Report Number: WUCSE-2005-49

2005-10-31

MobiQuery: A Spatiotemporal Query Service for Mobile Users in Sensor Networks

Guoliang Xing, Sangeeta Bhattacharya, Chenyang Lu, Octav Chipara, Chien-Liang Fok, and Gruia-Catalin Roman

This paper presents MobiQuery, a spatiotemporal query service that allows mobile users to periodically collect sensor data from the physical environment through wireless sensor networks. A salient feature of \MQ is that it can meet stringent spatiotemporal performance constraints, including query latency, data freshness, and changing areas of interest due to user mobility. We present three just-in-time prefetching protocols that enable MobiQuery to achieve desired spatiotemporal performance despite low node duty cycles, while significantly reducing communication overhead. We validate our approach through both theoretical analysis and extensive simulations under realistic settings including varying user movement patterns and location errors.

... Read complete abstract on page 2.

MobiQuery: A Spatiotemporal Query Service for Mobile Users in Sensor Networks

Guoliang Xing, Sangeeta Bhattacharya, Chenyang Lu, Octav Chipara, Chien-Liang Fok, and Gruia-Catalin Roman

Complete Abstract:

This paper presents MobiQuery, a spatiotemporal query service that allows mobile users to periodically collect sensor data from the physical environment through wireless sensor networks. A salient feature of \MQ is that it can meet stringent spatiotemporal performance constraints, including query latency, data freshness, and changing areas of interest due to user mobility. We present three just-in-time prefetching protocols that enable MobiQuery to achieve desired spatiotemporal performance despite low node duty cycles, while significantly reducing communication overhead. We validate our approach through both theoretical analysis and extensive simulations under realistic settings including varying user movement patterns and location errors.

2005-49

MobiQuery: A Spatiotemporal Query Service for Mobile Users in Sensor Networks

Authors: Xing, Guoliang; Bhattacharya, Sangeeta; Lu, Chenyang; Chipara, Octav; Fok, Chien-Liang; Roman, Gruia-Catalin

October 31, 2005

Abstract: This paper presents MobiQuery, a spatiotemporal query service that allows mobile users to periodically collect sensor data from the physical environment through wireless sensor networks. A salient feature of \MQ is that it can meet stringent spatiotemporal performance constraints, including query latency, data freshness, and changing areas of interest due to user mobility. We present three just-in-time prefetching protocols that enable MobiQuery to achieve desired spatiotemporal performance despite low node duty cycles, while significantly reducing communication overhead. We validate our approach through both theoretical analysis and extensive simulations under realistic settings including varying user movement patterns and location errors.

MobiQuery: A Spatiotemporal Query Service for Mobile Users in Sensor Networks[1]

Guoliang Xing; Sangeeta Bhattacharya; Chenyang Lu
Octav Chipara; Chien-Liang Fok, Gruia-Catalin Roman

This paper presents MobiQuery, a spatiotemporal query service that allows mobile users to periodically collect sensor data from the physical environment through wireless sensor networks. A salient feature of MobiQuery is that it can meet stringent spatiotemporal performance constraints, including query latency, data freshness, and changing areas of interest due to user mobility. We present three just-in-time prefetching protocols that enable MobiQuery to achieve desired spatiotemporal performance despite low node duty cycles, while significantly reducing communication overhead. We validate our approach through both theoretical analysis and extensive simulations under realistic settings including varying user movement patterns and location errors.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Sensor Networks, Spatiotemporal Query, Mobile User

1. INTRODUCTION

As large-scale wireless sensor networks make it possible to monitor the physical environment at unprecedented spatial and temporal granularities, a key research challenge is to develop data services that deliver information to mobile users at the right time and right location. In this paper, we propose a new data service for mobile users in sensor networks called *spatiotemporal query*. In contrast to existing data services that generally assume fixed areas of interest [Intanagonwiwat et al. 2000; Madden et al. 2002; Ye et al. 2002; Kim et al. 2003; Yao and Gehrke 2002], spatiotemporal query is motivated by a class of mission-critical applications in which mobile users need to continuously gather real-time information from their vicinity. For example, a fireman fighting a wild fire may request a periodic update of a temperature map within one mile around his current location to remain alert to the surrounding fire conditions. As the fireman moves, the query area changes accordingly. As another example, a robot in a search and rescue operation needs to continuously query surrounding sensors for information about nearby dangers and survivors as it moves in a dynamic environment. Based on the query results, the robot can locate survivors and find the best rescue route through motion planning [Li et al. 2003; Alankus et al. 2005]. Queries in the above application class are subject to the following common constraints.

—*Spatial constraints:* Only the nodes inside the query area corresponding to the user's current position should contribute to the query result. Involving more nodes wastes precious energy without improving the quality of service. Furthermore, it is desirable

[1]Guoliang Xing is with City University of Hong Kong. The other authors are with Washington University in St. Louis. This work is funded in part by the NSF under an ITR grant # CCR-0325529 and the ONR under MURI research contract # N00014-02-1-0715.

to aggregate data from enough nodes in the current query area to improve the fidelity of the query result.

—*Temporal constraints:* A query is also subject to temporal constraints in terms of *query deadlines* and *data freshness*. A query result must be delivered to the user by the end of each query period. Moreover, sensor data must be *fresh*. The freshness of a sensor datum is defined by its maximum validity interval after it is read from the sensor.

Meeting these constraints is crucial for the mission-critical applications that rely on surrounding sensor data to maintain spatiotemporal context awareness. For example, a fireman may be endangered by a quickly evolving wild fire if the query results are aggregated from old sensor readings, are delivered too late, are aggregated from sources at wrong locations, or if too few nodes in the current query area contribute to the results.

Meeting all the spatiotemporal constraints is especially challenging in wireless sensor networks due to their severe power and resource constraints. Although the temporal constraints require the nodes to respond to a query in a timely fashion, the low node duty cycles forced by the power constraint can significantly increase the communication delay. Let's consider the following example to better understand the fundamental tension between the constraint on communication delay and the requirement of extended network lifetime[2]. For a MICA2 mote to remain operational for 450 days, the duty cycle needs to be lower than 1% [Polastre et al. 2004]. The maximum communication delay between nodes operating in a duty cycle depends on the length of active window. The active window of S-MAC [Ye et al. 2004] is 150 ms in its MICA implementation. Consequently, the sleep period is at least $15s$ and the communication with sleeping nodes is subject to a maximum delay of $15 - 0.15 = 14.85s$. Such a long latency is intolerable to mobile users that must maintain real-time context awareness in response to rapidly changing environments. In addition, a spatiotemporal query service faces the challenge of reducing communication overhead caused by the continuous queries from mobile users.

The key contributions of this paper are as follows. (1) We design a new spatiotemporal query service called *MobiQuery* that allows a mobile user to periodically query a surrounding area under spatiotemporal constraints (see Section 4). (2) We propose a novel just-in-time prefetching scheme that can meet stringent spatiotemporal constraints despite the long communication delays caused by low node duty cycles. Furthermore, our analysis shows that an advantage of just-in-time prefetching is that it significantly reduces communication overhead caused by continuous queries from mobile users (see Section 6). (3) We present three just-in-time prefetching protocols, Directional Tree Creation (DTC), Directional Tree Maintenance (DTM) and Omni-directional Tree Creation (OTC) to support MobiQuery. DTC can achieve satisfactory spatiotemporal performance under extremely low node duty cycles. DTM can further reduce communication overhead when query areas overlap while OTC is particularly robust against unpredictable user movement patterns (see Section 5). (4) We validate the design of MobiQuery through theoretical analysis and extensive simulations under realistic settings. For example, our simulation results show that both DTM and OTC successfully deliver over 85% of the query results to a user querying its surrounding area with 150m radius twice every second, even when the wake-up delay is as high as about $15s$, the user changes its motion pattern about every 60s, and the location

[2]While the example is based on low-power MACs (e.g., S-MAC [Ye et al. 2004]) on MICA2 motes, MobiQuery is designed to work with a broad class of MACs that employ periodic sleep schedules like 802.11 PSM.

error is as large as 10m. (see Section 7).

The rest of the paper is organized as follows. We present related work in Section 2 followed by problem formulation in Section 3. The architecture of MobiQuery is discussed in Section 4. We then present the design of the three prefetching protocols in Section 5, followed by a theoretical analysis of the protocols in Section 6. Simulation results are discussed in Section 7, followed by conclusions in Section 8.

2. RELATED WORK

Several data query services have been developed for sensor networks [Intanagonwiwat et al. 2003; Madden et al. 2002; Ye et al. 2002; Kim et al. 2003]. Directed Diffusion [Intanagonwiwat et al. 2003] is a data-centric communication paradigm that allows for in-network data aggregation. TinyDB [Madden et al. 2002] is an energy-efficient query service for sensor networks. However, unlike MobiQuery, both TinyDB and Directed Diffusion assume fixed query areas and are not designed to handle moving users or query areas. It has been shown in [Kim et al. 2003] that Directed Diffusion is unsuitable for mobile users. TTDD [Ye et al. 2002] and SEAD [Kim et al. 2003] are data services that allow mobile users to collect data from fixed areas. TTDD builds a virtual grid to deliver the data to mobile sinks. SEAD maintains a routing tree for mobile users and uses data caches to balance latency and energy consumption. Unlike these protocols, MobiQuery is designed to query an area that moves with the user. Dealing with both user mobility and time-varying data sources introduces new challenges to data services.

Earlier projects on tracking in WSNs also deal with the mobility of objects. The solutions in [Li et al. 2002; Zhao et al. 2002; Blum et al. 2003; Aslam et al. 2003] do not employ wake-up protocols to handle sleeping nodes. As a result, they may suffer from poor tracking performance when nodes sleep most of the time and cannot contribute to the tracking process. For example, He et al. [He et al. 2004] observed that the sleeping schedule severely affects the tracking performance of their surveillance system when an object moves fast.

Several recent projects have studied wake-up protocols for object tracking [Zhang and Cao 2004; Pattem et al. 2003; Gui and Mohapatra 2004]. Gui et al. propose a wake-up mechanism to achieve specified quality of surveillance for moving objects [Gui and Mohapatra 2004]. A selective sensor activation scheme is proposed in [Pattem et al. 2003] to achieve tradeoffs between energy and quality of tracking. DCTC [Zhang and Cao 2004] is an object tracking protocol that maintains a tree around the predicted route of a moving target. The above protocols, however, were not designed for spatiotemporal queries. More importantly, they were best-effort solutions that are not designed to meet spatiotemporal constraints. As a result, they may not be able to maintain the desired tracking performance in the face of fast objects and/or low network duty cycles. The ability to meet stringent spatiotemporal constraints differentiate our protocols from the above work.

Mobicast [Huang et al. 2003] bears some resemblance to this work. Mobicast is a spatiotemporal multicast protocol designed for disseminating data to a changing area just in time. Mobicast deals only with data *dissemination*, while MobiQuery *collects* data from a moving area in addition to dissemination. Moreover, Mobicast does not consider node duty cycles while MobiQuery uses a novel prefetching scheme to overcome the long communication delays due to node duty cycles.

3. PROBLEM FORMULATION

In this section we introduce the spatiotemporal query model, formalize its spatiotemporal requirements and discuss the assumptions underlying the design of MobiQuery. A user issues a spatiotemporal query to periodically collect data from sensors in surrounding areas while moving through a sensor field. A spatiotemporal query Q has six parameters: $(\theta,\ F,\ A(P_u(t)),\ T_p,\ T_f,\ T_{life})$, which are provided by the user based on application requirements. θ is the type of sensor data being queried. F is an aggregation function that is applied to the results inside the network; in-network aggregation is a well-investigated technique utilized by existing data services [Madden et al. 2002] to reduce bandwidth consumption. $A(P_u(t))$ is a function defining the query area relative to the user's position, $P_u(t)$, at time t. For simplicity, we assume $A(P_u(t))$ is a circle with radius R_q centered at $P_u(t)$ in the rest of the paper, although our design can be easily extended to other types of query areas. T_p is the query period. The user expects to receive a new result by the end of each query period. T_f specifies the data freshness constraint, i.e., a query result is acceptable only if it is aggregated from sensor readings no more than T_f seconds old. T_{life} is the lifetime of the query. As a query usually spans multiple query periods, a query may collect query results from multiple query areas in its lifetime. In the rest of this paper, a *query instance* refers to the process of collecting a query result from a query area within one period, and so a *query* may comprise multiple query instances.

Before formalizing the requirements of a spatiotemporal query, we first define the following notation.

—V denotes the set of nodes in the network. P_i denotes the location of node i, $i \in V$.

—φ_k denotes the result that is received by the user from the network for the k^{th} query instance.

—$N(\varphi_k)$ is the set of nodes whose sensor readings are contained in φ_k (after being aggregated by function F).

—$t_s(\varphi_k)$ represents the earliest time instance that a reading aggregated in φ_k is taken by a node in $N(\varphi_k)$.

—$t_r(\varphi_k)$ is the time instance that φ_k is received by the user.

The objective of our problem is to satisfy the following constraints for a spatiotemporal query, Q$(\theta,\ F,\ A(P_u(t)),\ T_p,\ T_f,\ T_{life})$, issued by the user at time instance t_0:

$$\forall k, 1 \le k \le \lfloor \frac{T_{life}}{T_p} \rfloor,\ t_0 + (k-1)T_p \le t_r(\varphi_k) \le t_0 + kT_p \tag{1}$$

$$\forall k, 1 \le k \le \lfloor \frac{T_{life}}{T_p} \rfloor,\ t_0 + kT_p - T_f \le t_s(\varphi_k) \le t_0 + kT_p \tag{2}$$

$$\forall k, 1 \le k \le \lfloor \frac{T_{life}}{T_p} \rfloor,\ \forall i \in N(\varphi_k),\ \ P_i \in A(P_u(t_0 + (k-1)T_p)) \tag{3}$$

Eqn. (1) and (2) specify the temporal constraints of the query result. That is, for the k^{th} query instance, a query result φ_k is expected to be received at or before deadline $t_0 + k \cdot T_p$, and the data in the result is at most T_f old. Eqn. (3) specifies the spatial constraint of the query result. That is, for the k^{th} query period, all and only nodes within the query area (defined according to the user's location at the beginning of the period) should contribute to the query result.

To solve this problem, we make some assumptions about the underlying sensor network: 1) all nodes have synchronized clocks, 2) each node knows its own location through a localization service, and 3) to prolong the network lifetime, the network runs a power management protocol that selects a small subset of nodes to keep *active* while the remaining nodes operate in a synchronized duty cycle. The active nodes form a *backbone* network that allows the communication delay between any two nodes to remain in the order of one duty cycle. A number of existing power management protocols [Chen et al. 2001; Xing et al. 2005; Xu et al. 2001] maintain such backbones.

4. SYSTEM ARCHITECTURE

MobiQuery is spread across two types of devices: a proxy and nodes of the sensor network. A proxy is a mobile device (such as a PDA or laptop) carried by the user as it moves through a sensor field. The *query gateway* on the proxy serves as the interface between the proxy and the nodes in the network. Three components of MobiQuery reside on the nodes: *prefetching*, *query dissemination* and *data collection*. In addition, MobiQuery also works with power management protocols on nodes and an optional motion predictor on the proxy.

The query gateway issues queries to the network for the user. If a motion profile is available, the query gateway appends it to the query, before issuing the query to the network. Note that a user only needs to submit a query once at the beginning of the query. MobiQuery will periodically deliver new query results to the user until the query expires. In the network, MobiQuery handles a spatiotemporal query, by first alerting nodes in future query areas about the query, using a technique called *prefetching*. MobiQuery then uses the query dissemination and data collection components to establish a query tree that aggregates the sensor readings from the nodes in a query area and delivers them to the user, when the user reaches the query area.

4.1 Motion Predictor

The motion predictor generates a motion profile that specifies the user's future movement. MobiQuery uses the motion profile to predict future query areas. However, this component is optional, and MobiQuery can operate without a motion profile. Note that this component resides on the proxy carried by the user, which typically has more processing power than the network nodes.

4.1.1 *Generation of Motion Profiles.* We discuss two ways of generating motion profiles. First, a motion profile can be generated based on the recent movement history of the user obtained from a GPS in the proxy or a location service [Hightower and Borriello 2001]. Second, autonomous robots usually generate motion profiles based on motion planning [Latombe 1991] and control their future movement accordingly. In such cases, the motion planner can provide the motion profiles to MobiQuery without incurring additional overhead.

As a simple example, a motion profile P includes a velocity $\overrightarrow{v}$, assuming the user moves at $\overrightarrow{v}$ in the future. $\overrightarrow{v}$ can be estimated based on two previous user positions (p_1, t_1) and (p_2, t_2), where p_i $(i = 1, 2)$ is the user position at time t_i. Let $\delta = t_2 - t_1$. Then $\overrightarrow{v}$ can be estimated as $\overrightarrow{v} = \frac{\overrightarrow{p_1 p_2}}{\delta}$, where δ represents the sampling period of user positions. δ affects the accuracy of the motion prediction and is a system parameter of the motion predictor. Intuitively, when there are errors in location readings (e.g., p_1 and p_2), a small δ may result in high prediction errors. The proxy periodically monitors the user's position and issues a new motion profile whenever the user diverges from the motion profile by a threshold.

4.1.2 *Motion Profile Model.* A motion profile P in MobiQuery is defined by the 4-tuple (p_s, p_e, t_s, t_e) where p_s, p_e are two points in space that denote the user location at times t_s and t_e respectively. The user is thus predicted to travel along a straight line from p_s to p_e within time $[t_s, t_e]$. Each motion profile P is also associated with a third timing parameter t_g, which specifies when P is generated. Let $T_a = t_s - t_g$. T_a represents how early the proxy receives P before P takes effect and is referred to as the *advance time* of P.

When P is generated by a motion planner, T_a is positive since P is always created before the user takes the planned path. In contrast, when P is generated based on movement history, T_a is negative because the motion profile is available only after one sampling period. In this case, part of P ($[t_s + T_a, t_s]$ $(T_a \leq 0)$) has expired by the time the proxy receives P.

MobiQuery uses the history based approach to generate the motion profile. The user position is obtained at every sampling period T_s from GPS. T_s is lower bounded by the delay in acquiring a location reading of the user location. For example, obtaining an initial location reading takes about $8s$ [Papyrus 2004]. The recent experiments on Leadtek GPS with Berkeley motes show that the lag of location readings is about $2 \sim 3s$ when the user walks briskly [Firebug-Project 2004]. MobiQuery monitors the distance between the current user location obtained at each sampling period and the location predicted by the current motion profile, and generates a new motion profile, if the distance is greater than a threshold. Our experiments in Section 7.4 show that MobiQuery can achieve satisfactory performance when working with this simple motion prediction technique. We note that more complex techniques [Aljadhai and Znati 2001] can be used to improve the accuracy of the motion prediction.

4.2 Prefetching, Query Dissemination and Data Collection

Prefetching is a method that alerts nodes in future query areas about the user query so that they can respond to the query in time. It is a key component of MobiQuery and enables MobiQuery to meet the spatiotemporal constraints of a query despite the extremely low node duty cycles. To understand why prefetching is necessary, we consider the following simple example. Assume that the nodes in a network are active for $150ms$ in every 15 seconds and that the user needs to query the nodes in its vicinity every 5 seconds. If the user disseminates the query at the beginning of each query period, due to the sleep schedule, on average only $1/3$ of the nodes can be woken up by the end of the query period. Prefetching solves this problem by alerting the sleeping nodes about the query beforehand, enable them to wake up at the right time to participate in the query. Two possible prefetching schemes are discussed next.

Greedy Prefetching. Nodes in all future query areas can be alerted in an as-soon-as-possible fashion. This approach maximizes the slack time at successive query areas in order to meet query deadlines. However, it can cause high communication overhead per period and excessive network contention as a query may be disseminated in adjacent query areas simultaneously. We formally analyze the communication overhead under greedy prefetching in Section 6.2.

Just-In-Time Prefetching. This scheme alerts nodes just-in-time to respond to the query. It uses a *hold and forward* strategy where the forwarding is paced in order to reduce com-

T_f	freshness requirement of query results	T_{sleep}	sleep period
T_a	advance time of motion profile	T_s	sampling period of motion profile
T_{life}	lifetime of a user query	$T_{collect}$	duration of data collection process
T_p	period of user query	N_a	average number of active nodes per query area
v_{alt}	speed of *alert* message	T_{setup}	delivery delay of *setup* message in a query area
v_{user}	speed of user	R_p	max distance between a pickup point and collector node
R_c	communication range of nodes	T_{delay}	max delivery delay of query results set by the user

Table I. Symbols used in protocol design and analysis.

munication overhead and network contention. When the motion profile is available, it reduces the cost caused by user motion changes by sending a cancel message along the abandoned path to stop the previous prefetching process. However, the cancellation mechanism is not effective for greedy prefetching because the query is distributed to most of the query areas on the path by the time a motion change occurs.

The query dissemination component of MobiQuery is responsible for distributing the query to a query area, and building a routing tree for data collection. In data collection, the nodes perform a measurement and send their data via the routing tree to the user.

5. PROTOCOL DESIGN

In this section, we present three prefetching protocols: Directional Tree Creation (DTC), Directional Tree Maintenance (DTM), and Omni-directional Tree Creation (OTC). Both DTC and DTM calculate the user's path based on the user's motion profile. DTC wakes up nodes ahead of the user and creates a new query tree in each query area along the predicted path, while DTM maintains a single tree that moves with the user. DTM can significantly reduce the communication overhead and network contention when the query areas overlap. OTC on the other hand assumes no knowledge of user motion profile and wakes up nodes in a circular region that covers all possible future query areas ahead of time. Operating without the user's motion profile, OTC is therefore robust to user's motion changes and location errors. Table I lists all the symbols used in protocol design and analysis.

5.1 Directional Tree Creation

DTC wakes up nodes along the predicted user path based on a motion profile. After receiving the query and the motion profile from the proxy, DTC uses the motion profile to predict locations called *pickup points* where the user expects to receive the query result in each query period. An *alert* message containing the query specification and the motion profile is then relayed by the network to each future pickup point using an area anycast [He et al. 2005]. The area anycast delivers the *alert* message to a node within a certain distance, R_p, of the pickup point. To guarantee the delivery of the *alert* message, R_p should be selected based on the density of the sensor network. For instance, when the deployment region has sensing coverage, i.e., every point in the region is within the sensing range of at least one node, R_p can be set to the sensing range. This is because the sensing coverage guarantees that there exists a node within the sensing range of any point. The first node within R_p of the pickup point that receives the *alert* message is called a *collector node*. It is responsible for relaying the *alert* message to the next pickup point, distributing the query to its query area, and aggregating the results in time for delivery to the user when the user reaches the pickup point The process of forwarding the *alert* message along the predicted user path is illustrated in Fig. 1.

Upon receiving the *alert* message, a collector node floods a *setup* message with the

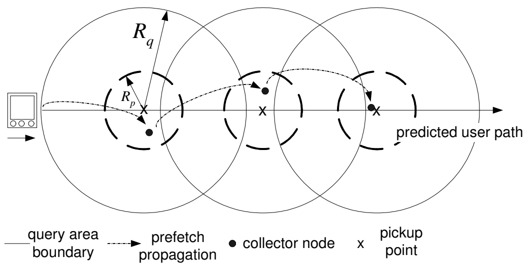


Fig. 1. The process of forwarding an *alert* message along the predicted path.

query specification to all the nodes in its query area. Each active node sets the first node from which it receives the message as its parent [3]. The active nodes store the message and deliver it to the sleeping nodes when they wake up. Upon receiving the message, the sleeping nodes set the active node from which they first receive the message as their parent, and then reconfigure their sleep schedule to wake up at time $kT_p - T_f$ if the message from the collector node is for the k^{th} query period. Note that this is the earliest time any node can perform measurements without violating the freshness constraint. Sleeping nodes are restricted to be leaves of the query tree to allow them to quickly go back to sleep after measuring and transmitting a single sensor reading. The constitutes the query dissemination phase.

In the data collection phase, each leaf node in the k^{th} query area takes a sensor reading at time $kT_p - T_f$, and sends it to its parent. Each parent node collects data reports from its children until its *sub-deadline*. It then performs its own sensor measurement, aggregates its data with the data reports from its children, and sends the aggregated data report to its parent. The sub-deadline of each parent node is chosen to allow data aggregation while meeting the temporal constraints. DTC uses the following heuristic to assign node u's sub-deadline, $d_u(k)$, for the k^{th} query area:

$$d_u(k) = \begin{cases} k \cdot T_p & if\ parent\ is\ user \\ k \cdot T_p - \frac{|up|}{R_p + R_q} \cdot T_f & otherwise \end{cases} \tag{4}$$

where $R_p + R_q$ is the maximum distance between a node in the query area and the collector node that resides within R_p range of the pickup point. $|up|$ is the Euclidean distance between u and the collector node p. Eqn. (4) ensures that the further a node is from the pickup point, the quicker it will timeout and forward the result to its parent, which increases the likelihood of aggregation and timely delivery of query results to the user.

The DTC protocol is shown in Fig. 2. DTC supports both the greedy and just-in-time prefetching schemes discussed in Section 4.2. In the rest of this paper, DTC with greedy and just-in-time prefetching are denoted by DTC-GP and DTC-JIT, respectively. In DTC-GP, each collector node forwards the *alert* message to the next collector node *immediately* after receiving it. In contrast, DTC-JIT holds the message for a certain amount of time before forwarding it. The time to forward the message without violating the query

[3]Several existing techniques may be used to eliminate asymmetric links from the tree [He et al. 2004]. Another potential issue is that the tree may need to be reconfigured when the underlying power management protocol dynamically puts an active node to sleep in order to achieve load balancing [Chen et al. 2001]. To avoid the overhead of tree reconfigurations, parent node stays awake until the end of the current query instance.

```
When alert message is received
(1) Calculate the next pickup point, $P$, using the motion profile.
(2) If in greedy mode, send an anycast alert message to $P$. If in JIT mode, set ForwardTimer
    to fire at time given by (13).
(3) Broadcast a setup message.
(4) Set SendTimer to fire at data collection time given by (4).
When setup message is received
(1) If in query area, set sender as parent. Rebroadcast the setup message if active node.
(2) Set SendTimer to fire at data collection time given by (4).
When ForwardTimer fires
(1) Send an anycast alert message to $P$.
When SendTimer fires
(1) If sleeping node, wake up (turn on the radio).
(2) Send query result to parent.
```

Fig. 2. DTC Protocol.

deadlines, referred to as *prefetch forwarding time*, is a key design parameter of DTC-JIT.

5.1.1 *Prefetch Forwarding Time Analysis.* The prefetch forwarding time should be early enough so that the next query result can be delivered to the user in time, and also late enough to achieve just-in-time prefetching. We first make the following assumptions:

$$T_{collect} \leq T_f \tag{5}$$
$$T_{setup} \leq T_f \tag{6}$$
$$v_{user} < v_{alt} \tag{7}$$

$T_{collect}$ is the duration of the data collection process. $T_{collect} \leq T_f$ is necessary to meet both the freshness and the deadline constraints. Otherwise it is impossible for the network to deliver a query result to the user before the data becomes too old. This condition is enforced by the sub-deadline assignment discussed earlier. Similarly, we assume $T_{setup} \leq T_f$. T_{setup} is the time it takes to create the partial query tree composed of only active nodes in a query area, and is usually shorter than the data collection delay because the tree is set up as soon as possible and does not incur any aggregation delay. Moreover, only active nodes communicate during T_{setup}, which involves fewer hops than in data collection. v_{user} and v_{alt} in (7) denote the speed of the user and the *alert* message, respectively. v_{alt} is defined as the distance between two consecutive collector nodes divided by the communication delay between them. Intuitively, (7) is necessary for the network communication to be able to catch up with the user. This assumption is reasonable because an *alert* message is always forwarded by active nodes, without wake-up delays.

Our goal is to derive $t_{send}(k-1)$ which is the time at which the $(k-1)^{th}$ collector node should forward the *alert* message to the k^{th} collector node such that the deadline of the k^{th} query result ($k \cdot T_p$) is met. We first derive the time (denoted by $t_{recv}(k)$) by which the k^{th} collector node should receive the *alert* message in order to meet the deadline. Upon receiving the *alert* message, the k^{th} collector node needs to set up the query tree and to collect the data before the deadline $k \cdot T_p$. Hence the query deadline is met if the following inequality holds:

$$t_{recv}(k) \leq k \cdot T_p - T_{tree} - T_{collect} \tag{8}$$

where T_{tree} denotes the time taken to create the query tree composed of all nodes in the query area. T_{tree} equals the sum of T_{setup} and the delay in waking up all sleeping nodes, which is upper bounded by the sleep period T_{sleep}:

$$T_{tree} \leq T_{setup} + T_{sleep} \tag{9}$$

From (9) and (6), we have:

$$T_{tree} \leq T_{setup} + T_{sleep} \leq T_f + T_{sleep} \tag{10}$$

Based on (10), (5) and (8), the deadline $k \cdot T_p$ will be met if the following condition holds:

$$t_{recv}(k) \leq k \cdot T_p - T_{sleep} - 2 \cdot T_f \tag{11}$$

The time it takes the *alert* message to travel between the two considered collector nodes is $\frac{v_{user} \cdot T_p}{v_{alt}} < T_p$. Hence, the *alert* message will be received before $t_{recv}(k)$ by the k^{th} collector node if:

$$t_{send}(k-1) \leq t_{recv}(k) - T_p \tag{12}$$

Based on (12) and (11), the deadline $k \cdot T_p$ will be met if the prefetch forwarding time of the $(k-1)^{th}$ collector node satisfies the following condition:

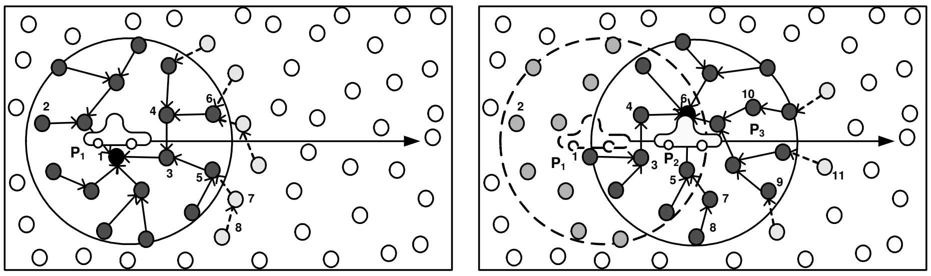
$$t_{send}(k-1) \leq (k-1) \cdot T_p - T_{sleep} - 2 \cdot T_f \tag{13}$$

We note that this bound is not tight as DTC-JIT may still meet query deadlines even if DTC forwards the *alert* message at a later time. However, our simulation results in Section 7.2 show that this bound is sufficient for DTC-JIT to achieve significant performance improvement over DTC-GP.

5.2 Directional Tree Maintenance

Although DTC can enhance the performance of spatiotemporal queries, it may incur high communication overhead and network contention when many consecutive query areas overlap each other because it creates a new query tree in each query period. DTM addresses this drawback by maintaining a *single* moving tree that is rooted at the user and travels along with the user. Like DTC, DTM depends on the user motion profile. In DTM, the query dissemination process is broken up into two phases: the initial *tree building* phase, followed by the *tree maintenance* phase. The data collection phase is similar to that in DTC.

In the tree building phase, DTM creates a spanning tree in the first query area after the user issues a *query* message that contains the query parameters and the user motion profile. Active nodes receiving the *query* message set the user or an active neighbor that is closer to the user location, as their parent. These nodes then propagate the query information through broadcasted *join* messages. Active nodes in the current or future query areas join the tree upon receiving a *join* message, by either setting the source node or a neighbor that is closer to the pickup point of the first query area that they are a part of, as their parent. Note that the *join* message contains the user motion profile and hence each node can compute the future query areas and pickup points. Like in DTC, only active nodes form the internal nodes of the tree. Active nodes hold the *join* message and deliver it to the sleeping nodes when they wake up. Sleeping nodes thus form the leaves of the tree.



(a) Tree building phase. (b) Tree maintenance phase.

Fig. 3. DTM tree building and maintenance. P1, P2 and P3 are three pickup points along the user trajectory. Dark gray nodes denote query tree nodes that are in the current query area (shown by a solid circle). The query subtree root is denoted by a black node. The subtree root has the user as its parent. Lightest gray nodes denote nodes that are a part of the query tree but do not belong to the current query area. These nodes are connected to the tree using dashed arrows. Nodes that were in the previous query area (shown by a dotted circle) but are no longer in the current query area are shaded in light gray.

In the tree maintenance phase, nodes are woken up along the predicted user path and the tree moves in the direction of user motion through node additions and deletions. This is achieved as follows. On receiving a *join* message, a node u in a future query area joins the existing query tree by first computing the earliest query period j that it participates in, and then selecting an active neighbor closest to the j^{th} pickup point as its parent. The node then performs the following actions: (1) it precomputes its parents for all query periods that it participates in, (2) it rebroadcasts the *join* message at time $t_{join}(j)$ (derived in Section 5.2.1) if it is an active node, (3) after sending the result for the j^{th} query period at time $d_u(j)$ (given below), it adjusts its parent for the next query period that it participates in. Note how the tree grows by node additions and by local parent adjustments. Moreover, nodes that are not in any future query area automatically drop out of the tree. Thus, local computations on each node, based on neighborhood information and knowledge of the user motion profile, allow the query tree to move in the right direction without incurring extra communication overhead.

The tree maintenance phase is robust to *join* message loss, since the same message is broadcasted by a number of nodes, which leads to redundant messages. Therefore, to reduce overhead, a node does not retransmit the *join* message. Also, in DTM, a node always chooses the user if it is within the communication range of the pickup point, or the active neighbor closest to the pickup point, as its parent. However, if it is the closest to the pickup point but is not within the communication range of the pickup point, it is in a network void. Network voids can lead to cycles on the tree, thus reducing data fidelity. DTM prevents cycles by disallowing nodes within network voids from being parents in the query tree. Each node checks if it is within a network void with respect to each query period it participates in, and includes such information in the *join* message. A node only selects a node that is not within a network void, as its parent. When a node on the tree finds itself in a new network void (e.g., due to the change of query period), it notifies its neighbors by broadcasting a message. On receiving this message, the node's children select new parents that are not within network voids.

The data collection process in DTM is similar to that in DTC. Query results are aggregated via the query tree and delivered to the user by the root of the tree. Sleeping nodes are restricted to be the leaves of the query tree and hence wake up at time $k \cdot T_p - T_f$ to send the k^{th} query result. Active nodes send the query result according to a sub-deadline assignment scheme to facilitate data aggregation. Thus, an active node u sends the result for the k^{th} query period to its parent at time $d_u(k)$, given by:

$$d_u(k) = \begin{cases} k \cdot T_p & if\ parent\ is\ user \\ k \cdot T_p - \frac{|up|}{R_q} \cdot T_f & otherwise \end{cases} \tag{14}$$

The DTM protocol is shown in Fig. 4 and the tree building and tree maintenance phases are illustrated in Fig. 3(a) and Fig. 3(b), respectively. Fig.3(a) shows the initial query tree that is formed in query area 1, in response to a *query* message broadcasted by the user. The query tree consists of a single network query tree that is rooted at node 1. As seen in the figure, the query tree contains nodes that belong to query area 1 (nodes colored dark gray), like nodes 1-6 and also some nodes that belong to query area 2 (colored very light gray, with dashed arrows), like nodes 7 and 8. Nodes that do not belong to query area 1 (e.g., 7, 8) join the tree on receiving a *join* message, by selecting a parent that is closest to the pickup point of the first query area that they are part of. For example, node 8 selects node 7 as its parent, since among its neighbors, node 7 is the closest to the pickup point of query area 2 (P2), which is the first query area that node 8 belongs to. Even though nodes that do not belong to query area 1 are part of the tree, they do not participate in the data collection phase for query area 1. Once the nodes in query area 1 transmit the sensor data to their respective parent, they adjust their parent pointer to point to the neighbor that is closest to P2. For example, in Fig. 3(b), node 3 changes its parent pointer from node 1 to node 4, node 1 sets it parent to node 3 while node 6 finds itself to be the closest to the second pickup point P2 among its neighbors, and hence becomes the new root. Nodes like 2 (colored light gray) that do not belong to any future query area drop out of the tree while nodes like 9, 10 that belong to query area 2 get added to the tree. Nodes in query area 2 that are not part of the tree along with nodes like 11 that belong to the third query area get added to the tree when nodes belonging to query area 2 that are already part of the tree (like nodes 7, 8) broadcast a *join* message at time $t_{join}(2)$. The above process repeats when the user reaches P2 and so on.

5.2.1 *Forwarding Time Analysis.* Nodes in future query areas are woken up by forwarding the *join* message ahead of the user. The time to forward this message is critical for a query to meet its spatiotemporal constraints, and can be derived similarly as the forwarding time of DTC because the just-in-time prefetching scheme is employed by both protocols. However, DTM allows for a later forwarding time because the processes of disseminating the query to a future area and setting up the tree are combined in a single step in DTM while they are separate in DTC. Specifically, the time at which a node rebroadcasts the *join* message to notify its neighbors to join the tree, $t_{join}(k)$ (where k is the first query area that it is a part of), must satisfy:

$$t_{join}(k) \leq (k-1)T_p - T_f - T_{sleep} \tag{15}$$

We can see the forwarding time of DTM is T_f later than that of DTC (see (13)). The detailed derivation can be found in [Bhattacharya et al. 2005], and is omitted here due to space limitation.

```
When join message is received
(1) Accept if participating in current or future query area.
(2) For each query period $k$ that this node is a part of
    (a) Set active neighbor closest to the $k^{th}$ pickup point as parent.
    (b) If this node itself is the closest to the $k^{th}$ pickup point, set user as parent.
(3) If active node, calculate $t_{join}(k)$ according to (15) and set timer JoinTimer to fire at $t_{join}(k)$.
(4) Calculate time to send result according to (14) and set timer SendTimer to fire at the right time.
(5) If sleeping node, switch to sleep schedule.
When JoinTimer fires
(1) Broadcast Join message.
When SendTimer fires
(1) If sleeping node, wake up.
(2) Send query result to parent.
(3) Adjust parent for next query period.
(4) If sleeping node, switch to sleep.
```

Fig. 4. DTM Protocol.

```
When query message is received
(1) If there is a neighbor closer to the user, set it as parent, else set user as parent.
(2) If active node, calculate $n$ using (21) and send setup message.
(3) If sleeping node, switch to sleep after $nT_p$ s.
When setup message is received
(1) Accept if in query area.
(2) Set node source as parent.
(3) Rebroadcast the setup message if active node.
(4) Set SendTimer to fire at data collection time (given by (16)).
When SendTimer fires
(1) Send result to parent.
(2) If leaf and active node, broadcast wake-up message.
When wake-up message is received
(1) Accept if in circle $C$.
(2) Rebroadcast the message if active node.
(3) If sleeping node, switch to sleep after $nT_p$ s.
```

Fig. 5. OTC Protocol.

5.2.2 *Discussion.* DTM works only when the query areas overlap and hence the *join* messages can be forwarded continuously along the user's path. To address this limitation, we can choose DTM or DTC based on knowledge of whether the query areas overlap or not, which can be obtained from the user's motion profile. An alternative solution, which is robust to user motion changes, is to use a hybrid form of DTM and DTC where a single moving query tree is maintained as long as the query areas overlap, and a tree per query area is maintained otherwise. The switching between DTM and DTC can be done as follows. If the next query area does not overlap with the current one, only the node whose parent is the user sends an anycast *alert* message to the node closest to the pickup point of the next one. This results in conversion from DTM to DTC. Once the query areas start overlapping, DTC can be converted once again to DTM, by having the node closest to the pickup point of the query area overlapping the next query area, broadcast *join* messages to create the single moving tree.

5.3 Omni-directional Tree Creation

Both DTM and DTC assume the availability of a user motion profile. In situations where the user movement pattern is highly unpredictable, or the motion history information has high location error, it is not possible to wake up just the right nodes ahead of time. However, if the maximum user speed is known, we can wake up all the nodes in a circle C centered at the current user location, henceforth called *wake-up area*, such that nodes in n future query areas in C, are ready to aggregate and provide the query result to the user, irrespective of the user's actual speed and direction of motion. This is the approach taken by OTC. The constant n is called *wake-up horizon*; it is dependent on the node duty cycle and is chosen such that nodes in the n^{th} query area are woken up by the time the user reaches the query area.

In OTC, nodes are not aware of the user location ahead of time and hence learn about a pickup point only when the user reaches the pickup point and issues a new query instance. After the user arrives, nodes deliver the query result to the user within a certain delay T_{delay} which is set by the user. The *query* message contains the parameters described in Section 3 except T_f, which is replaced by T_{delay}, along with v_{max} and p_{user}, where v_{max} is the maximum user speed and p_{user} is the user location. A query tree is built at each pickup point, in response to a *query* message, by flooding a *setup* message in the query area. The message contains query information, the pickup point number and the value of the wake-up horizon. Each node u on the tree then calculate the time $d_u(k)$ after which the result needs to be sent to the parent as follows:

$$d_u(k) = \begin{cases} k \cdot T_p + T_{delay} & if\ parent\ is\ user \\ k \cdot T_p + \frac{R_q - |up|}{R_q} \cdot T_{delay} & otherwise \end{cases} \tag{16}$$

Once the result is sent, active leaf nodes further broadcast a *wake-up* message containing T_p, n, p_{user}, and R, where R is the radius of the circle C. The message is flooded in the circle C. All future participating nodes are woken up ahead of time, and go back to sleep only after nT_p seconds. R is chosen such that sleeping nodes in the n^{th} query area relative to the current query area are woken up by the time the user reaches the query area. Thus,

$$R = nv_{max}T_p + R_q \tag{17}$$

where $nv_{max}T_p$ is the distance between the n^{th} pickup point and the current pickup point while R_q is the maximum distance of a node in the n^{th} query area from the corresponding pickup point.

Let $t(d)$ represent the time taken by a message to travel distance d. $t(d)$ can be calculated by:

$$t(d) \le \alpha \lceil \frac{d}{R_c} \rceil \tau \tag{18}$$

where R_c is the communication range, α is the *network dilation* and τ is the one-hop broadcast delay. Network dilation is defined as the upper bound on the ratio of the hop count between any two nodes to the minimum hop count $\lceil \frac{d}{R_c} \rceil$ between them, where d is the Euclidean distance between two nodes. Network dilation is shown to be bounded in sensor networks with sensing coverage [Xing et al. 2004]. For networks without sensing coverage, the network dilation can be measured as shown in [Huang et al. 2003]. In our simulations, we use the network dilation bound as given in [Xing et al. 2004] and measure the one-hop broadcast delay by timestamping messages and calculating the delay in message reception.

The time it takes the user to travel R is nT_p, while the time it takes a message to travel R is bounded by $t(nT_pv_{max} + R_q) + T_{sleep} + \epsilon$ where ϵ is the additional time taken by a node to send out a *wake-up* message. Since a node sends out a message only after sending the query result, we have $\epsilon \le T_{delay}$, where T_{delay} (defined in Section 5.3) is the maximum delay between the time when the user issues the query at a pickup point and when the query result is received. Since the message should reach the nodes before the user gets to the pickup point,

$$nT_p \ge t(nT_pv_{max} + R_q) + T_{sleep} + T_{delay} \tag{19}$$

Applying (18) in (19), we see that the timing constraints of the query will be met, if

$$nT_p \geq \alpha(\frac{nT_p v_{max} + R_q}{R_c} + 1)\tau + T_{sleep} + T_{delay} \tag{20}$$

In our protocol we use the minimum n that satisfies this inequality, denoted as n^*. Thus,

$$n^* = \frac{\alpha(R_q + R_c)\tau + (T_{sleep} + T_{delay})R_c}{T_p(R_c - \alpha\tau v_{max})} \tag{21}$$

From (17) and (21), the radius of the wake-up area can be derived:

$$R = \frac{\alpha(R_q + R_c)\tau + (T_{sleep} + T_{delay})R_c}{R_c - \alpha\tau v_{max}} \cdot v_{max} + R_q \tag{22}$$

6. ANALYSIS

In this section, we analyze MobiQuery under the three prefetching protocols: DTC, DTM, and OTC. In Section 6.1, we derive the *warmup interval* during which sleeping nodes cannot be woken up in time and hence MobiQuery only yields best-effort performance. In Section 6.2, we analyze the communication overhead of the three protocols.

6.1 Warmup Interval

MobiQuery may experience *warmup intervals* denoted by T_w, during which not all the required nodes are alerted about the query through prefetching, and hence some query results may violate query deadlines. DTM and DTC experience a warmup interval every time the user motion profile changes while OTC experiences it only once at the start of the query. During the warmup interval, prefetching cannot be performed in time, i.e., conditions (13), (15), and (19) do not hold for DTC, DTM, and OTC, respectively. Thus, the protocols use greedy prefetching during this period and switch to just-in-time prefetching once the prefetching conditions hold. We now analyze the upper bound on T_w for DTC, DTM and OTC, labeled T_{w-DTC}, T_{w-DTM} and T_{w-OTC}, respectively. This bound quantifies the robustness of DTC and DTM in the presence of user motion changes. Its significance is however reduced for OTC, which just experiences a warmup interval at the start, irrespective of the user motion pattern. We first evaluate T_w for DTC (T_{w-DTC}), followed by that for DTM and OTC.

Suppose DTC receives a new motion profile T_a seconds before the motion change occurs and the warmup interval lasts T_{w-DTC} seconds since the motion profile is issued. Let p_k denote the pickup point where the warmup interval ends. The user needs to travel a distance of $v_{user} \cdot (T_{w-DTC} + T_a)$, where v_{user} is the user velocity, to reach p_k from the point where it receives the motion profile. We approximate the locations of collector nodes with those of the corresponding pickup points. Then, the time it takes the *alert* message to reach p_k, T_{p-DTC}, is as follows:

$$T_{p-DTC} = \frac{v_{user} \cdot (T_{w-DTC} + T_a)}{v_{alt}} \tag{23}$$

where v_{alt} is the velocity of the *alert* message. After the message reaches the pickup point, it takes $T_{tree} + T_f$ seconds to build the query tree and collect the results. In the worst case, the deadline of the last query period in the warmup interval cannot be met. Hence, we have:

$$T_{w-DTC} + T_a \leq T_{p-DTC} + T_{tree} + T_f \tag{24}$$

Solving T_{w-DTC} using (24) and (23):

$$T_{w-DTC} \leq \frac{T_{sleep} + 2T_f - (1 - \frac{v_{user}}{v_{alt}}) \cdot T_a}{1 - \frac{v_{user}}{v_{alt}}} \tag{25}$$

Thus, T_{w-DTC} becomes zero when $T_a = (2T_f + T_{sleep})/(1 - \frac{v_{user}}{v_{alt}})$. That is, when the motion of the user can be predicted early enough, the query does not incur any warmup interval. In addition, $v_{alt} \gg v_{user}$ holds in practice. As a quantitative estimation of $\frac{v_{user}}{v_{alt}}$, let us consider the following simple example on MICA2 motes [Crossbow 2003]. Suppose two consecutive collector nodes are $100m$ apart and there are 5 hops between them. The size of an *alert* message is 60 bytes. The bandwidth of a mote is 38.4 Kbps [Crossbow 2003]. To account for routing/MAC overhead and contention delay, we assume the effective bandwidth of a mote is 5 Kbps. Then v_{alt} can be calculated as follows:

$$v_{alt} = \frac{100m}{5 \times (60\, bytes \times 8)/5000\, bps} \times \frac{3600s}{1000 \times 1.6} \approx 469\, mph$$

Obviously, v_{alt} is much greater than the velocity of a human or a vehicle. Approximating $\frac{v_{user}}{v_{alt}}$ to zero, we get $T_{w-DTC} \approx T_{sleep} + 2T_f - T_a$.

The warmup interval of DTM is similar to that of DTC because both protocols employ greedy prefetching in the warmup phase. The two protocols' operation in the warmup phase, however, differs in how the nodes are woken up. Nodes in DTM are woken up in the broadcast process that spans multiple adjacent query areas while an unicast is used in DTC to first notify adjacent query areas which is then followed by a broadcast process within each query area to wake up nodes. A quantitative example, similar to the one given earlier in this section, can show that the speed of broadcast messages is orders of magnitude higher than that of the user. Therefore, the nodes in DTM are woken up at roughly the same time as in DTC, from the user's perspective. As a result, the warmup interval of DTM can be obtained using a derivation similar to that for DTC and hence, $T_{w-DTM} = T_{w-DTC} \approx T_{sleep} + 2T_f - T_a$.

We now derive the warmup interval of OTC denoted by T_{w-OTC}. OTC alerts nodes by broadcasting *wake-up* messages to all nodes in a query area. Suppose k is the last query period within the warmup interval. The time it takes to alert the nodes in the first k query periods, T_{p-OTC}, can be derived as follows:

$$T_{p-OTC} = \frac{v_{user} \cdot T_{w-OTC}}{v_{msg}} + \frac{R_q}{v_{msg}} + T_{sleep} \tag{26}$$

where the first term is the time taken by a *wake-up* message to reach the k^{th} pickup point. The second term, $\frac{R_q}{v_{msg}}$, is the time taken to alert the rest of the active nodes in the k^{th} query area. v_{msg} represents the broadcast speed of the *wake-up* message. We assume the process of broadcasting the *wake-up* message from the pickup point to the active nodes in a query area incurs the same delay as the data collection process in which each node sends its result to the user at the pickup point, because these two processes involve a similar amount of communication. According to the design of OTC presented in Section 5.3, the data collection process must complete within T_{delay} seconds specified by the user. Hence, $\frac{R_q}{v_{msg}} < T_{delay}$. The third term in (26), T_{sleep}, is the wake-up delay of the sleeping nodes in the query area. Since the query tree in a query area is created only after the user reaches the corresponding pickup point, the deadline of the last period in the warmup interval, i.e., the k^{th} query period, cannot be met when $T_{w-OTC} \leq T_{p-OTC}$. T_{w-OTC} can be solved from this inequality and (26), as shown below:

$$T_{w-OTC} \leq \frac{T_{sleep} + T_{delay} - (1 - \frac{v_{user}}{v_{msg}}) \cdot T_a}{1 - \frac{v_{user}}{v_{msg}}} \approx T_{sleep} + T_{delay} \quad (27)$$

As seen above, T_{w-DTC}, T_{w-DTM} and T_{w-OTC} are affected by the node sleep period T_{sleep}. As expected, the longer the sleep period, the longer the warmup interval. Moreover, T_{w-DTC} and T_{w-DTM} also depend on the advance time T_a. The warmup interval is reduced, when the motion profile is obtained before the start of the motion profile, i.e. $T_a > 0$. However, when the motion profile is obtained using the history based prediction method (as explained in Section 4.1.2), $T_a = -T_s < 0$ and the warmup interval increases with T_s. Let us consider the following example. Suppose that the nodes wake up every $T_{sleep} = 9s$, the freshness requirement (T_f) is $0.5s$ and that a query result must be returned to the user within $T_{delay} = 0.5s$ after the user issues a new query instance. Also, let us assume that a motion predictor based on history is used to generate user motion profiles and that the user's location is sampled from a GPS every $T_s = 12s$. Thus, the advance time of the motion profile, T_a, is equal to $-T_s = -12s$. It is important to note that the choice of T_s is a tradeoff between the length of the warmup period and the amount of location error. The shorter the sampling period, the shorter the warmup period but larger the location error. In this example, we choose a value of T_s that reduces the location error. The warmup interval of different protocols can be calculated as follows:

$$\begin{aligned} T_{w-DTC} &= T_{w-DTM} = T_{sleep} + 2T_f - T_a = 9 + 2 \cdot 0.5 + 12 = 22s \\ T_{w-OTC} &= T_{sleep} + T_{delay} = 9 + 0.5 = 9.5s \end{aligned}$$

For example, assuming the user maintains approximately the same velocity for 4 minutes, the warmup interval corresponds to 9.2% of the duration of the query in DTC and DTM, and 4% of the duration of the query in OTC. Furthermore, in practice, the query areas based on a new motion profile and the ones based on the previous motion profile may overlap, which reduces the effect of the warmup interval during motion changes.

6.2 Communication Overhead

In this section, we analyze the communication overhead incurred by the different protocols. We quantify the communication overhead of a protocol by the average number of control messages sent during a query period. This metric is also related to the likelihood of network contention. In particular, due to low node duty cycles, the setup of a query tree may last multiple query periods resulting in interference among adjacent query areas. Therefore, a protocol that sends fewer control messages per query period likely causes a lower level of network contention among overlapping query areas. In the rest of this section, we assume that there are an average of N_a active nodes in each query area and that the total number of active nodes that participate in the query is N_p.

6.2.1 *Number of Control Messages Sent by DTC.* We now analyze the number of control messages sent by DTC-GP and DTC-JIT. Our analysis shows that although just-in-time prefetching sends the same total number of control messages as greedy prefetching , it incurs smaller per-period communication overhead by scheduling the transmissions of control messages at the right time.

Since each active node in DTC-GP sends a *setup* message for each query area that it is a part of, the total number of *setup* messages sent during a query is $N_a \cdot \frac{T_{life}}{T_p}$. The time interval during which these messages are sent is $v_{user} \cdot T_{life}/v_{alt} + T_{setup} + T_{sleep}$, in which $v_{user} \cdot T_{life}/v_{alt}$ is the time taken by the *alert* message to travel from the first pickup point to the last, T_{sleep} is the wake-up delay of the sleeping nodes in the last query area, and T_{setup} is the delay of delivering the *setup* message to all active nodes in the last query area. As discussed in Section 5.1.1, $T_{setup} \leq T_f$. The communication overhead of DTC-GP, C_{GP}, can be calculated as the total number of control messages divided by the number of query periods it takes to send these messages:

$$C_{GP} = \frac{N_a \cdot \frac{T_{life}}{T_p} \cdot T_p}{\frac{v_{user} \cdot T_{life}}{v_{alt}} + T_{sleep} + T_{setup}} \leq \frac{N_a}{\frac{v_{user}}{v_{alt}} + \frac{T_{sleep}+T_f}{\cdot T_{life}}} \approx N_a \cdot \frac{T_{life}}{T_{sleep} + T_f} \tag{28}$$

We now derive the number of *setup* messages sent by DTC-JIT, which is denoted by C_{JIT}. It is easy to see that the total number of control messages sent by DTC-JIT is the same as in DTC-GP. However, the time it takes to send these message is much longer, which results in a lower per-period communication overhead and network contention level. As discussed in Section 5.1, DTC-JIT only sends one *alert* message and hence builds only one tree in every query period after the warmup interval. In other words, each active node under DTC-JIT sends only one *setup* message in each period, resulting in the same number of *setup* messages for every query period. Hence, C_{JIT} is simply N_a. From (28), we can see that the average number of control messages sent by DTC-GP per period is $\frac{T_{life}}{T_{sleep}+T_f}$ times that of DTC-JIT. This result shows that just-in-time prefetching significantly reduces the per-period communication overhead by delaying transmissions of control messages.

6.2.2 *Number of Control Messages Sent by DTM.* We have showed that just-in-time prefetching effectively reduces per-period communication overhead compared to greedy prefetching. DTM can further reduce communication overhead because, unlike in DTC, a node in DTM broadcasts only a single *join* message even when it lies within multiple query areas. As a result, DTM generates much less network traffic and hence much lower network contention. For the convenience of discussion, we define the *non-overlapping factor* β as the ratio of total area of the region that falls into at least one query area to the total area of all query areas. β is within $(0, 1]$ and quantifies the level of non-overlap between adjacent query areas. When there is no overlap between any two adjacent query areas, $\beta = 1$. The total number of *join* messages sent by DTM is N_p. We assume that the active nodes in the network are distributed uniformly. Hence:

$$N_p = \beta \cdot \frac{T_{life}}{T_p} \cdot N_a \tag{29}$$

The number of *join* messages sent by DTM per query period, C_{DTM}, can be calculated as follows:

$$C_{DTM} = \frac{N_p}{\frac{T_{life}}{T_p}} = \frac{N_a \cdot \beta \cdot \frac{T_{life}}{T_p}}{\frac{T_{life}}{T_p}} = \beta \cdot N_a \tag{30}$$

As discussed in Section 6.2.1, the number of control messages sent by DTC per period is N_a. Thus, DTM causes a lower communication overhead when query areas overlap (i.e.,

$\beta < 1$).

6.2.3 *Number of Control Messages Sent by OTC.* According to Fig. 5, in each query period of OTC, each active node in the query area sends a *setup* message and each active node in the wake-up area C sends a *wake-up* message. Hence the total number of control messages is approximately equal to $N_a \cdot \frac{R^2}{R_q^2}$ where R is the radius of wake-up area defined in (17)[4]. Although the total number of control messages sent by OTC is higher than DTM, the control messages are sent over a larger geographic region. As a result, the contention level of OTC is moderate compared to DTM. This intuition is confirmed by our simulation results presented in Section 7 which show that OTC's performance is comparable to that of DTM.

7. SIMULATION RESULTS

We implemented the three prefetching protocols in NS-2. We first evaluate the proposed protocols with accurate user motion profiles in Sections 7.2 and 7.3. The best protocols are then evaluated in a more realistic scenario where the user's motion profiles are inaccurate, i.e., the user location readings may have errors and the user may diverge from the motion profile due to unexpected motion changes, Section 7.4.

We use the following metrics in our performance evaluation: (1) The *data fidelity* of a query instance, defined as the ratio of the number of nodes that contribute to a query result to the total number of nodes in a query area. (2) The *success ratio* of a query, defined as the ratio of the number of query instances that meet deadlines and whose data fidelity exceed a threshold, to the total number of query instances. The success ratio indicates the overall quality of service received by the user. (3) The *average energy consumption* of each node normalized by the total number of query results received by the user. (4) The *communication overhead* of a protocol, defined as the total number of control messages sent by a protocol, normalized by the total number of query results received by the user.

7.1 Experimental Settings

In each simulation, 200 nodes are randomly distributed in a $450m \times 450m$ region. IEEE 802.11 with the extended power saving mode (PSM) from [Chen et al. 2001] is used as our MAC protocol. The active window in PSM is $100ms$. The sleep period varies from $3s$ to $15s$, which results in duty cycles from 3.3% to 0.67% for sleeping nodes. The radius of a query area is $150m$. Coverage Configuration Protocol (CCP) [Xing et al. 2005] is used as the power management protocol. CCP maintains network connectivity and sensing coverage through a backbone. The communication and sensing range in CCP are set to $105m$ and $50m$, respectively. Under these settings, a query tree has about $2 \sim 4$ levels. The node bandwidth is 2 Mbps. Each simulation lasts for 400 seconds. Each data point is the average of results under three different network topologies.

In Sections 7.2 and 7.3 where we consider the knowledge of an accurate motion profile, the user starts from a corner of the region and moves in a random direction with a speed randomly chosen within a range. Three speed ranges are used, $3 \sim 5m/s$, $6 \sim 10m/s$ and $16 \sim 20m/s$, corresponding to a walking human, a running human and a vehicle with moderate speed, respectively. If the user hits the boundary of the region before the

[4]The formula gives an upper bound on the overhead of OTC as non-leaf nodes on query tree do not send *wake-up* messages.

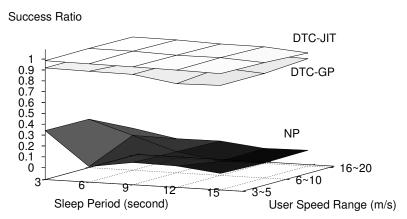


Fig. 6. Success ratios of MobiQuery under DTC-GP, DTC-JIT, and No-prefetching.

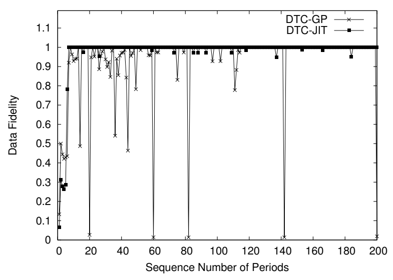


Fig. 7. Dynamic data fidelity of DTC-GP and DTC-JIT. The sleep period is $15s$ and the user speed is $3 \sim 5m/s$.

simulation ends, it changes it's direction toward a random point within the region. The user changes its direction and speed every 50 seconds. The motion profile that specifies the complete user path is provided to each protocol at the beginning of the simulation. Details of how inaccurate motion profiles are generated are provided in Section 7.4.

7.2 Advantage of Just-in-time Prefetching

We first study the advantage of just-in-time prefetching against greedy prefetching by comparing DTC-JIT against DTC-GP as DTC-JIT is the simplest protocol among all proposed just-in-time protocols. Detailed performance comparison between DTC-JIT, DTM and OTC is presented in Section 7.3. We set the threshold for data fidelity at 95%, in the success ratio metric. A protocol called No-prefetching (NP) is implemented as the baseline for our performance evaluation. In NP, no prefetching is used and the user broadcasts a query to the network at the beginning of each query period. In this set of simulations, query period and data freshness constraints are set to 2s and 1s, respectively.

Fig. 6 shows the average success ratios of NP, DTC-GP and DTC-JIT. The success ratio of DTC is nearly 100% in all settings, even when the sleep period is as long as $15s$, i.e., 7.5 times the query period. In sharp contrast, the success ratio of NP remains below 35% due to the high packet loss rate. NP performs worse when the sleep period or user speed increases. Packet losses in NP are mainly due to packet collisions because *setup* messages are broadcast to the nodes in different query areas within a very short period of time. The success ratio of DTC-GP reaches about 90% when the sleep period is shorter than $9s$ and decreases when the sleep period becomes longer. DTC-GP performs consistently *worse* than DTC-JIT due to packet losses caused by packet collisions.

To examine the dynamic behavior of MobiQuery, we plot the data fidelity at each pickup point in Fig. 7. Both DTC-JIT and DTC-GP suffer from an initial warmup phase in which about 5 queries have relatively low data fidelity, which conforms to the analytical warmup interval in Section 6.1. DTC-JIT achieves a data fidelity of 100% in most periods after the warmup phase. In contrast, the performance of DTC-GP varies significantly during a query, with its data fidelity dropping to almost zero in several query periods. We observed that the severe performance degradation is caused by excessive network contention which results in significant packet loss during query dissemination. This result conforms to our analysis on communication overhead in Section 6.2. The high performance variation of DTC-GP

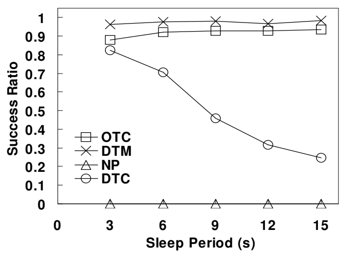


Fig. 8. Success ratios of DTC, DTM and OTC. The user speed is $3 \sim 5m/s$. The query period is $0.5s$.

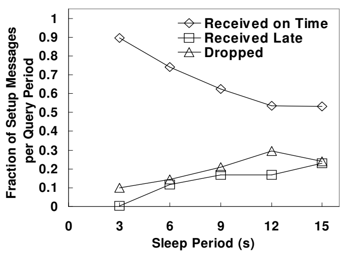


Fig. 9. The fractions of *setup* messages per query period that is dropped, received late and on time under DTC.

makes it unsuitable for mission-critical applications that require predictable performance assurance.

7.3 Comparison of DTC, DTM and OTC

In the previous section, we showed the advantage of just-in-time prefetching over greedy prefetching. In this section, we compare the three just-in-time prefetching protocols, DTM, OTC, and DTC (we omit the JIT in DTC-JIT in the rest of this section). We evaluate the effect of various factors on the performance of the three protocols, including query period, sleep period, and user speed. In these simulations, the data freshness constraint is set to $0.5s$ which represents a tighter temporal constraint for the query results. The threshold of data fidelity is set to 90% for the success ratio.

Fig. 8 shows the average success ratios of all protocols with different sleep periods, when the query period is 0.5s. The success ratio of NP remains below 10% in all settings, which clearly indicates that the prefetching mechanism used by other protocols is crucial in networks with low duty cycles. The performance of DTC drops quickly when sleep period increases. This is due to its high communication overhead, which causes more packets to be dropped or delayed, thus decreasing the data fidelity, as the sleep period increases. This explanation is validated by Fig. 9. The figure shows that the fraction of *setup* messages[5] that are dropped/delayed per query period increases as the sleep period increases. As a result, the fraction of *setup* messages received per query period and hence the data fidelity decreases as the sleep period increases. Unlike DTC, OTC maintains a success ratio of above 90% in most settings. Its performance degrades slightly when the sleep period becomes shorter. This result can be explained as follows. OTC wakes up sleeping nodes before a tree is created. As the sleep period becomes shorter, more traffic (i.e., overhead packets of 802.11 PSM and CCP) is generated by the active nodes in the network, resulting in higher network contention during tree creation. In contrast, DTM achieves a success ratio of nearly 100% in all settings since it requires much lower bandwidth.

7.3.1 *Effect of User Speed.* Fig. 10 shows the performance of the different protocols when the user moves at different speeds. Four speed ranges: 3m/s $\sim$ 5m/s, 6m/s $\sim$ 10m/s, 16m/s $\sim$ 20m/s and 36m/s $\sim$ 40m/s are used, which correspond to the speeds for walking or

[5]*Setup* messages are used to build the tree per query area in DTC.

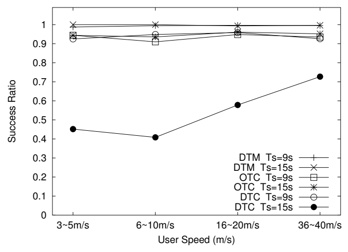


Fig. 10. Effect of User Speed. The sleep period is $9s$. The query period is $1s$.

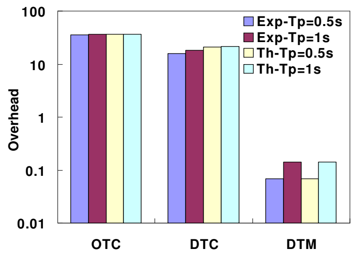


Fig. 11. The number of control messages per query instance. The sleep period is $9s$, the user speed is $3 \sim 5m/s$.

jogging, running, driving at moderate speed and driving at high speed, respectively. When the user speed increases, the distance between two consecutive query areas becomes larger, which reduces the level of network contention and hence results in higher query success ratios. Although DTC delivers above 90% of the query results under all user speeds when the sleep period is $9s$, its performance degrades significantly when the sleep period is $15s$. This is because DTC fails to wake up many sleeping nodes in time due to the high network contention level caused by its high communication overhead. On the other hand, the success ratio of both DTM and OTC remains above 90% under all settings. This result shows that both protocols adapt successfully to different ranges of user speeds by waking up sleeping nodes in advance, with moderate communication overhead.

7.3.2 *Communication Overhead.* As mentioned earlier, we define the overhead as the total number of messages (except query messages and query results) sent by a protocol, normalized by the total number of query results received by the user. Fig. 11 shows the different protocol overheads when the sleep period is $9s$. Along with the experimental results we also plot the expected overhead of the protocols based on our theoretical analysis in Section 6.2. Note that the Y-axis of the figure is in log-scale due to the high variance in the overhead of the protocols. Several interesting results can be seen from Fig. 11. First, DTM incurs much lower communication overhead than the other protocols. This is due to the low overhead of query tree maintenance in DTM. OTC, on the other hand, incurs the highest overhead among the three protocols since it wakes up a large number of nodes due to lack of knowledge of the user motion profile. Even though OTC has a higher communication overhead, it is offset by its better performance in comparison to DTC, as seen in Fig. 8. Second, the overheads of all protocols except DTM are independent of the query period. In the case of DTM, according to the analytical result (Eqn. (30)) in Section 6.2, the overhead is proportional to the non-overlapping factor, β, defined as the ratio of total area of the region that falls into at least one query area to the total area of all query areas. β is determined by the query radius and the distance between two consecutive query areas which in turn is proportional to the query period. Consistent with the analysis, the number of control messages per query result in DTM drops roughly by half (from 0.14 to 0.069) when the query period decreases from $1s$ to $0.5s$, which demonstrates that DTM can effectively reduce the overhead by taking advantage of overlap between adjacent query areas. The simulation results are very close to the theoretical bound for all protocols.

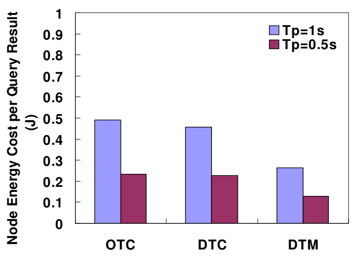


Fig. 12. Total network energy consumption per query result. The sleep period is $9s$, the user speed is $3 \sim 5m/s$.

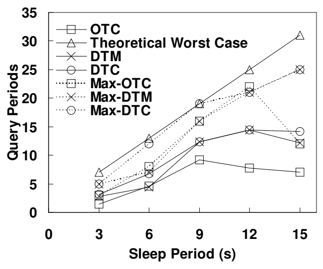


Fig. 13. Warmup interval. Query period is $0.5s$ and the user speed is $3 \sim 5m/s$.

7.3.3 *Energy Consumption.* We used the energy model of Cabletron 802.11 network card [Chen et al. 2001]. In accordance with this model, the power consumption of transmit, receive, idle and sleeping modes were set to 1400mW, 1000mW, 830mW and 130mW respectively. Fig. 12 shows the per node energy expenditure of the different protocols, normalized by the total number of query results received by the user, when the sleep period is $9s$. As seen in the figure, OTC has the highest energy expenditure among the protocols. This is to be expected, since in OTC, a large number of nodes are woken up due to the lack of a user motion profile. DTM has the lowest energy expenditure as expected, since it has a much lower overhead than the other protocols as shown in Fig. 11.

7.3.4 *Warmup Interval.* Fig. 13 shows the average and maximum warmup intervals of the different protocols, obtained in our experiments. We compare these warmup intervals with the theoretical worst case values presented in Section 6.1. Due to the particular settings of the experiments, the theoretical worst case warmup intervals of all three protocols evaluate to the same values and hence is plotted as a single curve in the figure. As expected, the maximum warmup intervals of all three protocols remain within the theoretical bounds under varying sleep periods. The average warmup intervals of the protocols are significantly lower than the worst case values indicating better average performance than the theoretical bounds. This is expected as the theoretical bounds are derived based on worst-case scenarios.

7.3.5 *Effect of Node Density.* Fig. 14 shows the success ratios of the different protocols under varying node densities. The success ratio of a particular protocol remains nearly constant under different node densities and drops drastically only when the number of nodes drops to 50. This is because, despite the different network densities, the underlying power management protocol (CCP) maintains the sensing coverage of the network deployment region using roughly the same number of active nodes. We plot the number of active nodes in the network under different node densities in Fig. 15. We can see that the number of active nodes remains nearly constant ($\sim$ 60) as long as the the total number of node is greater than 50 nodes. As a result, the protocols perform similarly as long as the region is sensing-covered. This result is consistent with the analysis in Section 6.2, which shows that the overhead of the protocols is only dependent on the density of active nodes. When the total number of nodes falls to 50, the region is no longer sensing-covered and hence all nodes are activated by CCP. As a result, routing voids exist in such a case, which

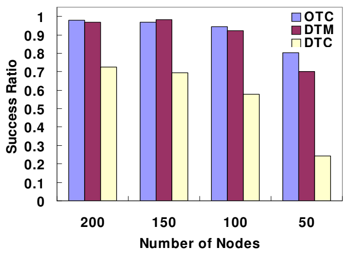


Fig. 14. Success ratio for different node densities. The query period is $0.5s$, the user speed is $3 \sim 5m/s$, and the sleep period is $9s$.

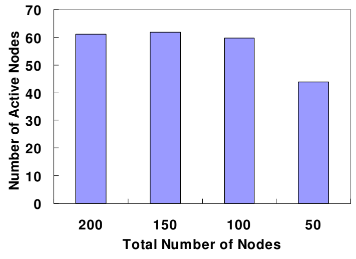


Fig. 15. Number of active nodes for varying node densities.

causes packet loss and lower success ratios for all protocols. [6] According to our analysis in Section 6.2, all three protocols incur less overhead and expect to perform better when the number of active nodes is smaller.

The results in this section indicate that 1) DTM and OTC are more robust to effects of sleep periods and user speed than DTC; 2) DTM incurs the minimal overhead and energy expenditure among all protocols; 3) the simulation results match the analytical results in Section 6.

7.4 Performance under Inaccurate Motion Prediction

In this section, we evaluate the performance of the proposed protocols under a more practical scenario where the user's motion profile is obtained using user motion history (as explained in Section 4.1.2) and is hence inaccurate, due to location errors and unexpected motion changes. The parameters of the motion predictor of MobiQuery discussed in Section 4.1.2 are set as follows. The user's location is sampled from a GPS every $T_s = 12s$, and two location readings are extrapolated for a time $T_{exp} = 50s$ to obtain a user motion profile. Everytime the GPS is sampled, the location reading is compared with the current user location calculated from the motion profile and a new motion profile is generated if the difference exceeds a threshold of $20m$.

As shown in the previous sections, NP and DTC cannot provide satisfactory performance even with accurate motion profiles. Hence, we focus on the performance of OTC and DTM in this section. The results are the average of 5 runs under different network topologies. We set the threshold of data fidelity at 80% for the success ratio metric.

7.4.1 *Effect of Location Error.* Fig. 16 shows the performance of DTM and OTC under different location errors. As expected, location error has minimal impact on OTC, since OTC does not make use of the user motion profile, and always wakes up nodes in a

[6]We note that the number of active nodes required to cover the deployment region depends on the spatial distribution of nodes. According to [Kershner 1939], when the circles are placed on hexagons, the number of circles required to cover a region is minimum, which is approximately $2A\sqrt{3}/(9r^2)$ where A is the area of the region and r is the radius of the circles. In our settings, the sensing range is $50m$ and the area of deployment region is $450m \times 450m$, and the minimum number of active nodes is about 32. The number of active nodes in our simulations therefore corresponds to reasonable densities given the random node distribution used in our simulations.

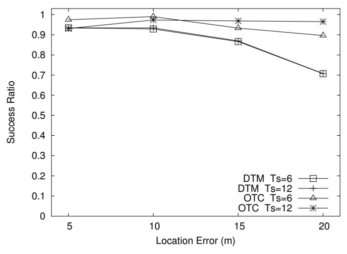


Fig. 16. Success ratios with location errors. The query period is $0.5s$, the user speed is $3 \sim 5m/s$, and the sleep period is $15s$.

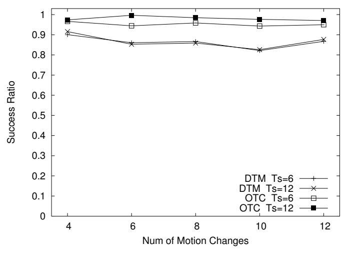


Fig. 17. Success ratios vs. num of motion changes. The query period is $0.5s$, the user speed is $3 \sim 5m/s$, and the sleep period is $15s$.

large area around the user that covers all possible user locations during the next few query periods. DTM, on the other hand, performs very well for location errors less than 15m (typical GPS location error is 10m-15m [Cooksey 2004]), and is able to deliver 80% of the query results successfully. These results show that the design of DTM is robust to moderate location errors.

7.4.2 *Effect of Motion Changes.* Next, we evaluate the performance of the protocols under different number of changes in the user's velocity. The user moves in a straight line at a constant speed until it makes a turn and chooses a new speed. Fig. 17 shows the performance of the protocols when the number of turns increases from 4 to 12 within a simulation time of $500s$. The location error for all the simulation runs is fixed at 10m.

As expected, the performance of OTC is not affected by the user motion changes. DTM performs slightly worse than OTC, because when the user makes a turn, the partial query tree that resides around the original predicted path becomes invalid and new nodes have to be woken up in order to form a query tree around the new predicted path. Consequently, some sleeping nodes along the new path may not be woken up early enough to respond to the query. However, DTM still maintains a success ratio over 80% in all settings.

8. CONCLUSION

In this paper, we present a new spatiotemporal query service called MobiQuery that allows a mobile user to periodically query a surrounding area and propose three novel just-in-time prefetching protocols, Directional Tree Creation (DTC), Directional Tree Maintenance (DTM) and Omni-directional Tree Creation (OTC) to achieve desired spatiotemporal performance. Furthermore, we provide theoretical analysis on several important factors that are critical to the practical performance of MobiQuery in WSNs including warmup interval and communication overhead. We evaluate MobiQuery with different prefetching strategies through extensive simulations under realistic settings including low node duty cycles, frequent user motion changes, and significant location errors. Our results show that 1) DTC can achieve satisfactory spatiotemporal performance under extremely low node duty cycles and erroneous/late prediction of the user's movement; 2) DTM can further reduce communication overhead and network contention caused by continuous queries to overlapping query areas; and 3) OTC is particularly robust against unpredictable user movement patterns and location errors. Our analysis and simulation results provide guidance for choosing different prefetching protocols under different application requirements and network settings.

REFERENCES

ALANKUS, G., ATAY, N., LU, C., AND BAYAZIT, O. B. 2005. Spatiotemporal query strategies for navigation in dynamic sensor network environments. In *IEEE RSJ International Conference on Intelligent Robots and Systems (IROS)*.

ALJADHAI, A. R. AND ZNATI, T. 2001. Predictive mobility support for qos provisioning in mobile wireless environments. *JSAC 19(10)*.

ASLAM, J., BUTLER, Z., CONSTANTIN, F., CRESPI, V., CYBENKO, G., AND RUS, D. 2003. Tracking a moving object with a binary sensor network. In *Proceedings of the first international conference on Embedded networked sensor systems*. ACM Press, 150–161.

BHATTACHARYA, S., XING, G., LU, C., ROMAN, G.-C., HARRIS, B., AND CHIPARA, O. 2005. Dynamic wake-up and topology maintenance protocols with spatiotemporal guarantees. In *International Conference on Information Processing in Sensor Networks (IPSN)*.

BLUM, B., NAGARADDI, P., WOOD, A., ABDELZAHER, T., SON, S., AND STANKOVIC, J. 2003. An entity maintenance and connection service for sensor networks. In *MobiSys*.

CHEN, B., JAMIESON, K., BALAKRISHNAN, H., AND MORRIS, R. 2001. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *MobiCom*.

COOKSEY, D. 2004. Online report-http://www.montana.edu/places/gps/understd.html: Understanding the global positioning system (gps).

CROSSBOW. 2003. Mica and mica2 wireless measurement system datasheets.

FIREBUG-PROJECT. 2004. http://firebug.sourceforge.net/gps_tests.htm.

GUI, C. AND MOHAPATRA, P. 2004. Power conservation and quality of surveillance in target tracking sensor networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*.

HE, T., KRISHNAMURTHY, S., STANKOVIC, J. A., ABDELZAHER, T., LUO, L., STOLERU, R., YAN, T., GU, L., HUI, J., AND KROGH, B. 2004. Energy-efficient surveillance system using wireless sensor networks. In *Mobisys*.

HE, T., STANKOVIC, J. A., LU, C., AND ABDELZAHER, T. F. 2005. A spatiotemporal communication protocol for wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems 16,* 10.

HIGHTOWER, J. AND BORRIELLO, G. 2001. Location systems for ubiquitous computing. *IEEE Computer 34,* 8.

HUANG, Q., LU, C., AND ROMAN, G.-C. 2003. Spatiotemporal multicast in sensor networks. In *Sensys*.

INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. 2000. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom*.

INTANAGONWIWAT, C., GOVINDAN, R., ESTRIN, D., HEIDEMANN, J., AND SILVA, F. 2003. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw. 11,* 1.

KERSHNER, R. 1939. The number of circles covering a set. *Amer. J. Math.* 61.

KIM, H. S., ABDELZAHER, T. F., AND KWON, W. H. 2003. Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks. In *Sensys*.

LATOMBE, J.-C. 1991. *Robot Motion Planning*. Kluwer Academic Publishers.

LI, D., WONG, K., HU, Y. H., AND SAYEED, A. 2002. Detection, classification and tracking of targets in distributed sensor networks. *IEEE Signal Processing Magazine 19 (2)*.

LI, Q., ROSA, M. D., AND RUS, D. 2003. Distributed algorithms for guiding navigation across a sensor network. In *MobiCom*.

MADDEN, S., FRANKLIN, M. J., , HELLERSTEIN, J. M., AND HONG, W. 2002. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*.

PAPYRUS. 2004. Gps specification 2004.

PATTEM, S., PODURI, S., AND KRISHNAMACHARI, B. 2003. Energy-quality tradeoffs for target tracking in wireless sensor networks. In *The 2nd International Workshop on Information Processing in Sensor Networks (IPSN)*. Palo Alto,CA.

POLASTRE, J., SZEWCZYK, R., SHARP, C., AND CULLER, D. 2004. The mote revolution: Low power wireless sensor network devices. In *Hot Chips 16*.

XING, G., LU, C., PLESS, R., AND HUANG, Q. 2004. On greedy geographic routing algorithms in sensing-covered networks. In *Fifth ACM Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc)*. Tokyo, Japan.

XING, G., WANG, X., ZHANG, Y., LU, C., PLESS, R., AND GILL, C. 2005. Integrated coverage and connectivity configuration for energy conservation in sensor networks. *ACM Transactions on Senor Networks 1,* 1, 36–72.

XU, Y., HEIDEMANN, J., AND ESTRIN, D. 2001. Geography-informed energy conservation for ad hoc routing. In *MobiCom*.

YAO, Y. AND GEHRKE, J. 2002. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec. 31(2)*, 9–18.

YE, F., LUO, H., CHENG, J., LU, S., AND ZHANG, L. 2002. A two-tier data dissemination model for large-scale wireless sensor networks. In *MobiCom*. 148–159.

YE, W., HEIDEMANN, J., AND ESTRIN, D. 2004. Medium access control with coordinated, adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*.

ZHANG, W. AND CAO, G. 2004. Dctc: Dynamic convoy tree-based collaboration for target tracking in sensor networks. *IEEE Transactions on Wireless Communications 3(5)*.

ZHAO, F., SHIN, J., AND REICH, J. 2002. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*.