query middleware for dynamic environments," in *the Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, December 2009.

[78] T. S. Rappaport, Wireless Communications Principles and Practice, *Prentice Hall PTR*, 2001.

[79] Q. Ren and Q. Liang, "Energy and quality aware query processing in wireless sensor database systems," *Information Sciences: an International Journal*, Volume 177, Issue 10, pp. 2188-2205, May 2007.

[80] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Transactions on Speech and Audio Processing*, Volume 3, Issue 1, pp. 72-83, Jan 1995.

[81] J. H. Schiller, Mobile Communications, *Addison-Wesley Professional*, 2003.

[82] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, "Antbased load balancing in telecommunications networks," *Adaptive Behavior*, no. 2, pp. 169207, 1996.

[83] C. Schroth, R. Eigner, S. Eichler, and M.Strassberger, "A framework for network utility maximization in VANETs," in *Proceedings of the 3rd international workshop on Vehicular ad hoc networks*, 2006.

[84] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler, "Design and Implementation of a Sensor Network System for Vehicle Tracking and Autonomous Interception," in *the Proceedings of the Second European Workshop on Wireless Sensor Networks*, pp. 93–107, July 2005.

[85] A. P. Sistla, O. Wolfson, and Y. Huang, "Minimization of communication cost through caching in mobile environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no.4, pp. 378–390, 1998.

[86] P. Stone, "Tpot-RL applied to network routing," in *Proceedings of the 17th International Conference on Machine Learning*, San Francisco, CA, 2000.

[87] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, March 1998.

[88] "*SUN SPOT World*," Web Page. http://http://www.sunspotworld.com/.

[89] P. Szczurek, B. Xu, J. Lin, and O. Wolfson, "Spatio-temporal Information Ranking in VANET Applications," *Prentice Hall software series*, 2010.

[90] D. Terry, A. J. Demers, K. Petersen, M.J. Spreitzer, M.M. Theimer, and B.B. Welch, "Session Guarantees for Weakly Consistent Replicated Data," in *Proceedings of Conference Parallel and Distributed Computing*, Austin, Texas, Oct. 1994.

[91] C.-K. Toh, *Ad Hoc Mobile Wireless Networks: Protocols and Systems*, Prentice Hall PTR, 1 edition, Dec 2001.

[92] A. Vargas, *OMNeT++*, Web Page. http://www.omentpp.org.

[93] S. Viglas, J. Naughton, and J. Burger, "Maximizing the output rate of multi-join queries over streaming information sources," in *Proceedings of 2003 International Conference on Very Large Databases*, September 2009.

[94] P. Wang and T. Wang, "Adaptive routing for sensor networks using reinforcement learning," in *Proceedings of the 6th IEEE International Conference on Computer and Information Technology (CIT)*, IEEE Computer Society, 2006.

[95] C. J. Watkins, "Learning from Delayed Rewards," *Ph.D. thesis*, Cambridge University, 1989.

[96] C. Watkins and P. Dayan, "Technical Note: Q-Learning," *Machine Learning*, vol. 8, no. 3, pp. 279-292, May 1992.

[97] Y. Wu, P. A. Chou, and S.-Y. Kung, “Information exchange in wireless networks with network coding and physical-layer broadcast,” in *the Proceedings of 39th Annual Conference on Information Sciences and Systems (CISS)*, 2005.

[98] H. Wu, Q. Luo, J. Li, and A. Labrinidis, “Quality aware query scheduling in wireless sensor networks,” in *the Proceedings of the Sixth International Workshop on Data Management for Sensor Networks*, Lyon, France, 2009.

[99] Y. Yao and J. Gehrke, “The Cougar approach to in-network query processing in sensor networks,” *ACM SIGMOD Record*, vol. 31, no. 2, pp. 9–18, 2002.