TRANSMISSION OF CONTINUOUS QUERY RESULTS IN MOBILE COMPUTING SYSTEMS

Hüseyin Gökmen Gök and Özgür Ulusoy

Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey.

Abstract

In a mobile computing environment, a user with a wireless connection to the information network can access data via submitting queries to data servers. As the mobility is the most distinguishing feature of the mobile computing paradigm, location becomes an important piece of information for the so called *location-dependent queries* where the answer to a query depends on the current location of the user who issued the query. A location-dependent query submitted by a mobile user can become more difficult to process when it is submitted as a *continuous query* for which the answer changes as the user moves. The answer to a location-dependent continuous query can be provided as a set of tuples $< S, begin, end >$ indicating that object S is the answer of the query from time *begin* to time *end*. Once the tuples in the answer set are determined, the next step is to determine the transmission time of these tuples to the user. Transmission time of tuples is critical in the sense that it can have a considerable impact on both the communication overhead imposed on the wireless network and the availability of tuples in case of disconnections. In this paper, we propose three tuple transmission approaches that can be used to determine the transmission time of tuples in the answer set of a location-dependent continuous query. We also evaluate the relative performance of the proposed approaches under different settings of environmental parameters.

Key words: Mobile Computing, Mobile Database Systems, Location-Dependent Queries, Continuous Queries, Simulation.

1 Introduction

Recent advances in computer hardware technology and wireless communication networks have led to the emergence of mobile computing systems [PB93, FZ94]. In a mobile computing environment, a user with a wireless connection to the information network does not require to maintain a fixed position in the network [Chr93, WC95]. As the mobility is the most distinguishing feature of the mobile computing paradigm, location becomes an important piece of information for the so called *location-dependent queries* [SWCD97, SWCD98, TUW98, WXCJ98]. Consider a database representing information about moving objects and their position in addition to information about stationary objects. A typical query submitted to a hotel management system might be: "display motels (with room price and availability) that are within 5 miles of my position"; or in a battlefield a typical query submitted might be: "display the friendly tanks within 10 miles of my position". Such queries may be issued from a moving object (e.g., car of a mobile user) or from a stationary user. Consequently, the answer to a location-dependent query may depend on the location of the mobile host (MH) which issued the query and/or the locations of the objects represented in the database.

A location-dependent query can become more difficult to process when it is submitted as a *continuous query* (CQ) [SWCD97, SWCD98]. The driver querying the motels in the above example may request the answer to the query to be continuously updated so that he/she can find a motel with a reasonable price. It is clear that the answer to such a query changes with the car movement and continuously updating driver's location would impose a serious performance and wireless bandwidth overhead. Existing database management systems are not well equipped to handle continuously changing data such as the position of moving objects, since the data is assumed to be constant unless it is explicitly modified. The position of a moving object changes continuously as a function of time. Hence, the answer to a CQ depends not only on the database contents but also on the time at which the query is issued.

In [SWCD97, SWCD98], a new data model called *Moving Objects Spatio-Temporal* (MOST) is proposed for databases containing position information about moving objects. MOST models the position of a moving object as a function of time. Therefore, the answer to the query: "retrieve the current position of the object O" in the MOST data model is different for time points t_1 and t_2 even if the value of the attribute specifying O's position has not been explicitly updated.

Consider again the CQ: "display motels within 5 miles of my position" issued by a person driving a car. When such a CQ is entered in the MOST data model, the query is evaluated once and a set of tuples is returned as the answer. The answer set consists of tuples $< S, begin, end >$ indicating that object S is the answer of the CQ from time *begin* to time *end*. Once the answer to the query is computed, a decision has to be made in order to determine the time to send the tuples in the answer set of the CQ to the MH. There are two basic approaches introduced in [SWCD97] to transmit the tuples to the MH: *Immediate Transmission* (IT) and *Delayed Transmission* (DT). In the IT approach, the whole answer set is transmitted immediately after being computed. In the DT approach, each tuple $< S, begin, end >$ is transmitted to the MH at time *begin*.

In this paper, we present three new approaches for the transmission of the tuples in the answer set of a location-dependent CQ. The first approach called *Periodic Transmission* (PT) transmits the tuples in the answer set periodically. At each w time units, this method transmits all the tuples $< S, begin, end >$ satisfying the condition $t \leq begin < t + w$ where t is the current time and w is the size of the time window. In the second approach which we call *Adaptive Periodic Transmission* (APT), as an extension to the first approach, w is dynamically adjusted according to the communication overhead changing due to environmental parameters such as data update rate, disconnection frequency, and disconnection period. The final approach, called *Mixed Transmission* (MT), differs from the first two approaches in that data objects are partitioned into two groups: one consisting of "hot" objects of updates and the other of "cold" objects of updates. This approach transmits the "hot" tuples as in APT and "cold" tuples as in IT.

We have implemented a simulation model of a mobile computing system that supports processing of CQs issued by MHs over the database of moving objects. The simulation model is used to study the performance of the proposed approaches in terms of the communication overhead and the availability of tuples in the answer set of a CQ in case of a disconnection, and also to investigate performance enhancements of these approaches over the basic schemes provided in [SWCD97, SWCD98].

The remainder of this paper is organized as follows. Section 2 provides the related work. In Section 3, the motivation for our work is discussed. In Section 4, we describe the approaches provided to determine the transmission time of the tuples in the answer set of location-dependent CQs. In Section 5, we present the simulation model that was used to evaluate the performance of the proposed approaches. Section 6 provides description of the experiments conducted and discussion of the results obtained. Concluding remarks are provided in Section 7.

2 Related Work

A mobile computing environment can be characterized by frequent disconnections of MHs, significant limitations of bandwidth and power, resource restrictions, and fast changing locations. All such characteristics associated with mobile systems make traditional techniques used in distributed computing systems inadequate and raise new challenging research problems.

There exist a considerable a number of papers discussing general issues and research challenges related to mobility. The new challenges in mobile data management are identified and their technical significance is investigated in [IB93, IB94]. [DHB97] focuses on the differences between data management solutions in a mobile computing environment and those in a distributed database environment. The impact of mobility on current software systems is discussed in [PB93]. Fundamental software design problems particular to mobile computing environments are addressed in [FZ94]. A general architecture for a mobile information system is described in [PB94].

Issues regarding mobile data management to respond to real-time data access requirements of supported applications are discussed in [Ulu98]. A variety of transaction execution models that take into account timing requirements of mobile computing system applications are proposed in [KU99a, KU99b].

The problems associated with the indexing of the dynamic attributes (such as location) in a mobile database system are addressed in [TUW98]. A variant of the quadtree structure for indexing dynamic attributes is proposed and an algorithm for generating the index periodically that minimizes the CPU and disk access cost is provided.

The problem of cache invalidation in mobile environments is addressed in detail in [BJ96]. The basic idea behind the Adaptive Periodic Transmission (APT) approach presented in our paper was inspired from the adaptive caching algorithm introduced in that paper. However, our context of adaptiveness is completely different. The problem we address is the determination of transmission times of the tuples in the answer set of a location-dependent continuous query, rather than the problem of cache invalidation. In order to adapt to the environmental parameters, the APT approach focuses on the overhead caused by the control messages and the retransmissions whereas the adaptive caching algorithm in [BJ96] deals with the overhead of the false cache invalidation.

The most relevant work to ours is the one presented in a series of papers [SWCD97, SWCD98, WXCJ98, WSCY99]. Issues related to moving objects databases such as indexing, location updates of moving objects, modeling, and querying moving objects are exclusively addressed in these papers. A new data model (MOST) is proposed to model moving objects. Future Temporal Logic (FTL) is provided as the query language for the MOST data model. An algorithm for processing FTL queries in the MOST data model is also provided. The MOST model supports continuous queries and their processing algorithm processes such queries without a recomputation of the query at each clock-tick. It is assumed by this model that there is a natural, user-friendly way of entering the current position and motion vector of objects into the database.

3 Motivation

Once the tuples in the answer set of a location dependent continuous query (CQ) are generated, the next step is to determine when to transmit these tuples to the mobile host (MH) which issued the query. There are two basic dimensions of the communication overhead regarding the transmission of the tuples in the answer set of a CQ:

1. Control Message Overhead: According to the point to point communication paradigm [SWD+96], a message to be transmitted is appended to a fixed size control message.

I.
$Ans(CQ_1) = \{< S, 3, 10 >\}$
$Ans(CQ_2) = \{< O, 7, 9 >\}$
(Initial answer sets)

II.
$Ans(CQ_1) = \{< S, 5, 13 >\}$
$Ans(CQ_2) = \{< O, 7, 9 >\}$
(Answer sets after an update on object S)

III.
$Ans(CQ_1) = \{\}$
$Ans(CQ_2) = \{< O, 7, 9 >, < S, 4, 7 >\}$
(Answer sets after another update on object S)

Figure 1: Possible Effects of an Explicit Update.

2. Tuple Retransmission Overhead: An explicit update to an object in the database may change the tuples referring to the updated object as shown in Figure 1. The same object may satisfy the query but *begin* and/or *end* attribute of the tuple may change (Figure 1, I and II). It is also possible that a tuple referring to the updated object may no longer satisfy the query (Figure 1, II and III), and/or a new object may satisfy one or more of the active queries that it did not satisfy previously (Figure 1, II and III).

Suppose that the subattributes representing the position of a moving object S are explicitly updated at time t_1 and the tuple $< S, begin, end >$ referring to S is updated accordingly (i.e., the tuple still satisfies the corresponding query). As far as the *begin* time of the tuple is concerned, there are two possible cases:

Case 1. $t_1 \leq begin$

Case 2. $t_1 > begin$

In the first case, a retransmission of the tuple to the corresponding MH is necessary only if the tuple was previously transmitted to the MH. In the second case, a retransmission is mandatory because the tuple must have been transmitted to the MH by the time *begin*.

We want to make it clear that tuple transmission approaches may handle Case 1 in different ways because it is possible to transmit a tuple at any time $t \leq begin$. In contrast, retransmission at the time of update cannot be avoided with any approach in Case 2. Therefore, from now on we limit the scope of the retransmissions to exclude the ones that are due to an explicit update at $t_1 > begin$.

In order to minimize the control message overhead, all tuples to be transmitted to the MH should be gathered and form a single message. This means that all tuples in the answer set are transmitted at any time before the *begin* time of the tuple with the earliest *begin*. On the other hand, this strategy causes retransmission of some tuples to the MH in case of an explicit update. In order to minimize the probability of retransmission of a tuple, the tuple should be transmitted by its *begin* time. However, in the worst case this strategy leads to a situation where each tuple is appended to a control message. It is clear that the efforts for reducing the control message overhead increases the retransmission overhead and vice versa.

Tuple transmission time is critical especially for the applications where message transmission service is charged a fixed amount of money per byte basis. For example, RAM Mobile Data Corporation charges a minimum of 4 cents per message, with the exact cost depending on the size of the message [WSCY99]. Given a set of tuples as the answer to a CQ, different tuple transmission approaches produce bills with different amounts.

Underlying tuple transmission approach also affects the duration the MH operates in doze (energy saving) mode. The number of transmissions and the total time the MH spends listening to the communication channel must be minimized in order to minimize the energy spent by the MH. Energy preservation is critical because MHs have limited battery capacity, two or three hours under normal use, which is expected to increase only 20% over the next 10 years [IB94, PB94]

Given the same set of tuples as the answer to a CQ, tuple transmission strategies may differ in their ability to support the stand-alone working capability of an MH in case of disconnection. That is, when an MH is disconnected after receiving a number of tuples that are in the answer set of an issued CQ, it can continue displaying the received tuples during the disconnection period in the stand-alone mode (although the updates cannot be transmitted to it). The performance of tuple transmission approaches in terms of supporting the above ability may also be critical in some applications (e.g., in a battlefield).

4 Tuple Transmission Approaches

In this section, we present the methods that are used to determine the transmission time of tuples in the answer set of a location-dependent CQ issued by an MH. We also discuss the benefits and drawbacks of the approaches in terms of control message overhead, tuple retransmission overhead, and the handling of disconnection behavior.

4.1 Immediate Transmission (IT) Approach

According to the IT approach presented in [SWCD97, SWCD98], all the tuples that belong to the answer set of a CQ issued by an MH are transmitted at once at the time the query processing is finished. Upon receiving the answer set, the MH displays them on the screen accordingly. This approach has the following characteristics:

1. It minimizes the control message overhead. All tuples are gathered in a single message which also means a single control message.
2. When a tuple is changed due to an explicit update of an object after the query is processed, it has to be retransmitted.
3. In case the MH disconnects after sometime it has received the answer set of its query, it has the whole answer set.

4.2 Delayed Transmission (DT) Approach

According to the DT approach proposed in [SWCD97, SWCD98], a tuple $< S, begin, end >$ is transmitted to the MH at time *begin*. Upon receiving a tuple, the MH immediately displays it on the screen. This approach has the following characteristics:

1. It maximizes the control message overhead. Each tuple is appended to a control message and then transmitted.
2. The probability that a tuple has to be retransmitted in case of an explicit update to a database object, is minimized.
3. In case the MH disconnects after sometime it has started to receive the tuples in the answer of its CQ, it has the partial answer set.

4.3 Periodic Transmission (PT) Approach

PT is an intermediate approach lying between IT and DT. According to this approach, at each w time units, all the tuples $< S, begin, end >$ satisfying the condition $t \leq begin < t + w$ where t is the current time, are transmitted to the MH. We call w the window size which specifies the time interval containing the *begin* time of the tuples to be transmitted. This approach has the following characteristics:

1. The control message overhead is less than that of the DT approach but greater than that of the IT approach.
2. The probability that a tuple has to be retransmitted in case of an explicit update to a database object is less than it is in the IT approach but greater than it is in the DT approach.
3. In case the MH disconnects after sometime it has started to receive the tuples in the answer of its query, it has the partial answer set.

4.4 Adaptive Periodic Transmission (APT) Approach

The PT approach maintains a constant window size (w) for determining the tuple transmission times. The value of w affects both the control message overhead and the retransmission overhead. Large values of w reduces the control message overhead while increasing the retransmission overhead. Likewise, small values of w reduces the retransmission overhead while increasing the control message overhead.

Data update rate and the resulting overhead due to the retransmission of the updated tuples might vary during the execution of a mobile system. It might be appropriate to have a large w value in order to reduce the control message overhead when updates to the database objects are rare. Similarly, it might be appropriate to have a small w value in order to reduce the retransmission overhead when the updates are frequent. Taking into account the above facts, the APT approach adjusts w by evaluating the information about the relative overheads due to control messages and retransmissions. The period of adjustment of w is called the *evaluation period* of the window size.

The control message overhead is specified by the number of control message bits transmitted with the original tuples (excluding updated tuples) in the answer set of a CQ. The retransmission overhead is specified by the number of bits transmitted as the retransmission messages which consist of the updated tuples and their control messages. We capture the information about these two overheads in a parameter called *overhead ratio* that can be defined as follows:

Definition 4.1 *The overhead ratio V_i during the i^{th} evaluation period is the ratio of control message overhead C_i over retransmission overhead R_i during that period. It is specified by the formula*

$$V_i = \frac{C_i}{R_i}$$

APT uses the *overhead ratio* as a measure to evaluate the performance with w for the last evaluation period. Comparing the values of the *overhead ratios* for the last two evaluation periods, APT decides how to adjust w for the next evaluation period. At the i^{th} evaluation, the window size is adjusted by using the following formula:

$$D_i = V_i - V_{i-1}$$

- $D_i > 0$ means that the control message overhead relative to the retransmission overhead during the i^{th} evaluation period is higher when compared to the $(i-1)^{th}$ evaluation period. So, the window size should be increased to reduce the control message overhead.

- $D_i < 0$ means that the retransmission overhead relative to the control message overhead during the i^{th} evaluation period is higher when compared to the $(i-1)^{th}$ evaluation period. So, the window size should be decreased to reduce the retransmission overhead.

Formally,

$$w = \begin{cases} w + \epsilon & \text{if } D_i > 0 \\ w - \epsilon & \text{if } D_i < 0 \\ w & \text{otherwise} \end{cases}$$

It can be easily confirmed that the probability that an updated tuple will be retransmitted depends on the value of w. Large values of w increase the retransmission probability while the small values of w decrease that probability. Similarly, the value of w also affects the availability of the tuples in the answer set in case of disconnections. Large values of w makes it possible for the MH to have more tuples compared to the case with small values of w.

4.5 Mixed Transmission (MT) Approach

APT presented above maintains a single window size for the whole database. This approach does suffer from the following shortcoming. The database may consist of a mixture of frequently changing objects (e.g., moving objects like cars) and rarely changing objects (e.g., motels). It may happen in this database system that w cannot be increased because of the heavy retransmission overhead caused by frequently changing objects. On the other hand, small values of w are not appropriate for rarely changing objects since this would increase the control message overhead although this overhead is supposed to be minimal for such objects.

In order to handle the above problem, the MT approach partitions the database into two disjoint sets: one consisting of "hot" database objects (i.e., frequently changing) and the other consisting of "cold" database objects (i.e., rarely changing). This approach transmits the tuples referring the "cold" database objects as in the IT approach and the tuples referring the "hot" database objects as in the APT approach. Therefore, the control message overhead and the retransmission overhead is mostly limited to those associated with tuples referring to "hot" database objects. Consequently, we modify the definition of the overhead ratio to cover only "hot" database objects (O_h).

Definition 4.2 *The overhead ratio $V_i(O_h)$ for "hot" database objects during the i^{th} evaluation period is the ratio of control message overhead $C_i(O_h)$ over retransmission overhead $R_i(O_h)$ during that period. It is specified by the formula*

$$V_i(O_h) = \frac{C_i(O_h)}{R_i(O_h)}$$

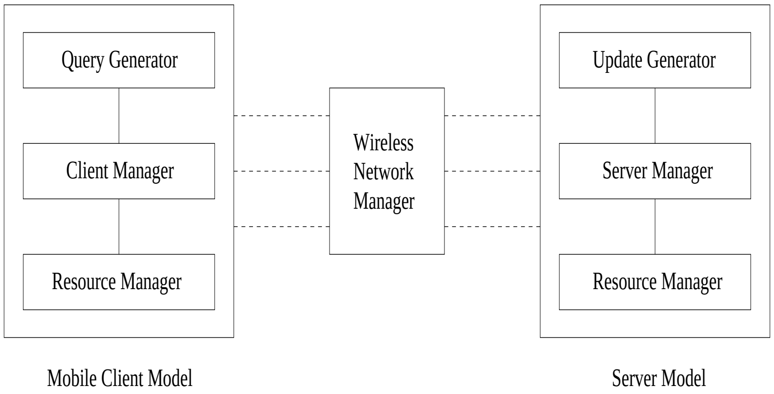

Figure 2: The Simulation Model.

MT decides how to adjust w for the next evaluation period using the following equation.

$$D_i(O_h) = V_i(O_h) - V_{i-1}(O_h)$$

Formally,

$$w(O_h) = \begin{cases} w(O_h) + \epsilon & \text{if } D_i(O_h) > 0 \\ w(O_h) - \epsilon & \text{if } D_i(O_h) < 0 \\ w(O_h) & \text{otherwise} \end{cases}$$

Thus, the control message overhead for the tuples referring to "cold" objects is minimized by making use of the fact that those objects are rarely updated. The retransmission and control message overheads for the tuples referring to "hot" objects is reduced by transmitting these tuples as in APT.

The availability of the tuples in the answer set of a CQ in case of disconnection can be considered separately for the tuples referring to "cold" and "hot" objects. All the tuples referring to "cold" objects will be available to the MH but the availability of the tuples referring to "hot" objects will depend on the current value of $w(O_h)$.

5 Simulation Model

We have designed a simulation model to compare the performance of tuple transmission approaches IT, DT, PT, APT, and MT under different settings of environmental parameters such as the data update rate and disconnection period. Our simulation model is based on the performance models proposed in previous related works such as [BJ96, LS97]. These models have been extended to support modeling of processing location-dependent CQs.

As shown in Figure 2, the simulation model consists of three basic components:

1. Mobile Client Model
2. Wireless Network Manager
3. Server Model

In the following subsections, we describe each component in detail.

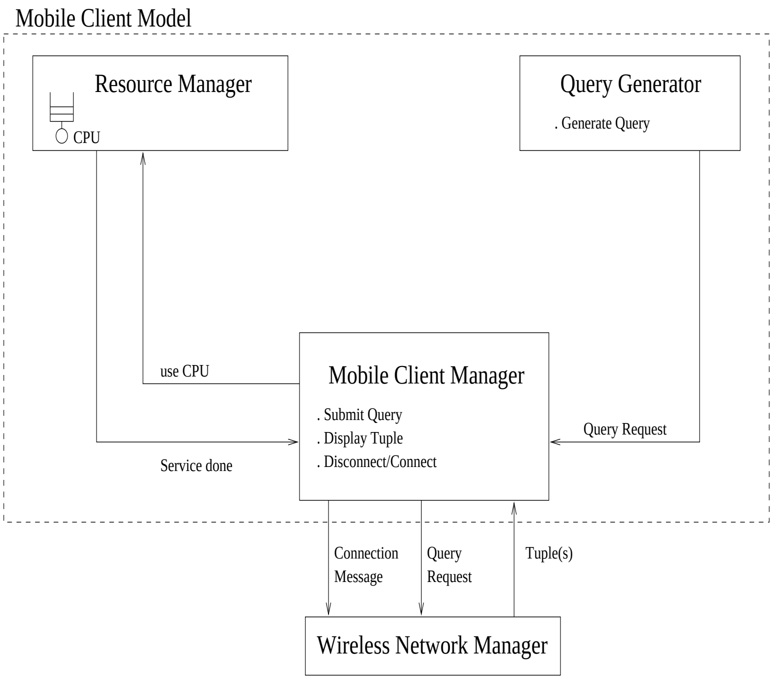

Figure 3: Mobile Client Model.

5.1 Mobile Client Model

Each mobile client is composed of three modules as shown in Figure 3: a Resource Manager which models the client CPU for handling the query results, a Query Generator which generates the query requests, and a Client Manager which processes the query requests and passes them to the server, models the disconnection operation, and receives and processes the tuples transmitted from the server.

Client queries are submitted from an MH to the server to be processed and a message (messages) containing the tuples that form the answer to the query is (are) transmitted back to the MH. The messages containing the tuples are processed by the MH and the tuples are displayed on the screen of the MH accordingly.

Table 1 lists the parameters of the Mobile Client Model. Each of the *NumMobileHosts* MHs generates a single stream of CQ with size *QueryRequestSize*. The arrival of a new query is separated from the completion of the previous query by an exponentially distributed think time with a mean of *ThinkTime*. The query duration, during which the query should be processed, is chosen randomly by the Query Generator and has the maximum value *MaxQueryDuration*. The probability that an MH will enter into a disconnection mode after issuing a query is determined by using *DisconnectProb* and the time delay before the disconnection is chosen uniformly within the execution time of the issued query. The duration that the MH will stay disconnected is chosen from an exponential distribution with a mean of *DisconnectTime*. When the MH later reconnects to the network, it sends a message having size *ConnectMsgSize* to inform the Server Manager.

No I/O time is modeled in the Resource Manager Module since we assume that the buffer pools of MHs are large enough to hold all the tuples received in response to an issued CQ. Each MH has a single CPU and the CPU time for processing a message per byte basis is determined by *ClientMsgTime*.

Parameter	Meaning
$NumMobileHosts$	Number of MHs
$QueryRequestSize$	Size of a CQ submitted by an MH
$ThinkTime$	Mean think time between queries in connect mode
$DisconnectTime$	Mean disconnect time
$MaxQueryDuration$	Maximum query duration
$DisconnectProb$	The probability that the MH will be disconnected after issuing a query
$ClientMsgTime$	CPU time to process a message per byte basis
$ConnectMsgSize$	Size of a connection indication message

Table 1: Mobile Client Model Parameters

Parameter	Meaning
$NetworkBandwidth$	Wireless network bandwidth
$ControlMsgSize$	Size of a control message on the wireless network

Table 2: Wireless Network Manager Parameters

5.2 Wireless Network Manager

Table 2 lists the parameters associated with the Wireless Network Manager. The Wireless Network Manager component assumes that all messages are of equal priority that will be served on a First-Come First-Served (FCFS) basis with a service rate of *NetworkBandwidth*. When a message is to be transmitted, it is appended to a control message having size *ControlMsgSize*.

When the Wireless Network Manager finds out (i.e., while sending a message to an MH) that an MH is disconnected, it informs the Server Manager about the disconnection so that the transmission of the tuples to the MH can be paused until the MH reconnects to the network.

5.3 Server Model

The central server model has three modules as shown in Figure 4: a Resource Manager Module which models the server CPU time for query and update processing, an Update Generator which generates update requests, and a Server Manager Module which coordinates the query requests from MHs and update requests from the Update Generator.

The input parameters for the Server Model are listed in Table 3. The Resource Manager Module that models the database and physical resources of the system has *NumCPU* CPUs. The CPU time for processing a query and an update are specified by the parameters *ServerQueryTime* and *ServerUpdateTime*, respectively. All query and update requests are processed with the same priority on an FCFS basis. The database is modeled as a collection of *DatabaseSize* objects each with size *ObjectSize*. No I/O operation is modeled since we assume that the buffer pool in the server is large enough to hold the entire database.

When a CQ is issued by an MH, it is processed by the Server Manager and the set of tuples satisfying the query are determined. The number of tuples in the answer set of a CQ is determined randomly using a maximum value of *MaxNumTuple*. The size of each tuple is specified by *TupleSize*.

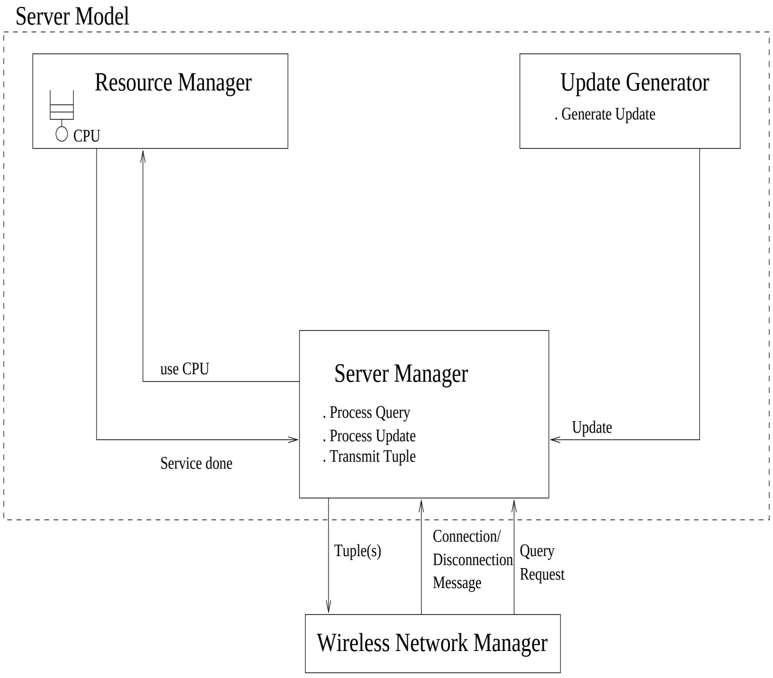

Figure 4: Server Model.

Parameter	**Meaning**
$NumCPU$	Number of CPUs
$ServerQueryTime$	Service time for a query in the data server
$ServerUpdateTime$	Service time for an update in the data server
$DatabaseSize$	Number of objects in the database
$ObjectSize$	Size of a database object
$MaxNumTuple$	Maximum number of tuples that can satisfy a CQ
$TupleSize$	Size of a tuple
$EvaluationPeriod$	Time period to adjust the window size
$WindowSize$	Initial window size
ϵ	Threshold value for the adjustment of the window size
$UpdateArrTime$	Mean interarrival time between updates
$HotUpdateBounds$	Data object bounds of hot update range
$ColdUpdateBounds$	Data object bounds of cold update range
$HotUpdateProb$	Probability that an update will be applied to a "hot" object
$HotQueryProb$	Probability that a tuple will refer to a "hot" object
$HotRemoveProb$	Probability that an updated tuple referring to a "hot" object will be removed from the corresponding answer set
$ColdRemoveProb$	Probability that an updated tuple referring to a "cold" object will be removed from the corresponding answer set

Table 3: Server Model Parameters

The Server Manager also decides when and which tuples should be transmitted to the MH depending on the underlying tuple transmission approach (i.e., one of the IT, DT, PT, APT, MT approaches). The window size is also adjusted by the Server Manager for the APT and MT approaches. The window size is evaluated and adjusted every *EvaluationPeriod* time units. Depending on the underlying policy, the window size is incremented or decremented by a small integer ϵ. We assume that the time needed to evaluate and adjust the window size is negligible and therefore do not take it into account in our model.

At the server, a single stream of updates is generated. These updates are separated by an exponentially distributed update interarrival time with a mean of *UpdateArrTime*. Our model can specify different update and query patterns. For the central data server, *HotUpdateBounds* and *ColdUpdateBounds* parameters are used to specify the "hot" and "cold" regions of the database respectively for update requests. *HotUpdateProb* and $HotQueryProb$ specify the probability that an update will be applied to a database object in the "hot" database region and a tuple in the answer set of a CQ will refer to a "hot" object, respectively. $HotRemoveProb$ and $ColdRemoveProb$ specify the probability that an tuple referring to a "hot" object and a "cold" object will be removed from the answer set, respectively.

When a database object is explicitly updated, we assume that all the tuples in the answer set of every CQ that refer to the updated object, are changed. For simplicity, we ignore the possibility that the updated object may satisfy new queries that it did not satisfy before. We also assume that the attributes representing the position of the MH that issued the query do not change until the query processing is completed; because, such a change results in reevaluation of the query and in this study we focus on retransmissions rather than reevaluations. However, this assumption does not mean that the querying MH is a stationary object.

When a tuple in an answer set is updated, it is immediately retransmitted to the corresponding MH. The original tuple (before update) may be in use at the MH at the time of the update and MH must be informed about the update to the tuple immediately so that it can invalidate the original tuple.

When the Wireless Network Manager detects that an MH is disconnected, it informs the Server Manager to pause transmitting tuples to the MH until it reconnects to the network. When the MH reconnects, the Server Manager resumes transmitting the valid tuples (tuples with *end* time $\leq$ current time) to the MH.

6 Experiments and Results

In this section, we present the performance results for the tuple transmission approaches for CQs that we discussed in Section 4. A number of simulation experiments have been conducted to study the behavior of different tuple transmission algorithms under various settings of data update rates, maximum query duration, disconnection period and update/query patterns.

Experiments were designed to evaluate the relative performance of the algorithms in terms of the communication overhead imposed on the wireless network and the availability of tuples in case of disconnections. All experiments were performed on SunSparc Workstations running SUNOS, using the CSIM [Sch92] simulation package. Each experiment was run until a total of 5000 CQs are completed. Each experiment was repeated 30 times with different *seeds* in order to obtain a statistically significant sample of CQs.

The primary performance metric used in this study is the average number of bits transmitted to an MH in response to a CQ. The number of bits transmitted for a CQ is computed by summing up the total number of bits transmitted as tuples and control messages in response to a CQ. Another

Parameter	Value
$NumMobileHosts$	100
$ThinkTime$	1000 s
$MaxQueryDuration$	varied from 240 s to 360 s
$QueryRequestSize$	256 bytes
$DisconnectProb$	1/10
$DisconnectTime$	varied from 50 s to 1000 s
$ConnectMessageSize$	4 bytes
$ClientMsgTime$	0.0001 s/byte
$NetworkBandwidth$	19200 bps
$ControlMsgSize$	256 bytes
$DatabaseSize$	1000 objects
$ObjectSize$	256 bytes
$TupleSize$	264 bytes
$UpdateArrTime$	varied from 1 s to 5 s
$HotUpdateBounds$	All the database
$NumCPU$	1
$ServerQueryTime$	0.01 s
$ServerUpdateTime$	0.02 s
$MaxNumTuple$	40 tuples
$HotRemoveProb$	0.01
$WindowSize$	180 s
$EvaluationPeriod$	500 s
ϵ	1 s

Table 4: Parameter Settings for the Base Experiment

metric used is the *availability* of tuples in the answer set of a CQ in case of a disconnection. The availability of tuples in case of a disconnection is specified as the ratio of the number of tuples received by the MH prior to disconnection over the total number of tuples that would have been received by the end of the disconnection period if the MH had been connected to the network.

The values of the simulation parameters were chosen so as to be comparable to the related simulation studies such as [BJ96, LS97]. Since there is no data available for modeling the tuples in the answer set of a CQ, we are concerned here with performance trends rather than with exact performance predictions. Table 4 provides the values of the simulation parameters which are common to all experiments except where otherwise specified.

6.1 The Base Experiment

We first examine the performance results of the proposed tuple transmission approaches under varying data update rates by setting $MaxQueryDuration$ and $DisconnectTime$ to 300 seconds. Performance of the MT approach is not examined in this experiment because the behavior of MT is the same as that of APT since all the database objects are assumed to be “hot”. Figures 5 through 8 show the performance results obtained.

As illustrated in Figure 5, DT performs the worst among all tuple transmission approaches in

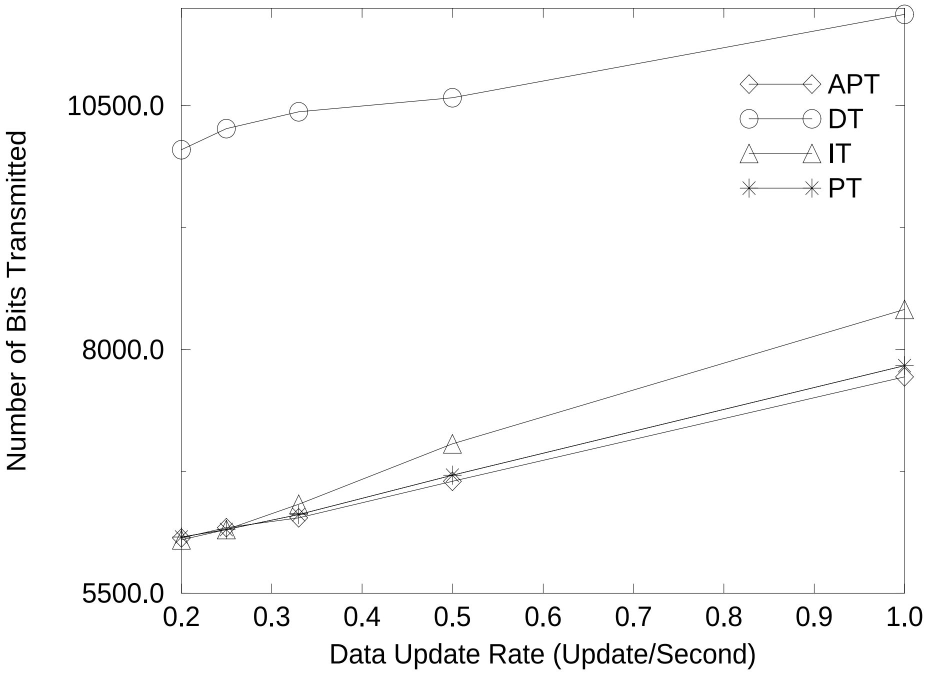

Figure 5: Average Number of Bits Transmitted vs Data Update Rate.

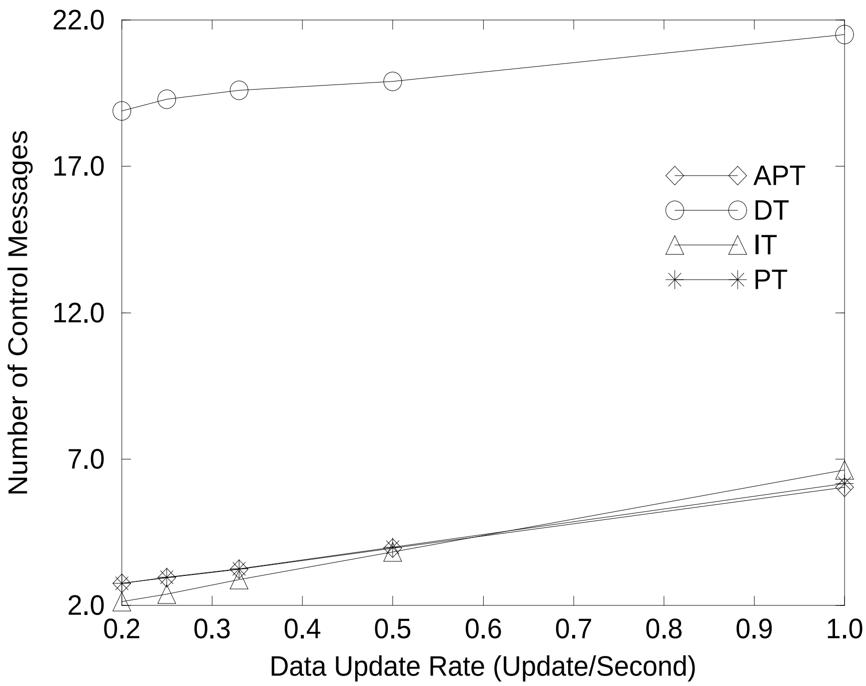

Figure 6: Average Number of Control Messages vs Data Update Rate.

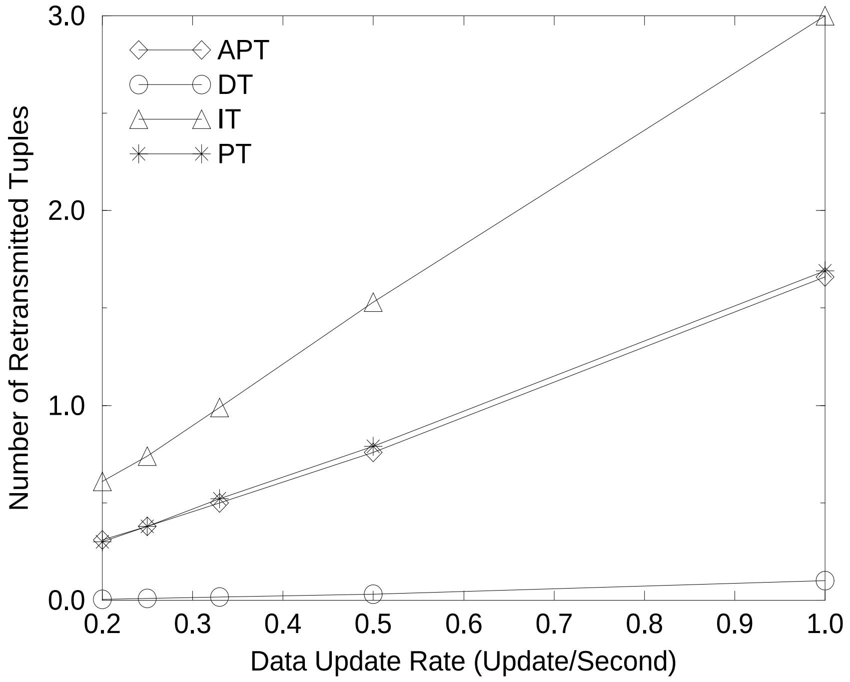

Figure 7: Average Number of Retransmitted Tuples per CQ vs Data Update Rate.

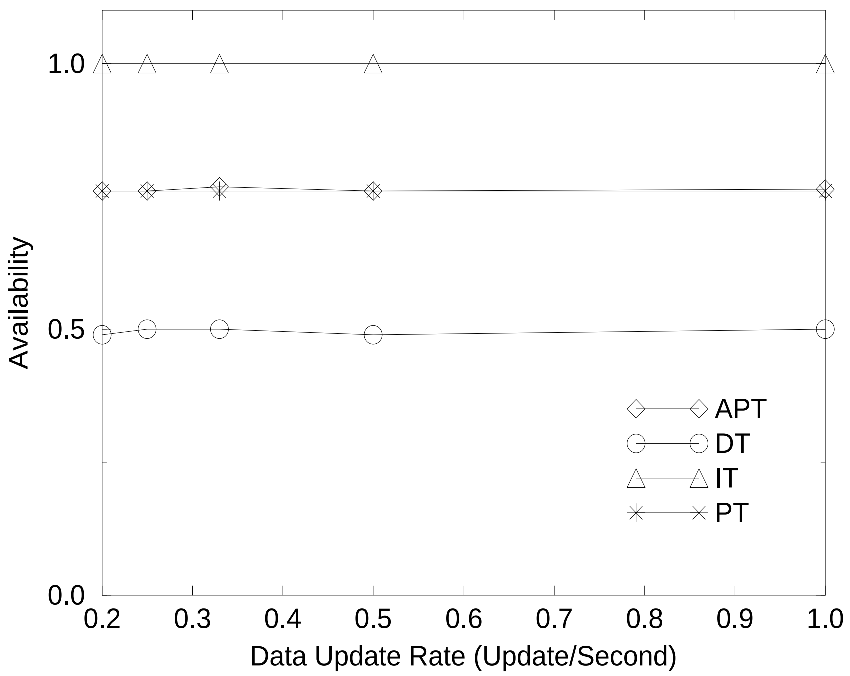

Figure 8: Availability of Tuples vs Data Update Rate.

terms of the average number of bits transmitted in response to a CQ. This result is due to involving the highest control message overhead caused by the transmission of each tuple separately as shown in Figure 6. At low data update rates the performance results of IT, PT, and APT are close to each other. Transmitting all the tuples at once or transmitting them periodically with $w = 180$ seconds in PT and APT, does not make much difference in terms of the control message overhead. As Figure 6 shows, the control message overhead involved with IT is close to that of PT and APT at low data update rates.

As the data update rate is increased, all the curves start to move upward due to the increasing retransmission overhead as shown in Figure 7. Furthermore, the performance difference between IT, PT, and APT in terms of the average number of bits transmitted becomes apparent with the high data update rates. PT and APT approaches have an important benefit over IT in terms of the retransmission overhead. Another observation is that the periodic adjustment of w according to the criterion we have formulated in APT provides some improvement over the performance of PT.

The reader may notice from Figure 7 that the number of retransmissions per CQ may not always be zero with the DT approach. This may seem contradictory as we have limited the scope of retransmissions to those of Case 2 (in Section 3) which excludes the retransmissions due to an update after the *begin* time of a tuple. However, when a tuple is changed due to an explicit update to the database, it is immediately retransmitted. Therefore, Case 2 retransmissions are also possible with DT.

As we have discussed before, supporting the ability for an MH to work in the stand-alone mode in case of disconnections can be very important in some applications. Figure 8 shows the availability of tuples in the answer set of a CQ in case of disconnections. As expected, IT has the highest availability since this approach transmits all the tuples together as soon as they are determined. The performance results of PT and APT in terms of availability are nearly the same. DT is the worst approach in supporting the stand-alone working ability since the transmission of a tuple is delayed until its *begin* time. We also observe that increasing data update rate does not have an impact on the performance of any approach in terms of availability.

6.1.1 Evaluation of the Impact of Query Duration

In this experiment, we examine the performance in terms of the average number of bits in response to a CQ for the tuple transmission approaches as the maximum query duration is varied while setting $UpdateArrTime$ to 1. Increasing the maximum query duration increases the probability that a tuple will be updated therefore the probability that it will be retransmitted as shown in Figure 9. This increase is dramatic in IT, since IT is the most prone approach to retransmissions.

The average number of bits transmitted in response to an issued CQ increases as the query duration increases as shown in Figure 10. As compared to Figure 5, the relative performance of the approaches does not change and DT is still the worst performing approach. The performance difference between IT and the other two approaches PT, APT becomes more apparent as the query duration is increased.

6.1.2 Evaluation of the Impact of Disconnection Period

In this section, we investigate the impact of the time period an MH stays disconnected after issuing a CQ, on the availability of the tuples. $MaxQueryDuration$ is set to 300 seconds. As expected, the availability ratio of tuples in IT is always 1 independent of the disconnection period. The availability of tuples in all the other approaches decreases up to a certain point as the disconnection period

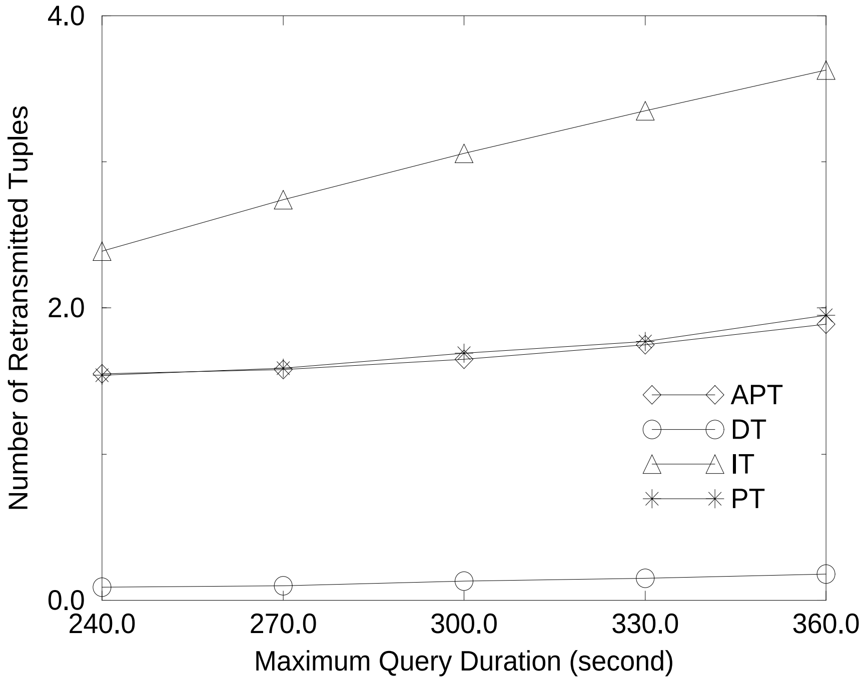

Figure 9: Average Number of Retransmitted Tuples vs Maximum Query Duration.

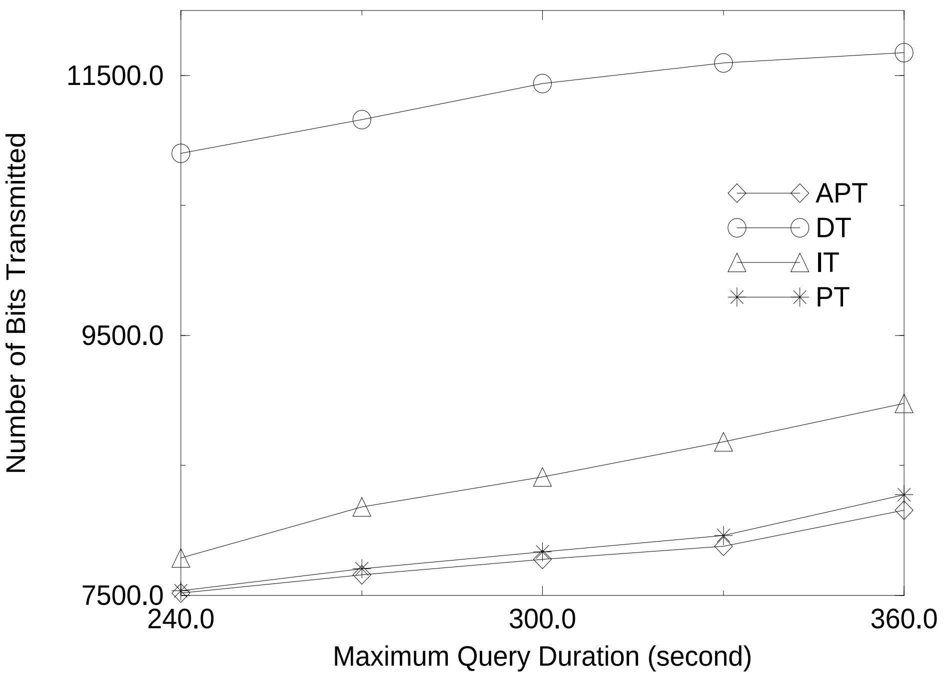

Figure 10: Average Number of Bits Transmitted vs Maximum Query Duration.

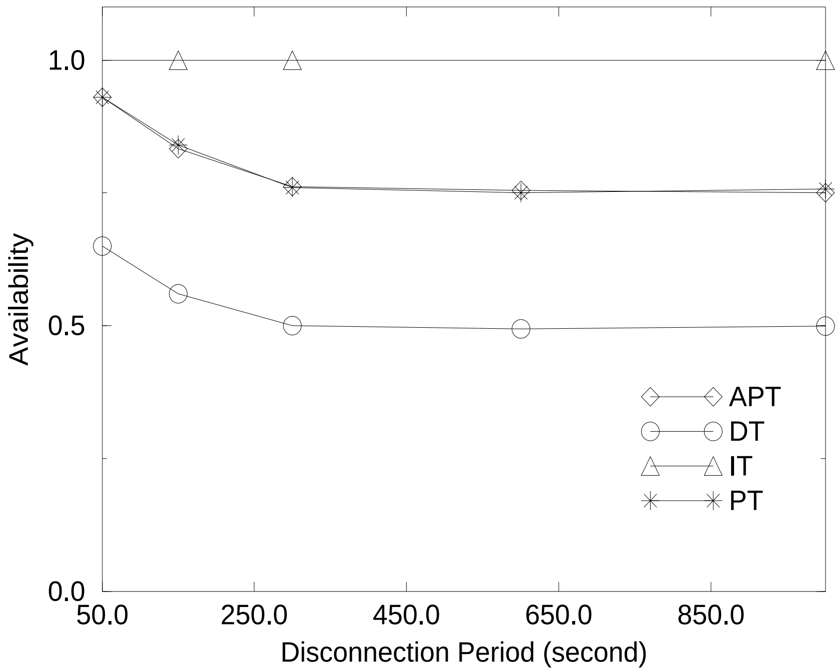

Figure 11: Availability of Tuples vs Disconnection Period.

increases as shown in Figure 11. After that particular point, the availability of tuples remains constant. This is a reasonable result because the availability of tuples is the same for disconnection periods longer than the query duration. Suppose that an MH issued a CQ with duration 200 seconds and is disconnected. The availability of tuples will be the same for disconnections lasting more than 200 seconds.

6.2 Evaluation of the Impact of Hotspots

As we have discussed earlier, the database may consist of a mixture of frequently changing and rarely changing objects. We set up an experiment in order to observe the performance of tuple transmission approaches in case there exists a hotspot in the database. Table 5 lists the values of the related parameters used in this experiment. According to the new update pattern in this experiment, 80% of the updates are applied to 20% of the database and the object a tuple will refer to is uniformly chosen from the database.

The comparison of Figures 5 and 12 shows that the relative performance of IT, PT, and APT does not change in terms of the average number of bits transmitted in response to a CQ. MT performs the best in terms of reducing the communication overhead. Figures 13 and 14 show that transmitting the tuples referring to "hot" objects as in APT and the tuples referring to "cold" objects as in IT reduces the communication overhead and offers the best performance among all the tuple transmission approaches we presented in case of a hotspot in the database.

MT performs closer to the performance of IT than that of DT, PT, and APT in terms of the availability of tuples in case of a disconnection (Figure 15). Since the tuples forming the answer set are chosen uniformly from the database, 80% of the tuples in the answer set of a CQ are

Parameter	Value
$HotUpdateBounds$	1-200
$ColdUpdateBounds$	201-1000
$HotRemoveProb$	0.01
$ColdRemoveProb$	0.04
$HotUpdateProb$	0.8
$HotQueryProb$	0.2
$WindowSize$	180 s (for PT & APT), 150 s (for MT)

Table 5: Parameter Settings

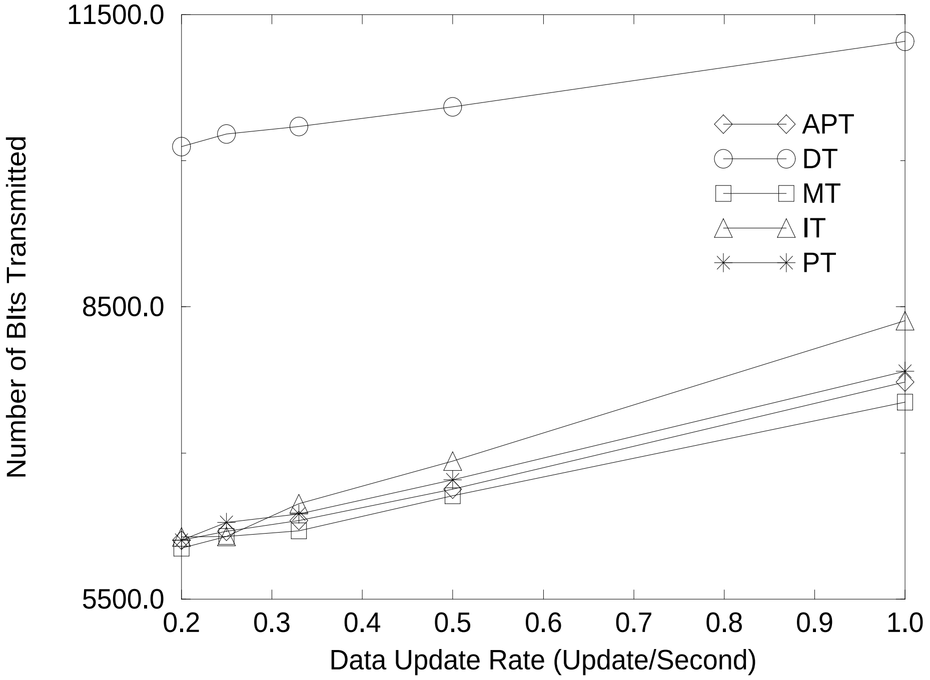

Figure 12: Average Number of Bits Transmitted vs Data Update Rate.

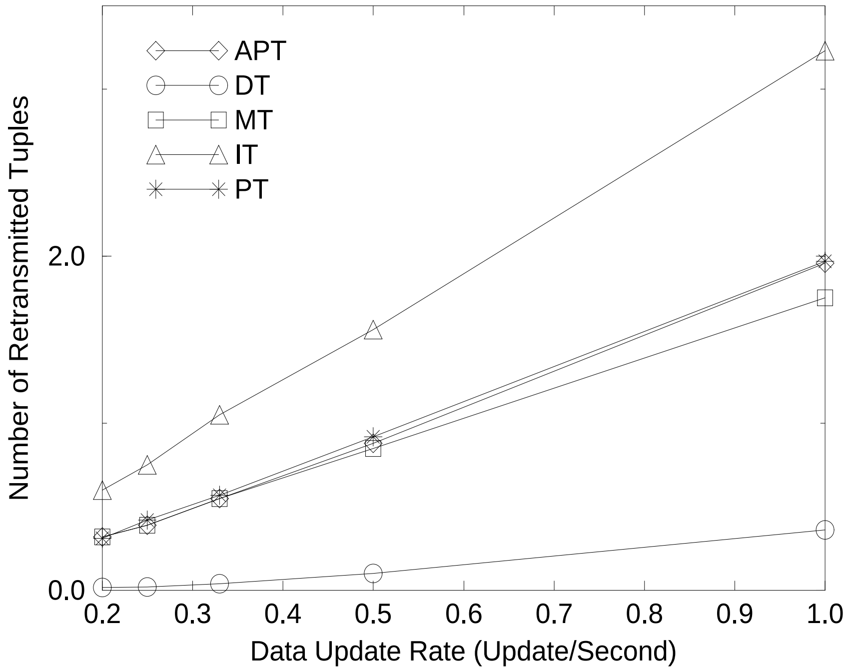

Figure 13: Average Number of Retransmitted Tuples vs Data Update Rate.

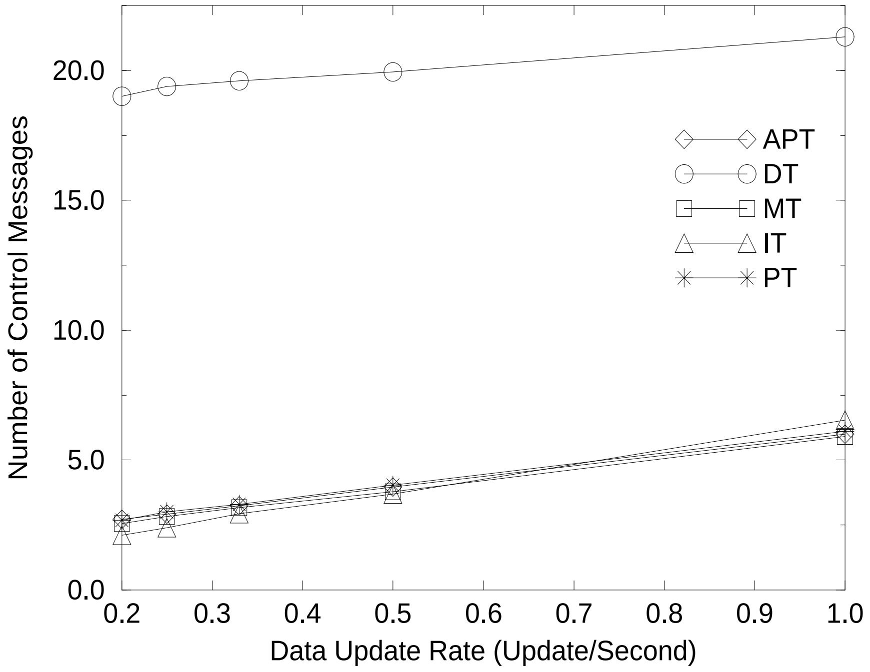

Figure 14: Average Number of Control Messages vs Data Update Rate.

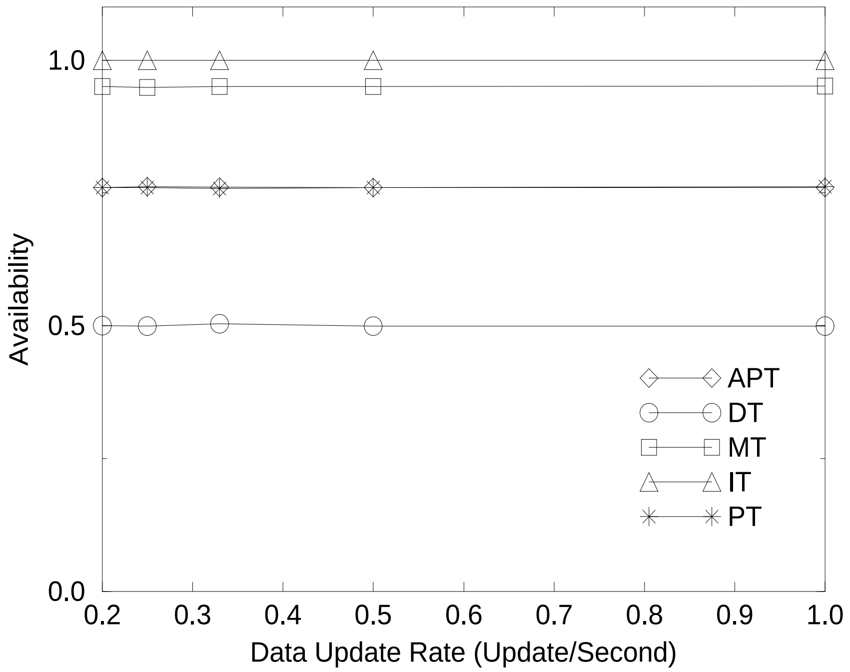

Figure 15: Availability of Tuples vs Data Update Rate.

supposed to be those referring to "cold" objects. MT transmits the tuples referring to "cold" objects immediately at the time they are determined. Therefore, those tuples are always available to the MH in case of a disconnection. Tuples referring to "hot" objects are partially available in case of a disconnection and the above combination leads to a higher availability of tuples than those of PT and APT.

6.3 Evaluation of the Impact of Query Hotspots

In a mobile database system, it is also possible that some objects are accessed more frequently than the others. We examine the impact of such a situation in this experiment by setting $HotQueryProb$ to 0.8. It means that the hotspot of updates which is bounded by $HotUpdateBounds$ is now a hotspot in terms of access workload too.

As shown in Figure 16, the new query pattern does not change the relative performance of the tuple transmission approaches we propose in terms of the average number of bits transmitted in response to a CQ. However, the performance difference between MT and the other approaches becomes more apparent with the high data update rates. In this experiment, the answer to a CQ consists mostly of "hot" objects. Therefore, retransmissions are more frequent compared to the previous experiments (see Figure 17). Heavy retransmission overhead makes IT perform worse than DT with an update rate of 1 update per second.

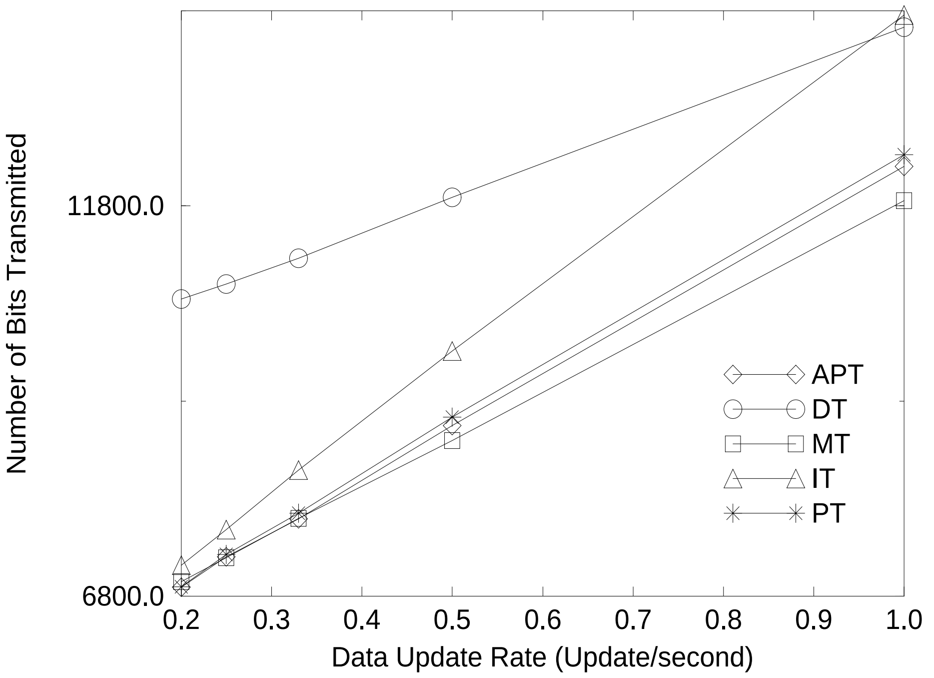

Figure 16: Average Number of Bits Transmitted vs Data Update Rate.

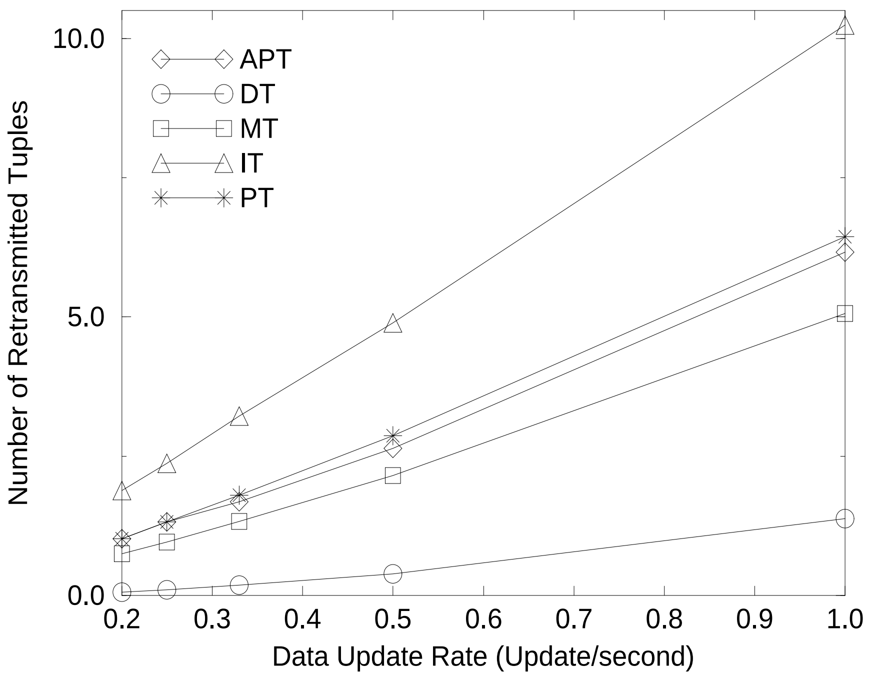

Figure 17: Average Number of Retransmitted Tuples per CQ vs Data Update Rate.

7 Conclusions

In this paper, we present new approaches for the transmission of the answer of a location-dependent continuous query (CQ) in mobile computing systems. We also evaluate the performance of proposed approaches in terms of the communication overhead imposed on the wireless network in response to a CQ and the availability of tuples in the answer set of a CQ in case of a disconnection. The experiments conducted demonstrate that the choice between proposed transmission approaches depends on the answer to the following question: How critical is it for the mobile hosts (MHs) to be able to continue displaying tuples on the screen in case of disconnections? Once the tradeoff between the high availability of tuples in case of disconnections and the low communication overhead is determined, the appropriate tuple transmission approach would be one of the approaches *Adaptive Periodic Transmission* and *Mixed Transmission* for low communication overhead, or the *Immediate Transmission* approach otherwise.

A possible direction of future research is to extend our simulation model to support caching of database objects at MHs. Frequently accessed database objects can be broadcast by the data server to MHs and stored in caches of MHs. In such a system, the data server processes the CQs submitted by an MH and transmissions will be limited to only those tuples referring to objects that are not cached at the MH. For the tuples referring to objects that are cached at the MH, only *begin* and *end* time attributes, and the object id need to be transmitted to the MH. Such a caching mechanism can reduce the communication overhead significantly especially if database objects are large. Adapting a caching strategy pops up new issues such as cache invalidation mechanisms and the period of the broadcast cycle which are subject to further investigation.

References

[BJ96] O. Bukhres and J. Jing. Analysis of adaptive caching algorithms in mobile environments. *Information Sciences*, pages 1–27, 1996.

[Chr93] P. K. Chrysanthis. Transaction processing in mobile computing environment. In *Proceedings of IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 77–83, Princeton, New Jersey, October 1993.

[DHB97] M. H. Dunham, A. Helal, and S. Balakrishnan. A mobile transaction model that captures both data and movement behavior. *MONET*, 2(2):115–127, 1997.

[FZ94] G. H. Forman and J. Zahorjen. The challenges of mobile computing. *IEEE Computer*, 27(6), April 1994.

[IB93] T. Imielinski and B. R. Badrinath. Data management for mobile computing. *ACM SIGMOD RECORD*, 22(1):34–39, 1993.

[IB94] T. Imielinski and B. R. Badrinath. Mobile wireless computing: Challenges in data management. *Communication of ACM*, 37(10):18–28, October 1994.

[KU99a] E. Kayan and O. Ulusoy. An evaluation of real-time transaction management issues in mobile database systems. *The Computer Journal*, 42(6), 1999.

[KU99b] E. Kayan and O. Ulusoy. Real-time transaction management in mobile computing systems. In *Proceedings of the 6th International Conference on Database Systems for Advanced Applications (DASFAA'99)*, pages 127–134, Hsinchu, Taiwan, April 1999.

[LS97] H. V. Leong and A. Si. Database caching over the air storage. *The Computer Journal*, 40(7), 1997.

[PB93] E. Pitoura and B. Bhargava. Dealing with mobility: Issues and research challenges. Technical Report TR-97-070, Department of Computer Sciences, Purdue University, 1993.

[PB94] E. Pitoura and B. Bhargava. Building information system for mobile environments. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, pages 371–378, Guithesburg, MD, November 1994.

[Sch92] H. Schwetman. *CSIM User's Guide*. MCC Corporation, 1992.

[SWCD97] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of the 13th International Conference on Data Engineering*, pages 422–432, Birmingham, UK, April 1997.

[SWCD98] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. *Querying the Uncertain Position of Moving Objects*, chapter Temporal Databases: Research and Practice, pages 310–337. Lecture Notes in Computer Science (Springer Verlag), 1998.

[SWD+96] A. P. Sistla, O. Wolfson, S. Dao, K. Narayonan, and R. Raj. An architecture for consumer-oriented online database services. In *Proceedings of the 6th International Workshop on Research Issues in Data Engineering: Interoperability of Nontraditional Database Systems*, New Orleans, LA, February 1996.

[TUW98] J. Tayeb, O. Ulusoy, and O. Wolfson. A quadtree based dynamic attribute indexing method. *The Computer Journal*, 41(3), 1998.

[Ulu98] O. Ulusoy. Real-time data management for mobile computing. In *Proceedings of the 1st International Workshop on Issues and Applications of Database Technology*, pages 233–240, Berlin, Germany, July 1998.

[WC95] G. D. Walborn and P. K. Chrysanthis. Supporting semantics-based transaction processing. In *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pages 31–40, September 1995.

[WSCY99] O. Wolfson, P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases Journal*, 7(3):1–31, 1999.

[WXCJ98] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: Issues and solutions. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 111–122, Capri, Italy, July 1998.