

# Broadcasting Consistent Data to Read-Only Transactions from Mobile Clients<sup>1</sup>

Kam-Yiu Lam, Mei-Wai Au and Edward Chan  
Department of Computer Science  
City University of Hong Kong  
83 Tat Chee Avenue, Kowloon  
email: {cskylam|csedchan}@cityu.edu.hk

## Abstract

*In this paper, we study the data inconsistency problem in data broadcast to mobile transactions. While data items in a mobile computing system are being broadcast, update transactions may install new values for the data items. If the executions of update transactions and broadcast of data items are interleaved without any control, the transactions generated by mobile clients, called mobile transactions, may observe inconsistent data values. In this paper, we propose a new protocol, called Update-First with Order (UFO), for concurrency control between read-only mobile transactions and update transactions. We show that although the protocol is simple, all the schedules are serializable when the UFO protocol is applied. Furthermore, the new protocol possesses many desirable properties for mobile computing systems such as the mobile transactions do not need to set any lock before they read a data item from the “air” and the protocol can be applied to different broadcast algorithms. Its performance has been investigated with extensive simulation experiment. The results show that the protocol can maximize the freshness of the data items provided to mobile transactions and the broadcast overhead is not heavy especially when the arrival rate of the update transactions is not very high.*

Keywords: Data broadcast, concurrency control, mobile computing, transaction processing

## 1 Introduction

Recent advances in mobile communication technology have greatly increased the functionality of mobile information services and have made many mobile computing applications a reality. Innovative applications, such as real-time traffic information and navigation systems, and real-time stock monitoring systems, will no doubt continue to emerge as data-hungry users require instant access to information using their palmtops, wearable computers and notebooks no matter where they are located. This trend is acknowledged in merging standards, such as MPEG-4, which contains features targeted explicitly for low bit rate environments such as mobile networks, and a large variety of mobile multimedia applications can be expected in the near future.

---

<sup>1</sup>The work described in this paper was partially supported by a grant from the Research Grants Council of Hong Kong Special Administrative Region, China [Project No. CityU1097/99E] and a grant from from City University of Hong Kong (Project No. 7001117).

A number of technical hurdles need to be surmounted before these scenarios materialize [1, 10]. Owing to the intrinsic constraints of mobile computing systems, such as asymmetric bandwidth, limited power supply and unreliable communication, the design of an efficient and cost effective mobile computing system poses many challenges [2, 6, 10, 15, 17]. One of the most important issues is efficient dissemination of consistent data to transactions from mobile clients [10, 16].

In recent years, various data dissemination methods have been proposed [2, 4, 5, 8, 12, 13, 21, 25]. Basically, there are two approaches to disseminate data items from a database server, which maintains a database, to transactions from mobile clients (called *mobile transactions*): *on-demand* and *data broadcast*. In the on-demand approach, the data items requesting by a mobile transaction will be sent from the database server on request. This approach is simple but does not scale well with the number of mobile clients. The waiting time for a data item will be very long if there are a large number of mobile transactions waiting for data items [2, 25].

In the data broadcast approach, the database server periodically and continuously broadcasts data items one by one to mobile clients. If there is a mobile transaction waiting for the data item, it will get the data item from the “air” while it is being broadcast [2]. Thus, the cost for data dissemination is independent of client number, resulting in a much more efficient way of using the limited bandwidth. It is therefore quite suitable for disseminating substantial amount of information and data to a large number of mobile clients where bandwidth efficiency is a major concern such as in stock monitoring systems and road traffic information systems.

In broadcasting data items to mobile transactions, two important requirements have to be satisfied. Firstly, the values of the data items have to be consistent. This is a concurrency control problem. However, providing consistent data to mobile transactions is not an easy problem. Due to the limitation of up-link channel bandwidth, it is not a good idea to send data request of a mobile transaction to the database server. In addition, the efficient methods for traditional database systems are not suitable. Secondly, the values of the data items must be up-to-date or at least “close<sup>2</sup>” to the most current values [22, 23, 24]. It addresses the need to maintain the temporal constraints of data since the values of the data items can be highly dynamic. These two issues unfortunately have received much less attention than the design of data broadcast algorithms even though they are also critical in the design of practical data broadcast systems.

In data broadcast, if the update to the database is done concurrently, the mobile transactions may observe inconsistent data values [17, 21]. The updates capture the most up-dated information of the external environment and refresh the data values in the database. Allowing the execution of updates to be interleaved with data broadcast is important in maintaining the “freshness” of the data items (the second requirement) [24]. In doing so, however, the problem of concurrency control must be addressed. Unfortunately, traditional concurrency control protocols such as two phase locking, optimistic method and timestamp ordering are not suitable to mobile computing systems using data broadcast as they require a heavy overhead in setting locks and detecting data

---

<sup>2</sup> The definition of “close” can be application dependent and depends on the dynamic properties of the data

conflicts in a mobile environment [3].

In this paper, we propose a new protocol called *Update-First with Order (UFO)*, which has many important advantages for mobile computing systems. In addition to providing consistent data values to mobile transactions, it also maximizes the currency of data items provided to mobile transactions. Its performance is independent of the number of mobile clients in the system and the additional overhead for providing consistent data values to mobile transactions is low. The performance characteristics of the protocol are studied through extensive simulation experiments and compared with another efficient protocol, multiversion broadcast [18]. The organization of the remaining parts of the paper is as follow. Section 2 reviews the related work on concurrency control in data broadcast. Section 3 defines our transaction models for mobile computing systems with data broadcast. Section 4 discusses the correctness and performance requirements. Section 5 discusses the data inconsistency problems using some examples. Section 6 introduces the UFO protocol and discusses its properties, correctness and implementation. Section 7 examines the overheads of the protocol and performance characteristics by means of a series of simulation experiments. The paper concludes with Section 8, where some indications on how we intend to extend our existing work are briefly described.

## 2 Related Work

Mobile computing has been the subject of much research in recent years [1, 7, 8, 24]. Unfortunately the important issue of concurrency control in data broadcast has not received adequate attention. Traditional concurrency control protocols are not suitable to mobile computing systems due to their heavy overheads for detecting data conflicts in a mobile environment [16]. Owing to the relatively poor quality of services of mobile networks, it is not easy to ensure data consistency and to detect data conflicts in a mobile network. To our knowledge, few studies until now have been done on concurrency control for mobile computing systems using data broadcast. One suggested solution is to relax the consistency requirement. In [16], a two-level consistency model is proposed. Semantically related data are grouped together into a cluster, and the data inside a cluster are mutually consistent. However, a certain degree of inconsistency is allowed among data items at different clusters.

In [21], a control matrix is proposed for data conflict resolution. For a database of  $n$  data items, a matrix of size  $n \times n$  is used. For each broadcast cycle, the control matrix is broadcast together with the data items. A mobile client performs consistency checking using the matrix before reading any data item from the “air”. This method can handle read-only transactions well as update transactions. The write operations are performed on local copies of the data items at the client. At the end of a transaction, the whole transaction including all of the read and write operations and the cycle numbers in which they are performed will be sent to the server for commitment.

In [17, 18], the multi-version data broadcast method is proposed to resolve the problem of reading inconsistent data values. In the method the server broadcasts previous versions of data items in addition to the

---

items.

committed version of the data items at the last broadcast cycle. When a mobile transaction wants to access a data item, it will get the latest version for its first read operation. The subsequent read operations of the transaction will read the data items with the largest version number which is smaller than or equal to the data version of the first operation. By allowing a transaction to read an older version of a data item, data consistency can be ensured at the expense of currency, i.e. a mobile transaction may not receive the latest value of a data item. In order to reduce the number of versions to be broadcast and to facilitate the checking of consistency, update transactions will update the database only at the end of a broadcast cycle even though they arrive in the middle of a broadcast cycle. The number of versions to be broadcast for a data item is determined from the life-span of the transactions. It is defined as the maximum number of broadcast cycle from which the transaction reads data. Another important assumption of the multi-version broadcast method is that at least one value (the current one) is broadcast for each data item in each cycle. In addition to the most updated version, all the previous versions within a time frame need to be broadcast as well. The method is further extended for handling updates with data caches in [19].

Another method to detect the non-serializability problem is to broadcast serialization graphs [17, 18]. Each client maintains its local serialization graph to ensure that the schedules of all the committed transactions are serializable. The first drawback of this method is also the heavy overhead in broadcasting the serialization graphs since every data conflict at the database server has to be broadcast. Secondly, each client must listen to the transmission channel to maintain and update its serialization graph. This leads to another serious problem, which can affect the correctness of this approach: the need to maintain the serialization graphs at the client mobile even when it is disconnected. The mobile network is unreliable and disconnection is frequent. When disconnection occurs between a mobile client and its base station, the mobile client cannot obtain new serialization information about its transaction, making it virtually impossible to ensure serializability of transaction execution.

### **3 System Model**

In this section, we introduce our system and define the model assumptions. The model is defined based on the characteristics of the target application systems, i.e., mobile stock information systems. In these systems, the mobile clients are palmtops, wearable computers and notebooks. They generate short read-only transactions to retrieve information from the database servers, i.e., stock tickers, sport news and traffic conditions.

The mobile computing system model consists of a base station, a number of mobile clients and a mobile network such as the GSM cellular radio system, which provides limited bandwidth for data broadcast [9]. The mobile clients, which represent users equipped with mobile machines, communicate with the base stations through low bandwidth wireless channels of the mobile network.

At the base station, a database server is maintained. The database contains useful information about the external environment such as stock updates, current traffic information and news. Their values can be highly dynamic. To maintain the validity or “freshness” of the data items, update transactions are generated to refresh

the data values whenever the status of the objects has changed such as when there is a change in stock price. Each update transaction is time-stamped. The time-stamp is used to indicate at which snapshot the update is generated and is not for concurrency control. It is recorded with the new value into the data item as its version number. It is assumed that stale data items will be much less useful.

In the model, it is assumed that update transactions are short transactions, consisting mainly of one to several operations. Some update transactions may also contain read operations. It is assumed that a well-formed concurrency control protocol, such as two phase locking [3], is used for concurrency control among the update transactions at the database server. Note that although the data items are associated with time-stamps, the time-stamps will not be used for concurrency control.

The database server periodically broadcasts data items from the database one by one. Each broadcast cycle follows the preceding cycle immediately and the data items may require different time for broadcast due to different sizes. Since the bandwidths of the up-link channels are very limited, it is assumed that the broadcast scheduler does not know what are the required data items of the mobile transactions. A broadcast algorithm is adopted for determining what are the data items to be broadcast and their broadcast schedules. In the last few years, various broadcast algorithms based on the deadlines of the transactions and the access frequencies of the data items have been proposed [13, 25]. As we shall see later, our protocol does not depend on the broadcast algorithm selected and can be applied to different broadcast algorithms. In the model, each broadcast cycle is modeled as a long read-only transaction called the *broadcast transaction (BT)*. The execution of the BT and the update transactions are interleaved to reduce the blocking delay due to the installation of update transactions.

The mobile clients issue transactions, called *mobile transactions (MT)*, to access data items at the database server. It is assumed that each MT is a set of data requests (read operations) and each MT is defined with a deadline. Although they are not (soft) real-time transactions [11, 14], meeting their deadlines is an important performance objective. The requirement may be a statistical one such as 95% of the mobile transactions has to be completed within 5 seconds after the submissions. Transactions, which have missed their deadlines, might still be of some use, but beyond a certain time interval they would be considered totally useless. The period between the arrival time of a transaction and the time when it is considered to be useless is called *drop period*.

We assume that there are two types of MTs in the system. The first type is where the data items requested by a transaction are unordered. An example for this type of transactions is a request for the weather forecasts of a few cities at the same time or a request for several stock data. A single request is issued instead of several separate requests simply because batching the requests can reduce the system overhead. The arrival sequence of the data items is unimportant as long as all requested data items are received before the deadline. A transaction is completed when it has obtained all its required data items. Then, the result will report to the mobile client upon its commitment.

The second type of MT requires the data items to be delivered in a pre-defined order. An example is a

request for the next few landmarks in a road traffic navigation system. The driver would definitely expect to see the results in order so as to determine whether he is still on the right track. Another example is in stock trading analysis. The investor would like to get the last traded price of a particular stock. If the value is greater (smaller) than a certain value, he would want to know the prices of other stocks as well.

The assumptions of the system model are summarized below:

- (1) All update transactions are at the database server (base station).
- (2) The arrival rate of the update transaction is high due to the dynamic nature of the external environment.
- (3) Each broadcast cycle is modeled as a broadcast transaction.
- (4) All mobile transactions are read-only.
- (5) Mobile transactions can be *unordered* or *ordered*.
- (6) The results will be reported to the client after the completion of the transaction upon its commitment.
- (7) Each mobile transaction has a drop period. A mobile transaction will be aborted if it cannot get all the required data items from the broadcast program within the drop period.

## 4 Correctness Requirements

Before the introduction of the new concurrency control protocol, we would like to identify the correctness and performance requirements of the protocol for mobile computing systems which use data broadcast to disseminate dynamic data values to a large number of mobile clients. Two important considerations are identified: consistency and currency.

### **Consistency:**

Data consistency is an important issue in the design of a transaction processing system. In this paper, we adopt serializability as the correctness criterion [3] as it has been commonly used for conventional database systems and generally accepted in the database community. If the serialization graph is acyclic, then the schedule is serializable. Since concurrency control amongst the update transactions are assumed to be done by a well-formed concurrency control protocol, e.g., 2PL, the only inconsistent problem is the data conflicts between broadcast transactions and update transactions. Our proposed protocol concentrates on resolving this type of data conflicts to ensure that all mobile transactions will read consistent data values and the execution schedules are serializable.

### **Currency:**

A mobile computing system maintains data items for recording information in the real world, e.g., the last traded stock prices of stocks, results of sports and traffic conditions of the roads. An important property of the data items is that their values could be highly dynamic and the update rates could be very high [20]. It is therefore difficult to maintain a tight consistency between the information in the real world and the corresponding values of the data items that are observed by transactions, and to provide the latest versions of data items to transactions. The problem is especially serious if the updates are generated at a remote site. For

example, in a stock monitoring system, updates are generated at the stock exchange. They are sent to the data vendor, e.g., Reuters, which maintains a database for the stocks. Even if we assume that there is no delay in transmitting the updates from its generation site, delay may still be significant due to resource and data contention at the database server.

In a mobile computing system, while data values are being broadcast from the database server to mobile clients, updates may arrive and new values may be installed into the database. When a mobile transaction reads a data item from the broadcast channels, the version read by the transaction may not be the latest one. In this paper, we define currency based on the stale access rate, which is the percentage of out-dated data versions observed by the transactions. A high stale access rate means low currency, i.e., more out-dated information is being observed. It is important to the usefulness of the applications to maximize currency.

## 5 Sample Inconsistent Cases

In this section we briefly describe the data inconsistency problem in data broadcast and illustrate the nature of the problems with some representative examples.

### **Example 1:** Data conflict between a MT and an update transaction

Suppose the update transaction, U, updates data item  $d_5$  and then data item  $d_2$ , and a mobile transaction, MT, wants to read  $d_2$  and  $d_5$ . If the schedule is:

- i) Broadcast transaction (BT) broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii) U updates  $d_5$
- iv) U updates  $d_2$
- v) Broadcast transaction (BT) broadcasts  $d_5$
- vi) MT reads  $d_5$

The mobile transaction MT may observe inconsistent data values. The serialization graph is cyclic such as  $MT \rightarrow U \rightarrow MT$  and is non-serializable. The reason is that MT reads a data item,  $d_2$ , which is in conflict with U before the update from U and it reads a conflicting data item,  $d_5$ , after the update from U.

### **Example 2:** A MT conflicts with two (or more) update transactions

Even though the serialization order between an update transaction and a mobile transaction is not cyclic, the final serialization graph can still be cyclic due to transitive dependencies<sup>3</sup>. Suppose there are two updates  $U_1$  and  $U_2$  such that  $U_1$  will update  $d_2$  and then  $d_1$ , and  $U_2$  will update  $d_1$  and then  $d_5$ . If the schedule is:

- i) Broadcast transaction (BT) broadcasts  $d_2$

- ii) MT reads  $d_2$
- iii)  $U_1$  updates  $d_2$
- iv)  $U_1$  updates  $d_1$
- v)  $U_2$  updates  $d_1$
- vi)  $U_2$  updates  $d_5$
- vii) Broadcast transaction (BT) broadcasts  $d_5$
- viii) MT reads  $d_5$

The serialization graph is cyclic such as  $U_2 \rightarrow MT \rightarrow U_1 \rightarrow U_2$ .

**Example 3:** Non-serializability involving two or more broadcast transactions.

A MT may not be able to get all its required data items from a single broadcast cycle. Non-serializability of transaction execution may occur over more than one broadcast transaction such as:

- i)  $BT_1$  broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii) End of  $BT_1$
- iv)  $U$  updates  $d_5$
- v)  $U$  updates  $d_2$
- vi)  $BT_2$  broadcasts  $d_5$
- vii) MT reads  $d_5$

The serialization graph is cyclic such as  $MT \rightarrow U \rightarrow MT$ .

Note that in the determination of the serializability of a serialization graph, we ignore the broadcast transactions as they are considered as “intermediate transactions”. They do not have any real effect on database consistency. Their only function is to provide data items for the mobile transactions. (The reason of treating it as a transaction is to facilitate the analysis. Since the mobile transactions read data items from broadcast transactions, their serialization order will always be  $BT \rightarrow MT$  if they have any conflicts. Of course, in our case, both of them are read-only transactions, they should not have any conflicts.)

The data inconsistency problem in the first two examples can be resolved by adopting a *serial* execution method. For example, each transaction, either a broadcast or an update transaction, is defined with a unique time-stamp based on its arrival time. The execution order of the transactions follows their time -stamps, i.e., all updates arrived during a broadcast cycle will be performed at the end of the broadcast cycle. In this way, the final schedule will be serial. However, this method has three serious problems. Firstly, the concurrency of the system will be lowered. Secondly, the waiting time of update transactions can be very long as the broadcast transactions

---

<sup>3</sup> If a transaction  $T_j$  reads an uncommitted data item from another transaction  $T_i$ ,  $T_j$  will be dependent on  $T_i$ .

are long transactions. Blocking update transactions for long time may affect the “freshness” of the data items and the objective of broadcasting the most updated data values to mobile transactions may not be achieved. Thirdly, even if the serial approach is used, the problem in the third example in above is still unresolved.

## 6 Update-First with Order (UFO) Protocol

The biggest problem in designing a concurrency control protocol for a mobile computing system using data broadcast is that the mobile transactions may read a data item at any time while the data item is being broadcast and the database server in general does not notice this. Thus, instead of detecting data conflicts between mobile transactions and update transactions, the UFO protocol checks data conflicts between broadcast and update transactions.

The basic principle of the UFO protocol is to ensure that if a data conflict occurs between a broadcast transaction and an update transaction, the serialization order between them will always be  $U \rightarrow BT$ .  $BT \rightarrow U$  will be allowed only if it is impossible for a mobile transaction, which reads from BT, to be dependent on other updates directly or transitively. Since mobile transactions read data items from broadcast transactions, the serialization order between a BT and a MT is always  $BT \rightarrow MT$  if they have any conflicting data items. (Note that this situation actually will not happen since both BT and MT consist of read operations only.) Thus, serialization order between a conflicting update transaction and a mobile transaction will always be  $U \rightarrow MT$  and the final serialization graph will be serializable. (The correctness of the UFO protocol will be discussed in details in Section 6.4.)

Basically, the UFO protocol consists of two parts:

- (1) Execution of update transactions; and
- (2) Conflict resolution between update and broadcast transactions.

### 6.1 Execution of Update Transactions

The execution of an update transaction is divided into two phases: the *execution phase* and *update phase*. During the execution phase, the operations of update transaction are executed and data conflicts with other update transactions will be resolved using 2PL. The updates of the data items are written in a private workspace of the transaction. When all the operations of an update transaction have been completed, it enters the update phase in which permanent updates of the database will be performed by copying the new values from the private workspace into the database.

Although dividing the execution of an update transaction into two phases will incur additional overhead to complete a transaction, there are two important advantages:

- (1) Reduce the blocking probability of broadcast transactions.

If a broadcast transaction wants to read a data item, which is locked by an update transaction, the broadcast transaction will be blocked until the commitment of the update transaction according to the principles of 2PL. The update transaction, which is holding the lock, may be blocked due to the data conflicts with other update transactions. Due to the transitive blocking, the blocking time of the broadcast transaction can be very long. Dividing the execution of update transactions into two stages can greatly reduce the blocking probability and blocking time of broadcast transactions since data conflicts between the update and broadcast transactions will occur only when the update transaction transactions are in the update stage, which is usually much shorter.

- (2) Facilitate the detection of data conflicts between the update and broadcast transactions.

During the update phase, the system knows which data items have been accessed by the update transaction. By comparing the write set of the update transaction with the read set of the broadcast transaction, the system can determine whether there is any data conflict between them.

## 6.2 Conflict Resolution between Update and Broadcast Transactions

The data conflict between an update transaction and a broadcast transaction is detected when the update transaction enters its update phase. The conflict resolution method used to resolve the data conflict depends on the time when the update phase starts with respect to the broadcast of the conflicting data item. If it is before the broadcast of the conflicting data item, no action needs to be taken since the serialization order will be  $U \rightarrow BT$ . Note that this is our desirable serialization order between the transactions. The only problematic case is when the conflicting data item is broadcast before the arrival of the update transaction as this means the serialization order will be  $BT \rightarrow U$ . In the UFO protocol, this will be resolved by re-broadcast, which we will discuss in section 6.2.2.

### 6.2.1 Checking for Data Conflicts across Multiple Broadcast Cycles

A problem, which we need to solve in detecting the data conflicts, is that a mobile transaction may span more than one broadcast cycle (see example 3 in section 5). Thus, checking data conflict with the current broadcast transaction is not sufficient. The simplest way to solve the problem is to include the broadcast transactions of the previous cycles for data conflict checking. Obviously we would like to minimize the number of cycles to check to reduce system overhead. It turns out that the drop period of the mobile transactions can be used to determine how many cycles to be checked:

$$\begin{aligned} \text{Number of previous cycles to be checked, } n_c \\ = DP_{\max} / \text{time for one broadcast cycle} \end{aligned}$$

where  $DP_{\max}$  is the maximum drop period of all mobile transactions from the mobile clients.

### 6.2.2 Algorithm at BT

The algorithm at BT essentially checks whether there is any overlap between the read set of the data items

of the broadcast transaction(s) and the write set of data items of the update transaction. If there is an overlap, then the system will check whether any conflicting data item has already been broadcast in the last drop period. If there are, the conflicting data items will be re-broadcast.

First, we define the following quantities.

$$BT = \sum_{j=0}^{n_i} BT_{i-j}$$

for any current broadcast cycle  $i$

$O_{BT}$  = set of data items of broadcast transaction, BT

$O_U$  = set of data items of update transaction, U

$B_A = \{x \in O_{BT} \cap O_U \mid x \text{ is already broadcast when U arrives}\}$

After finishing an update transaction, the following algorithm is performed:

```

If  $O_{BT} \cap O_U = \{\}$ 
Then BT and U have no dependency
Else
  If  $B_A = \{\}$ 
  Then the serialization order is  $U \rightarrow BT$ 
  Else
    For each data item  $d_i \in B_A$ 
      re-broadcast data item  $d_i$ 
    Next
    the serialization order is  $U \rightarrow BT$ 
  End If
End If

```

The main purpose of re-broadcasting the conflicting data item is to reverse the serialization order between the broadcast transaction and the update transaction. Before the broadcast, the serialization order between the broadcast transaction and its conflicting update could be  $BT \rightarrow U$ . Re-broadcasting the conflicting data item will change their serialization order to be  $U \rightarrow BT$  if they have any conflicts.

### 6.2.3 Algorithm at MT

The algorithms for the two types of read-only MT, unordered and ordered, will be given below. The first algorithm is for unordered read-only transactions. In the algorithm, whenever a data item is received from BT, it will be matched with the set of the requested data items. The process is repeated until all the requested data items are received or the drop period of the MT is expired.

First we define the following quantities.

DB = set of data items in the database

$S_{MT}$  = set of data items requested by MT

read-set = { }

loop until (read-set =  $S_{MT}$ ) or (drop period of MT is expired)

    read an data item  $d_i$  from BT

    if  $d_i \in S_{MT}$

        then read-set = read-set  $\cup$  { $d_i$ }

    end if

end loop

The second algorithm is for ordered MT. In the algorithm, the data items requested by a MT will be represented by a linked list. Processing of a MT starts from the first data request in the linked list. For each data item received from a BT, it will be matched with the current item in the linked list. If there is a match, MT will process the data item and the linked list will be updated. The update included adding new items and/or removing items from the list according to the processing logic at the MT. The process is repeated for the next data request until there is no more data request in the linked list or the drop period of the MT is expired. In case a data item, which is already read, is re-broadcast, MT will repeat processing from that data item again using the re-broadcast value and update the linked list accordingly.

First we define the following quantity.

$L_{MT}$  = list of data items requested by MT

$d_c$  = first item in  $L_{MT}$

loop until  $L_{MT}$  exhausted or (drop period of MT is expired)

    read an data item  $d_i$  from BT

    if  $d_i = d_c$

        then MT process  $d_i$  and update  $L_{MT}$

    else if  $d_i \in \{d_j\}$  where  $1 \leq j \leq c-1$  (i.e. if any data item already read is re-broadcast)

        MT repeat the processing on  $d_i$  and update  $L_{MT}$

    end if

$d_c$  = the next data item in  $L_{MT}$

end loop

#### 6.2.4 Reducing the Number of Data Re-Broadcast

Although under normal situations, the number of data re-broadcast due to the data conflicts should be small (due to short update transactions), the cost can still be substantial if the sizes of the conflicting data items are very large, such as video files. Furthermore, for some applications, the update transactions may have locality, e.g., the update rate of some data items is much more higher than the others. Many broadcast algorithms select data items based on the access frequencies of the data items, e.g., they broadcast the “hot” data items, which are more wanted by the mobile transactions. If the update locality is similar to the access frequencies of the data items, the probability of data conflict between the broadcast transaction and update transactions can be higher.

One way to reduce the number of data re-broadcast is to re-broadcast a data item only if the conflicting update transaction has other data conflicts with other broadcast transactions or other update transactions. The reason is that if re-broadcast is not performed, the serialization order between BT and U will be  $BT \rightarrow U$ . Thus, there is a risk of having a cyclic serialization graph if the update transaction conflicts with other transactions. However, if there is no further data conflict involving the update transaction, the serialization graph will not be cyclic, e.g.  $\dots \rightarrow BT \rightarrow U$ . To perform the checking of serializability, a serialization graph is maintained at the database server. In the graph, the nodes are update transactions. An update transaction will be included in the graph if:

- 1) it has a data conflict with a broadcast transaction; or
- 2) it has a data conflict with an update transactions defined in case 1; and
- 3) the difference in current time and their start time is smaller than the drop period.

If there is a data conflict between a broadcast transaction and an update transaction, or a data conflict between an update transaction with any transactions defined in the serialization graph, the serialization graph will be searched and updated. Re-broadcast of the conflicting data item between an update transaction and a broadcast transaction in the graph will be required if there is a cycle in the graph. Transactions are removed from the serialization graph in case their start times are older than the current time by the drop period.

### **6.2.5 Disconnection**

An important property of mobile network is frequent disconnection, which may be voluntary or involuntary. For a voluntary disconnection, the mobile client initiates a disconnection in order to conserve power of the mobile machine. Voluntary disconnection of a mobile client will only occur after the completion of its transaction. Thus, it will not affect the mechanism of the UFO protocol and create any problem to the correctness of the protocol. Involuntary disconnection results from instability in the mobile network. For example, in a cellular radio network, the strength of signal received by a mobile client is affected by a number of factors such as the distance between the mobile clients and the base station, as well as the surrounding buildings. Disconnection may occur once the signal received by the mobile client or the base station is too low. Usually, the disconnection is temporary. However, its impact on the consistency of transaction execution can be very serious.

The main problem of disconnection on the UFO protocol is that at the time of re-broadcast a mobile client

may be disconnected from the network. In that case, the mobile client cannot get the new version of the data item and the serializability order between the mobile transaction and the update transaction cannot be reversed. If they have any further data conflicts, the resulting schedule may be non-serializable.

One method to solve the problem is to include additional information in the header of a broadcast cycle. The information contains the identities and the time-stamps of all the data items, which have been re-broadcast in the previous drop period. When a mobile client reconnects, it has to wait until the start of a broadcast cycle. Then, it checks the header of the broadcast cycle to see whether any of its accessed data items have been rebroadcast while it is disconnected from the network. If there are some such data items, the accessed data items will be marked invalid and it has to get the data items again.

### 6.3 Examples

In this sub-section, we re-examine the examples in section 4 and discuss how the UFO protocol resolves the data inconsistency problem. In the examples below, we do not include the mechanism discussed in the previous section for reducing number of data re-broadcast and concentrate on unordered MT in order to simplify the discussion.

#### Example 1:

- i) Broadcast transaction (BT) broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii) Compare the data sets of U and BT
- iv) U updates  $d_5$
- v) U updates  $d_2$
- vi) BT re-broadcasts  $d_2$
- vii) MT reads the most updated value of  $d_2$
- viii) BT continues and broadcasts  $d_5$
- ix) MT reads  $d_5$

The serialization graph is acyclic such as  $U \rightarrow MT$ .

#### Example 2:

- i) BT broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii) Compare the data sets of  $U_1$  and BT
- iv)  $U_1$  updates  $d_2$
- v)  $U_1$  updates  $d_1$
- vi) BT re-broadcasts  $d_2$

- vii) MT reads  $d_2$  again
- viii) Compare the data sets of  $U_2$  and BT
- ix)  $U_2$  updates  $d_1$
- x)  $U_2$  updates  $d_5$
- xi) BT continues its process and broadcasts  $d_5$
- xii) MT reads  $d_5$

The serialization graph is acyclic such as  $U_1 \rightarrow U_2 \rightarrow MT$

**Example 3:**

- i)  $BT_1$  broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii) End of  $BT_1$
- iv) Start of  $BT_2$
- v) Compare the data sets of U and  $BT_1$  and  $BT_2$
- vi) U updates  $d_5$
- vii) U updates  $d_2$
- viii)  $BT_2$  re-broadcasts  $d_2$
- ix) MT reads  $d_2$  again
- x)  $BT_2$  broadcast  $d_5$
- xi) MT reads  $d_5$

The serialization graph is acyclic such as  $U \rightarrow MT$ .

## 6.4 Implementation and Correctness of the UFO protocol

### 6.4.1 Implementation

In addition to ensuring all mobile transactions read *consistent* data values, an important property of the UFO protocol is that it does not affect the mechanism of the broadcast algorithm and it can be applied to different broadcast algorithms (*flexible*) i.e. no change is required for the underlying broadcast algorithm. Yet another advantage of the UFO protocol is that every time a data item is being broadcast, it will be the most updated version (*currency*).

We have briefly mentioned some desirable properties of the UFO protocol. Now we will discuss why the implementation of the protocol is simple and incurs *minimal overhead*. In Section 7, we will confirm this observation with simulation experiments. Here, we simply note that the implementation of the UFO protocol does not require any changes in the mobile clients except in the case of getting rebroadcast data items. All the checking for data consistency is performed at the database server. The overhead for data conflict detection should be low. Specifically, the only overheads of the protocol are:

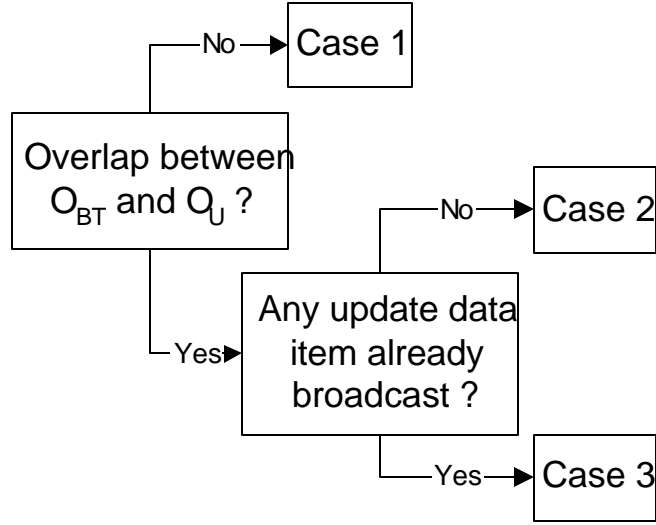
- (1) division of the execution of update transactions into two phases;
- (2) checking of the data sets of broadcast transactions and an update transaction whenever an update transaction wants to enter the update phase; and
- (3) re-broadcast of conflicting data items in case its broadcast occurs before the start of the update phase of the update transaction.

Dividing the execution of update transactions into two phases is trivial and should not incur much additional overheads. (This is similar to the deferred update approach [3].) The overhead for checking conflicting data sets and the probability of data conflict should be low as the number of data items to be updated by an update transaction is usually small. To speed up the checking process, the data items to be updated may be sorted according to their Ids. Even if there is a data conflict, re-broadcast is only required for the case where the conflicting data items are broadcast before the update phase of the update transaction causing the data conflict and the final serialization graph will be cyclic between the update and broadcast transactions. The MT algorithm is simple and does not incur additional overhead for checking. The only additional work is to replace the old version of a data item in case it is re-broadcast.

#### 6.4.2 Correctness

In this section, we discuss the correctness of the UFO protocol such that the serialization graphs among the mobile transactions and the update transaction are always acyclic. We will concentrate on the serializability between the update transactions and the broadcast transactions. In the proof below, all cases regarding the data conflicts between BT and U will be considered. These cases correspond to the different sections of the protocol defined in section 6.2.2. In each case, the serialization graph (SG) will be considered, where  $SG = (N, E)$ .  $N$  is the set of nodes representing the set of transactions. Since the only transactions for all cases is BT and U, therefore  $N = \{BT, U\}$ .  $E$  is the set of ordered pairs  $(T_i, T_j)$  representing that there is at least one pair of conflicting operations between  $T_i$  and  $T_j$  and the operation in  $T_i$  precedes the one in  $T_j$ . In our presentation,  $T_i \rightarrow T_j$  will be used to represent the order pairs  $(T_i, T_j)$ .

Cases that may occur are summarized in the following diagram:



### Case 1

This case corresponds to the ‘if-then’ part of the first level if-then-else loop in 6.2.2.

In this case there is no overlap between the sets of data items of the BT and any update transaction U, i.e.  $O_{BT} \cap O_U = \{\}$ . Therefore there is no conflict operation between BT and U, and  $E = \{\}$ . As a result SG is unconnected graph, with two nodes BT and U, and hence is acyclic.

### Case 2

This case corresponds to the ‘if-then’ part of the second level of the if-then-else loop in 6.2.2.

In this case there is an overlap between the sets of data items of the BT and any update transaction U, i.e.  $O_{BT} \cap O_U \neq \{\}$ . However  $\forall x \in O_{BT} \cap O_U$ ,  $x$  is not broadcast yet when the update phase starts. Therefore  $w_U(x) \rightarrow_{r_{BT}(x)}$  and  $E = \{U \rightarrow BT\}$ . The resultant SG is again acyclic.

### Case 3

This case corresponds to the ‘else’ part of the second level if-then-else loop in 6.2.2.

In this case, there is an overlap between the sets of data items of the BT and any update transaction U. i.e.  $O_{BT} \cap O_U \neq \{\}$ . In addition,  $\exists x \in O_{BT} \cap O_U$  such that  $x$  is already broadcast when the update phase of U starts. At this point in time,  $E = \{BT \rightarrow U\}$  since  $\forall x \in B_A$ ,  $r_{BT}(x) \rightarrow w_U(x)$ . Therefore if a mobile transaction commits on or before this point in time, SG is acyclic.

If the broadcast transaction continues its original broadcast schedule after the update phase, then  $U \rightarrow BT$  will be added to E since  $\forall x \notin B_A$ ,  $w_U(x) \rightarrow r_{BT}(x)$ . i.e.  $E = \{BT \rightarrow U, U \rightarrow BT\}$  which is a cyclic schedule. According to the protocol in 6.2.2  $\forall x \in B_A$ ,  $x$  will be re-broadcast at once to clear the dependency  $BT \rightarrow U$ . Therefore  $E = \{U \rightarrow BT\}$  and the resultant SG is again acyclic.

In the enhancement of the protocol, re-broadcast will only be required when the update transaction has further conflicts with the broadcast transaction or other update transactions, which has data conflict with the broadcast transaction. In that case, even  $U \rightarrow BT \in E, U \rightarrow BT$  will not be in  $E$ . Thus, the final schedule is still serializable.

## 7 Performance Studies

In this section, we study the performance of the UFO protocol using extensive simulation experiments. In order to better illustrate the performance characteristics of the UFO protocol, we compare it with another efficient concurrency control for data broadcast systems, the multiversion broadcast (MV) [18] which is also for data broadcast systems with read-only mobile transactions. MV has been shown to perform better than other methods such as the invalidation report scheme and serialization graph testing scheme [18]. Another reason of choosing MV is that it has a similar model assumption as the UFO algorithm, i.e., it assumes that the mobile transactions have deadlines. Following the definition in [18], the number of versions to be broadcast in MV is the number of versions created within the period of (current time – drop period of a mobile transaction) to current time. In MV, all updates arrived during a broadcast cycle are performed at the end of the broadcast cycle. A mobile transaction is required to read data items having the same version. We will investigate the performance of the protocols in meeting the deadlines of the transactions as well as the mean response times of the transactions under different workload and system setting. Since the probability of data re-broadcast depends on the probability of data conflicts between the update transactions and mobile transactions, we will test the UFO and MV protocols when the mobile transactions and update transactions have different data access patterns. A flat broadcast disk is assumed for selecting data items for broadcast in order to simplify the model.

### 7.1 Simulation Model

The simulation model is developed based on the model introduced in section 3. Basically, the simulation model consists of three main entities and four processes as shown in Figure 1. The three entities are the database server, the air media and the mobile clients. Three of the four processes are at the database server: the broadcast process, the server update process and the update database process. It is assumed that the database server is a multi-processor system and the processes can be done concurrently provided that they are not accessing the same data item at the same time. For the latter case, blocking will be used to solve the data synchronization problem. The last process is the client process at each mobile client for generating mobile transactions.

The database server is located at the base station. It maintains a database. It is assumed that the data items have similar size and each data item has a unique identifier. Air media is a collection of air channels. To simplify the case, we consider only one broadcast channel for both scheduled broadcast items and re-broadcast items. It is assumed that there are a large number of mobile clients in the system. They generate read-only transactions, the mobile transactions.

The broadcast process selects data items from the database for broadcast. To simplify the case, we assume a flat disk broadcast disk model. The broadcast cycle includes all the data items in the database. The broadcast process receives data items that need to be re-broadcast from the update database process. For each time slot in a broadcast cycle, the broadcast process first checks whether there are any data items waiting for re-broadcast in the re-broadcast queue. If there is, it will de-queue the first data item from the queue and re-broadcast it first. Otherwise, it broadcasts the next item in the pre-defined broadcast schedule.

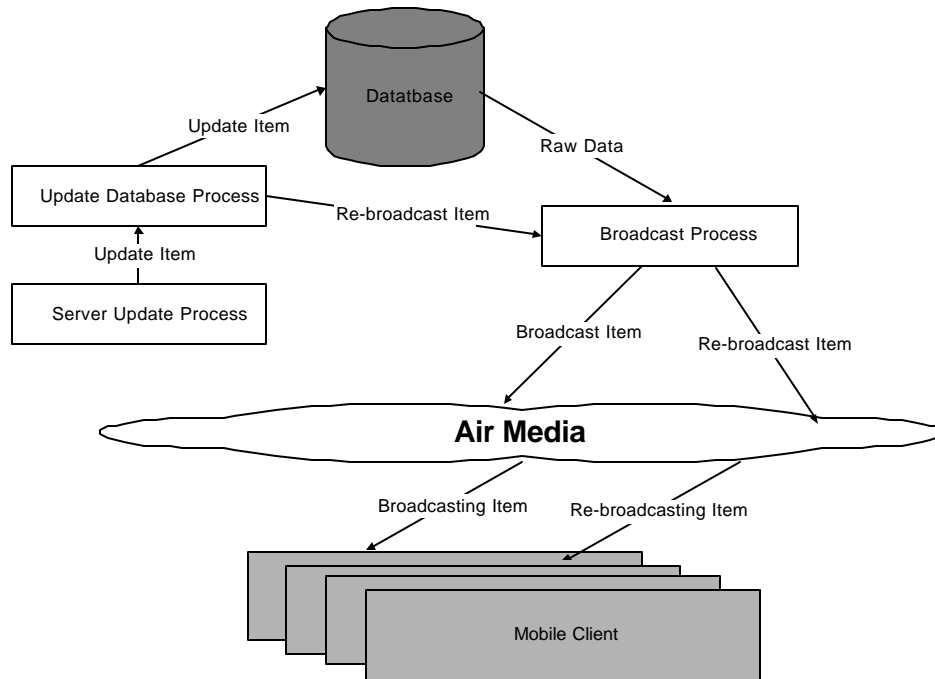


Figure 1. Simulation model for a mobile computing system with data broadcast

The server update process generates update transactions. We assume that all update transactions are processed at the base station. Each update transaction may read and write data items. The data items to be updated by an update transaction are assumed to be spread over the database uniformly. (In the experiments, we also test the system when the required data items of the update transactions following the Zipf distribution.) Each update transaction is assigned a unique time-stamp using the current time of its generation. The server update process uses a deferred update mechanism to update the database such that the actual update on the database is performed at the end of an update transaction in batch.

The update database process performs permanent update on the database and sends updated items that need to be re-broadcast to the broadcast process. According to the UFO protocol, the update database process will check data conflict against the broadcast transaction in current and some previous broadcast cycles. The number of cycles to be checked depends on the drop period of mobile client transactions as explained in section 6. It is not necessary to check for data items that are already broadcast more than a drop period ago.

The mobile client process generates mobile transactions. Once generated, the transaction will listen to the broadcast program continuously. It is assumed that the mobile transactions have the same drop period. The transactions, which have missed their deadlines, will be aborted since they will become totally useless. In the simulation model, we further assume that each mobile transaction is a collection of operations and the operations are unordered and they are unordered, e.g., they can get their required data items in any order. When any of its required data items is being broadcast, it will get the data item from the “air”. Then, it will access the CPU for computation. When a mobile transaction has received all its required data items, it can commit and is completed. The data access pattern of the mobile transactions follows the uniform distribution (or the Zipf distribution in the latter sets of the experiments). The mobile client generates the next mobile transaction after a thinktime upon the completion or abort of a mobile transaction. The think time is assumed to be exponentially distributed.

## 7.2 Model Parameters and Performance Measures

Table 1 lists the model parameters and their baseline values.

Database size	1000
No. of client	100
Broadcast rate	20 data items / sec
No. of data item in a mobile transaction	1 to 4
No. of data item in an update transaction	1 to 2
Drop period	20, 40, 60 sec
Mean think time of client	10 sec
Mean time between update transactions (MTBU)	0.1 to 20 sec
Skew coefficient for skewed data access	0.5, 1.0, 1.5

Table 1. Model parameters and baseline values

The primary performance measures are miss rate, mean response time, stale access rate, broadcast overhead and rebroadcast hit rate (for UFO only). The miss rate measures the number of mobile transactions, which miss their deadlines, over the total number of mobile transactions generated. It indicates the capability of meeting the timing requirements of the system. In measuring mean response time, the aborted transactions are also included. Stale access rate is the proportion of data items which are not the most current values corresponding to the external environment. Broadcast overhead in UFO is the utilization of broadcast channel for re-broadcast. Broadcast overhead in MV is the utilization of broadcast channel for broadcasting data versions which are not the current versions. Rebroadcast hit rate defines the number data requires which are satisfied by the data items from re-broadcast. It is only for UFO.

## 7.3 Performance Results and Discussion

The performance of the UFO protocol is studied through four sets of experiments using different data access patterns for mobile transactions (MT) and update transactions (U).

- (1) In the first set of experiments, we test the performance of the UFO and MV protocols when the

distribution of the required data items of the mobile transactions and the update items of the update transactions are uniformly distributed in the database.

- (2) In the second set of experiments, we investigate their performance when the data items of the mobile transactions follow a skewed distribution, the Zipf distribution, which is commonly used in similar studies on the area.
- (3) In the third set of experiments, the data access pattern of both mobile transactions and update transactions are skewed in the same way. Thus, they have the same set of hot data items.
- (4) The last set of experiments is the same as the third one except there is an offset between the distribution of the data items of the update and mobile transactions.

The simulation program is implemented using CSIM and the simulation length of each run is the completion of 200,000 mobile transactions. It is determined from a number of trial runs until the results are stable.

### 7.3.1 Uniform Data Access

In this set of experiments, the data access patterns of both MT and U are uniform. Figures 2 to 5 shows the results when the inter-arrival time of U is varied (different update transaction workload). As can be observed in Figure 2, the miss rates of MT in both UFO and MV drop with an increase in inter-arrival time of U (lower update transaction workload). Consistent with the results of miss rate, the mean response time also decreases gradually with a decrease in update workload as shown in Figure 3. The results are consistent with our expectation. For UFO, at a lower update transaction workload, the probability of data conflicts between MT and U will be smaller. Thus, MT can obtain their required data items earlier as a result of smaller probability of blocking due to data contention and smaller number of data re-broadcast. The smaller re-broadcast overhead at a lighter update transaction can be observed in Figure 4. As expected, the miss rate is smaller with a longer drop period as MT are allowed to wait longer for their data items. Similarly, the performance of MV is better when the update transaction workload is lower. This is due to smaller broadcast overhead as shown in Figure 4. When the update workload is lighter, the number of multiple versions for broadcast will also be smaller.

As shown in Figures 2 to 3, the miss rate and mean response time of UFO are both smaller than that of MV. The main reason is due the lower broadcast overhead of UFO as compared with MV as shown in Figure 4. From Figure 4, it is interesting to see that the broadcast overhead of MV decreases with drop period while the broadcast overhead of UFO is insensitive to changes in drop period. Instead, it depends on the conflict probability between update and broadcast transactions. Another important advantage of UFO as compared with MV is that the stale access rate of UFO is close to zero as shown in Figure 5. It is because the re-broadcast mechanism helps to maximize the currency of the data items provided to the mobile transactions. MV has a much higher stale access rate since a mobile transaction may read old versions.

### 7.3.2 Skewed Data Accesses of MT

In this set of experiments, we study the performance of UFO and MV when the required data items of the

mobile transactions are not evenly distributed in the database, e.g., some locality in data references. The data access pattern of the MT follows the Zipf distribution and the access pattern of the update transaction is uniform. The drop period for MT is set to be 40 seconds.

Figures 6 to 8 show the performance of UFO and MV for different update workloads and different skew coefficients for MT. A larger skew coefficient implies the set of hot data items for update is smaller. It can be seen that the miss rates of UFO are consistently lower than that of MV and the stale access rates of UFO remain close to zero. Again, the smaller miss rate of UFO is due to smaller broadcast overhead as shown in Figure 8.

### **7.3.3 Skewed Data Access of Both MT and U (No Offset)**

In this set of experiments, the data access pattern of both mobile transactions and update transactions follow the Zipf distribution and there is no offset between the data distributions of the update and mobile transactions. That is both mobile transactions and update transactions have the same set of hot data items. The results are shown in Figures 9 to 12. Consistently to the results from previous two sets of experiments, the performance of UFO is better than MV. Furthermore, the differences in their performance become even greater. For example, the difference in miss rate between UFO and MV is more than 50% when the update workload is heavy as shown in Figure 9. However, the better performance of UFO is not due to lower broadcast overhead. Actually, its overheads are consistently higher than that of MV for most cases as shown in Figure 11. Note that the number of multiple versions for broadcast in MV depends on the number of data items updated in the last broadcast cycle. When the update distribution is skewed, the number of data items updated in a cycle will be smaller than the case of uniform update distribution. On the other hand, the number of rebroadcast in UFO depends on the number of data items updates performed and less affected by the distribution of data items updated.

The better performance of UFO as compared with MV is due to high re-broadcast hit rate as shown in Figure 12. Although re-broadcasting a data item increase the broadcast overhead, at the same time it may meet the data requirements of some mobile transactions if the data items are “hot” data items since the update transactions and mobile transactions have the same set of hot items. Thus, we can see in Figure 9 that miss rate is smaller for larger skew coefficient which implies that the set of hot data items is smaller. For skew coefficient equals to 1.5, the re-broadcast hit can be update to 9 items per second as shown in Figure 12.

### **7.3.4 MT & U Skewed Access (Offset = 10%)**

In this set of experiments, the data access patterns of the MT and update transactions are both skewed with offset equals to 10%. That is MT and U have different sets of hot data items. The results are shown in Figures 13 to 16. Comparing Figure 13 with Figure 9, we can see that the performance of UFO is seriously affected by introducing an offset between the hot data items of update and mobile transactions. It is because the re-broadcast items are less likely wanted by the mobile transactions. The re-broadcast hit rate is thus much lower those shown in Figure 12. This also explains why the miss rate of UFO is smaller when the skew coefficient is smaller. Although smaller number of mobile transactions can benefit from data re-broadcast, the performance of

UFO is still better than MV in terms of miss rate and stale access rate as shown in Figures 13 and 14.

## 8 Conclusions and Future Work

Data broadcast has been shown to be an efficient method for disseminating data items to transactions generated by mobile clients. Although the research in the design of broadcast algorithms has received a lot of attention in previous years, the concurrency control issue has been greatly ignored. If the broadcast of data items and the execution of update transactions, which are important to maintain the freshness of the data items at the database, are uncontrolled, the mobile transactions may observe inconsistent data values. In this paper, we study the concurrency control between update transactions and mobile transactions, which consist of read-only operations. A new protocol, called Update First with Order (UFO), is proposed. Although the protocol is simple, it can effectively maintain the schedule update and mobile transactions to be serializable. Furthermore, it has several important properties such as it can be applied to different broadcast algorithms and its overhead for maintaining serializable schedules is low especially when the update workload is not heavy. Its impact on the mechanism in the mobile clients is minimal and it can maximize the currency of data items provided to mobile transactions. The only important impact on the client side is the change in the broadcast schedule due to data re-broadcast. Thus, some energy saving methods, which require the clients to get the broadcast index first and then “sleep” in order to reduce the energy power consumption, may not be suitable to work with UFO. Currently, we are working on the issue to reduce the impact of data re-broadcast in UFO on the broadcast schedule and to minimize the increase in energy consumption due to changes in broadcast schedule.

In order to investigate the performance characteristics of the UFO protocol, a simulation model of a mobile computing system using data broadcast has been implemented. Experiments are performed to investigate the impact of different system parameters such as inter-arrival time of update transactions, drop periods and the distribution of the requested data items of transactions on the performance.

Currently, we are extending the UFO protocol for systems where the clients maintain cached data for their own mobile transactions, and the executions of the operations in a transaction are pre-defined. The re-broadcast scheme in the UFO protocol can introduce a lot of transaction restarts if the operations are ordered. Once a transaction is restarted, it has to wait for all its required data items if the client, which originates the transaction, does not maintain any cached data items. Although maintaining cached data items can greatly reduce the data access delay and the restart overhead of a transaction, it requires an additional validation scheme for checking the consistency of the cached data items. In our current work, we are designing an efficient invalidation scheme into the UFO protocol for checking the data items at client cache.

## References

- [1] Acharya, S., Alonso, R., Franklin, M., Zdonik, S. (1995) Broadcast Disks: Data Management for Asymmetric Communications Environments. Proceedings of ACM SIGMOD, San Jose, CA USA, 22–25 May, p.199 – 210. Association for Computing Machinery.

- [2] Acharya, S., Franklin, M., Zdonik, S. (1997) Balancing Push and Pull for Data Broadcast. Proceedings of ACM SIGMOD, Tucson, Arizona USA, 11–15 May, p.183-194. Association for Computing Machinery.
- [3] Bernstein, P.A., Hadzilacos, V., Goodman, N. (1987) Concurrency Control and Recovery in Database System. Addison-Wesley Publishing Company.
- [4] Datta, A., Celik, A., Kim, J. and VanderMeer, D.E. (1997) Adaptive Broadcast Protocol to Support Power Conservant Retrieval by Mobile Users. Proceedings of 13th International Conference on Data Engineering, 7–11 Apr, p.124-133. IEEE Online Publications.
- [5] Anindya Datta, Ashlihan Celik, Debra E. VanderMeer and V. Kumar. (1999) Adaptive Broadcast Protocols to Support Efficient and Energy Conserving Retrieval from databases in Mobile Computing Environments. ACM Transactions on Database Systems, vol. 24, no. 1, March, p.1-79. Association for Computing Machinery.
- [6] Franklin, M. and Zdonik, S. (1998) Data In Your Face: Push Technology in Prospective. Proceedings of ACM SIGMOD Conference, Seattle, WA USA, 1–4 June, p.516-519. Association for Computing Machinery.
- [7] Hu, Q., Lee, D.L., and Lee, W.C. (1998) A Comparison of Indexing Methods for Data Broadcast on the Air. Proceeding 12<sup>th</sup> International Conference on Information Networking, 21-23 Jan, p.656-659. IEEE Online Publications.
- [8] Hameed, S. and Vaidya, N.H. (1997) Efficient Algorithms for Scheduling Single and Multiple Channel Data Broadcast. Technical Report 97-002, Department of Computer Science, Texas, A&M University, Feb.
- [9] Haug, T. (1994) Overview of GSM: Philosophy and Results. International Journal of Wireless Information Networks, vol. 1, no. 1, p.7-16.
- [10] Imielinski, T. and Badrinath, B.R. (1994) Mobile Wireless Computing: Challenges in Data Management. Communications of the ACM, vol. 37, no. 10, Oct, p.18-28. Association for Computing Machinery.
- [11] Kayan, E. and O. Ulusoy (1999) An Evaluation of Real-Time Transaction Management Issues in Mobile Database Systems. The Computer Journal, vol.42, no.6.
- [12] Leong, H.V. and Si, A. (1995) Data broadcasting strategies over multiple unreliable wireless channels. Proceedings of the 4th International Conference on Information and Knowledge Management, Baltimore, MD USA, 28 Nov-2 Dec, p.96-104.
- [13] Leong, H.V. and Si, A. (1997) Database Caching over the Air-Storage. The Computer Journal. Volume 40, number 7, p.401-415.
- [14] Ozsoyoglu G. and Snodgrass, R. (1995) Temporal and Real-Time Databases: A Survey. IEEE Transactions on Knowledge and Data Engineering, vol. 7, no. 4, p.513-532.
- [15] Pitoura, E. and Bhargava, B. (1993) Dealing with Mobility: Issues and Research Challenges. Technical Report, Purdue University, Nov.
- [16] Pitoura, E. and Bhargava, B. (1995) Maintaining Consistency of Data in Mobile Distributed Environments. Proceedings of the 15<sup>th</sup> International Conference on Distributed Computing Systems, 30 May-2 Jun, p.404-413. IEEE Online Publications.

- [17] Pitoura, E.(1998) Supporting Read-Only Transactions in Wireless Broadcasting. Proceedings of the DEXA'98 Workshop on Mobility in Databases and Distributed Systems , 26-28 August, p.428-433. IEEE Online Publications.
- [18] Pitoura, E. & Chrysanthis, P.K. (1999) Scalable Processing of Read-Only Transactions in Broadcast Push. Proceedings of the 19th IEEE International Conference on Distributed Computing System, 31 May-4 Jun, p.432-439. IEEE Online Publications.
- [19] Pitoura E. and Chrysanthis, P.K. (1999) Exploiting Versions for Handling Updates in Broadcast Disks. Proceedings of International Conference on Very Large Data Base, Edinburgh, UK, 7-10 Sep, p. 114-125.
- [20] K. Ramamritham, P. Deolasee, A. Katkar, A. Panchbudhe, and Prashant Shenoy (2000) Dissemination of Dynamic Data on the Internet. Proceedings of International Workshop on Databases in Networked Information Systems, University of Aizu, Japan, December, p. 173-187.
- [21] Jayavel Shanmugasundaram, Arvind Nithrakashyap, Rajendran Sivasankaran and Krithi Ramamritham (1999) Efficient Concurrency Control for Broadcast Environments. Proceedings of ACM SIGMOD International Conference on Management of Data, Philadelphia, Penn., 1-3 June, p. 85-96.
- [22] Srinivasan, R., Liang C., Ramamritham, K. (1998) Maintaining Temporal Coherency of Virtual Data Warehouses. Proceeding of Real Time Systems Symposium, Madrid, Spain, 2-4 Dec, p.60-70.
- [23] Ulusoy, O. (1998) Real-Time Data Management for Mobile Computing. Proceedings of International Workshop on Issues and Applications of Database Technology (IADT'98), Berlin, Germany, p. 233-240.
- [24] Wolfson, O, Sistla, A.P., Chamberlain, S. and Ye sha, Y. (1999) Updating and Querying Databases that Track Mobile Units. Distributed and Parallel Databases, vol. 7, p.257-287.
- [25] Xuan, P., O. Gonzalez, J. Fernandez & Ramamritham, K. (1997) Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments. Proceedings of 3<sup>rd</sup> IEEE Real-Time Technology Application Symposium, 9-11 Jun, p. 38 – 48.

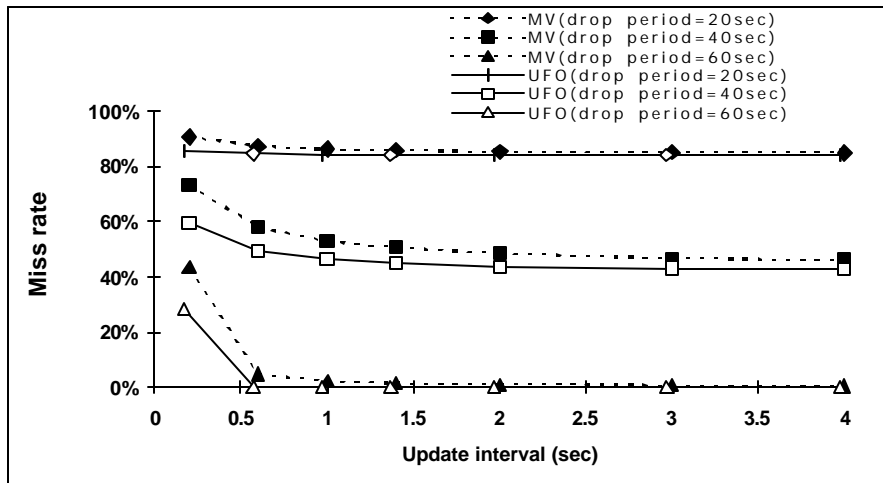


Figure 2. Miss rate (MT & U uniform access)

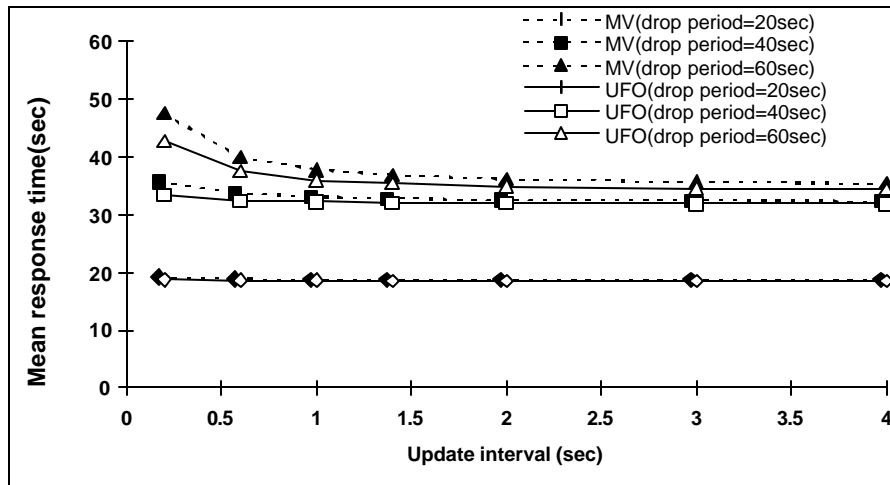


Figure 3. Mean response time (MT & U uniform access)

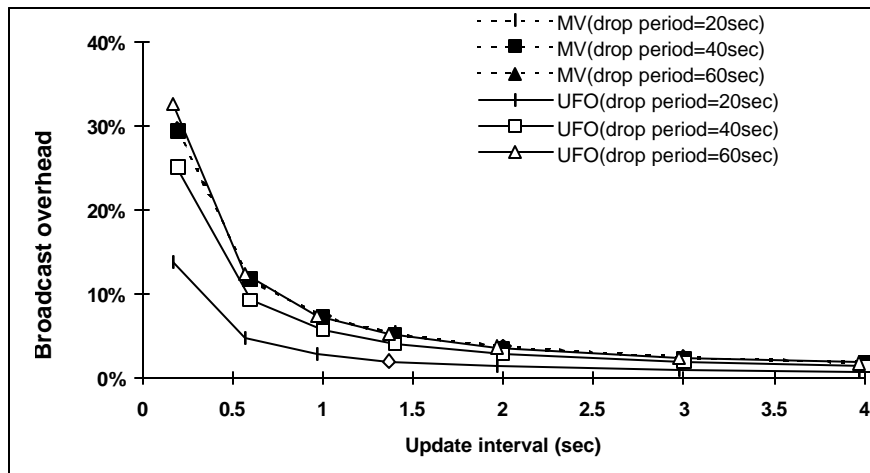


Figure 4. Broadcast overhead (MT & U uniform access)

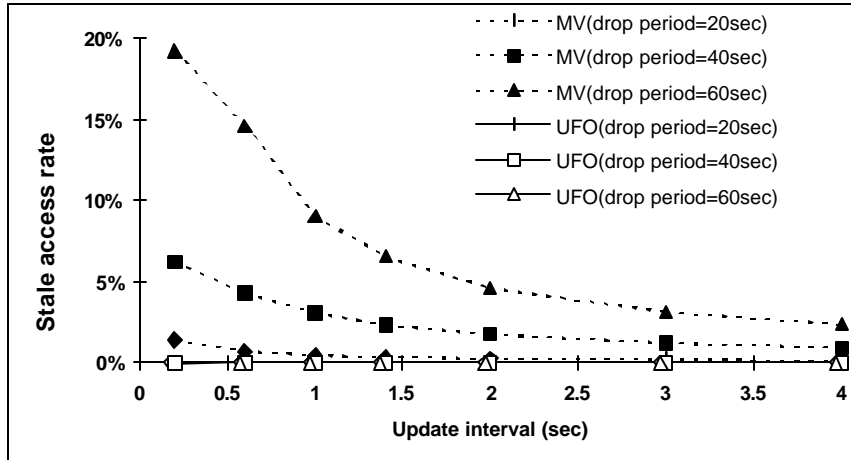


Figure 5. Stale access rate (MT & U uniform access)

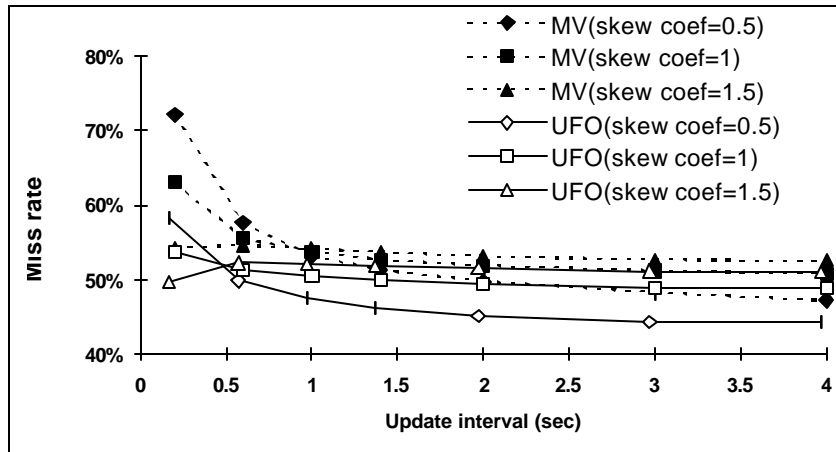


Figure 6. Miss rate (MT skewed access, U uniform access)

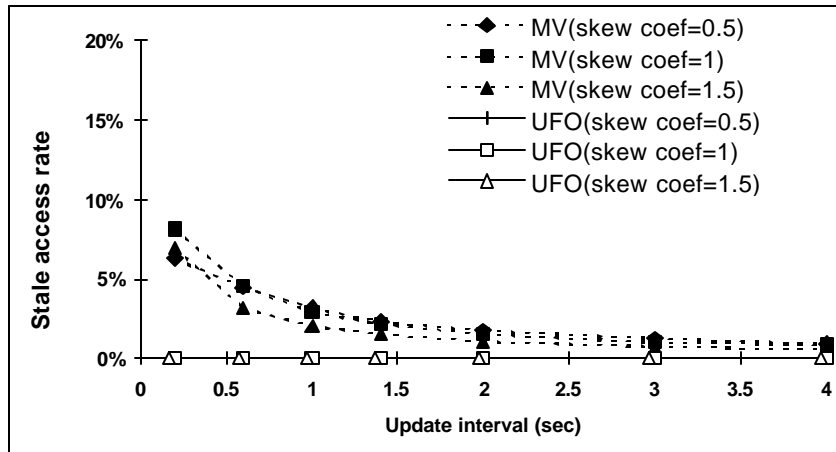


Figure 7. Stale access rate (MT skewed access, U uniform access)

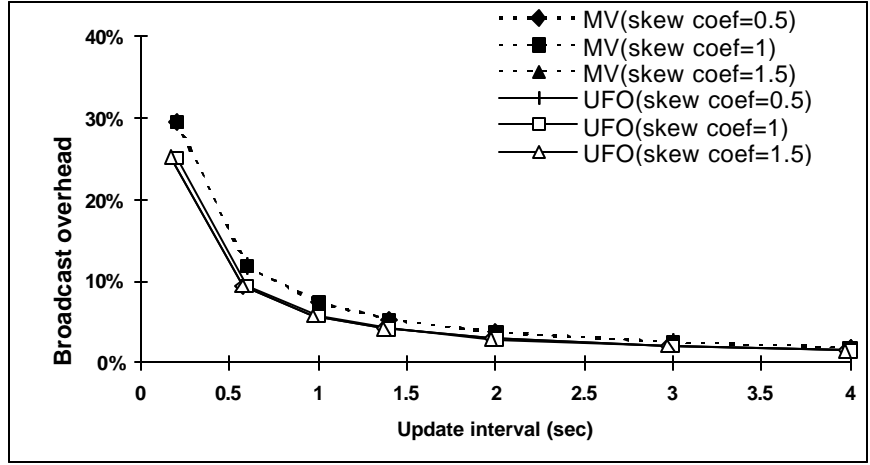


Figure 8. Broadcast overhead (MT skewed access, U uniform access)

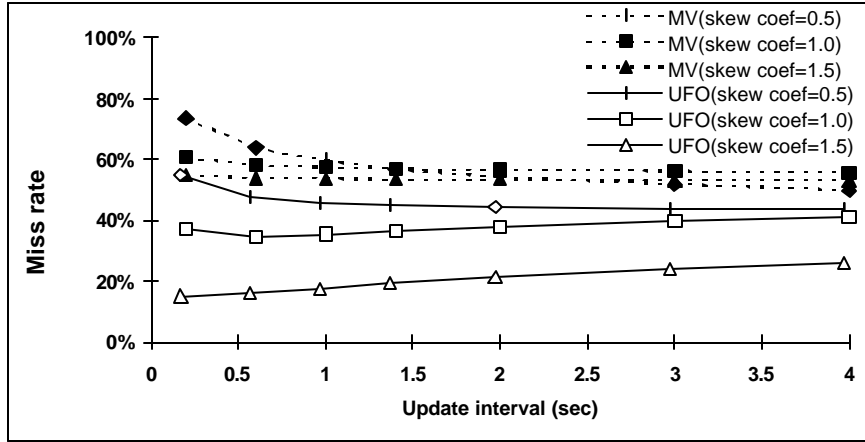


Figure 9. Miss rate (MT & U skewed access, no offset)

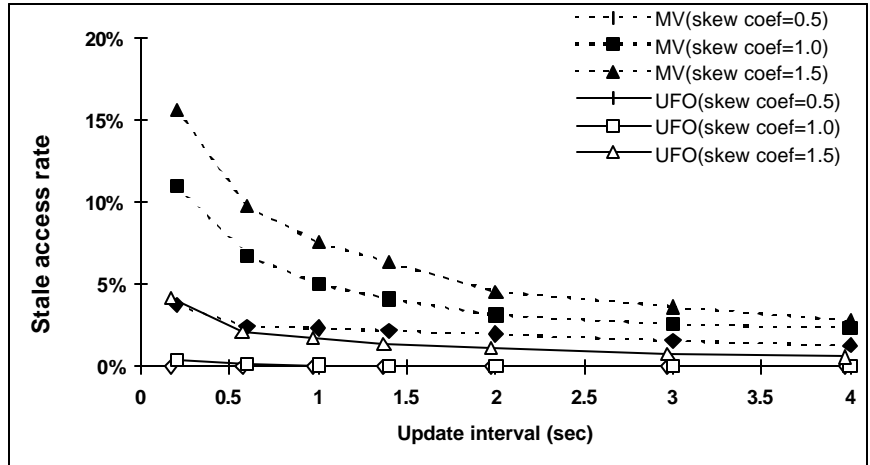


Figure 10. Stake access rate (MT & U skewed access, no offset)

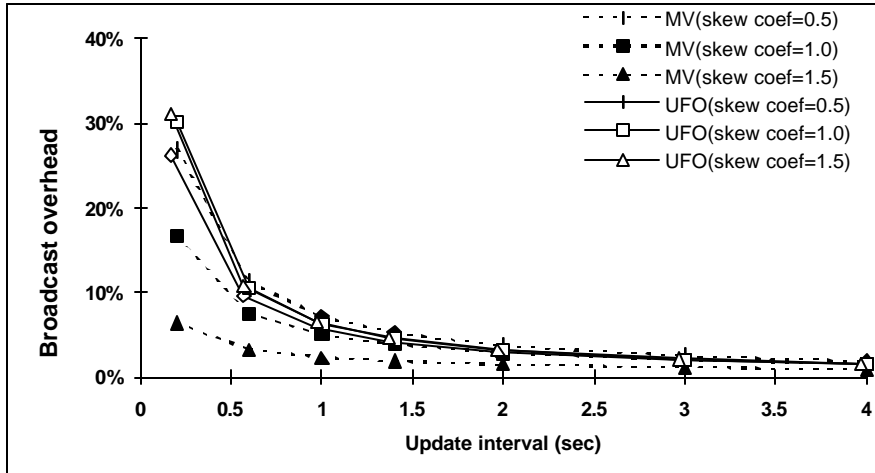


Figure 11. Broadcast overhead (MT & U skewed access, no offset)

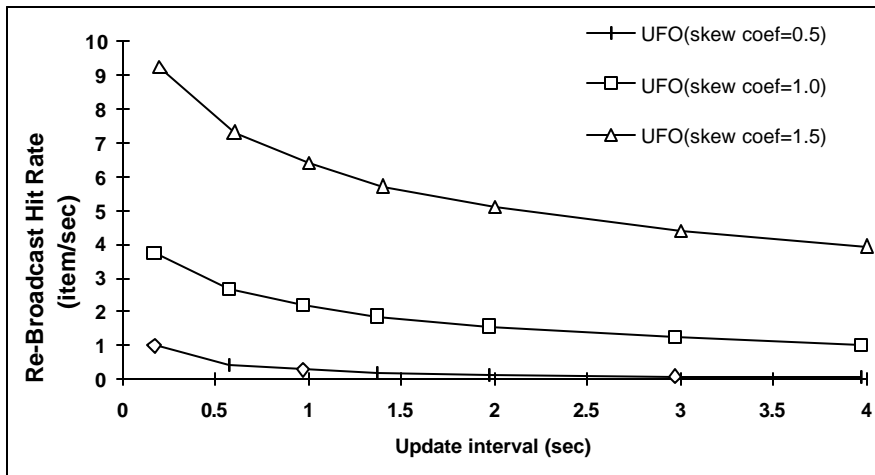


Figure 12. Re-broadcast hit rate (MT & U skewed access, no offset)

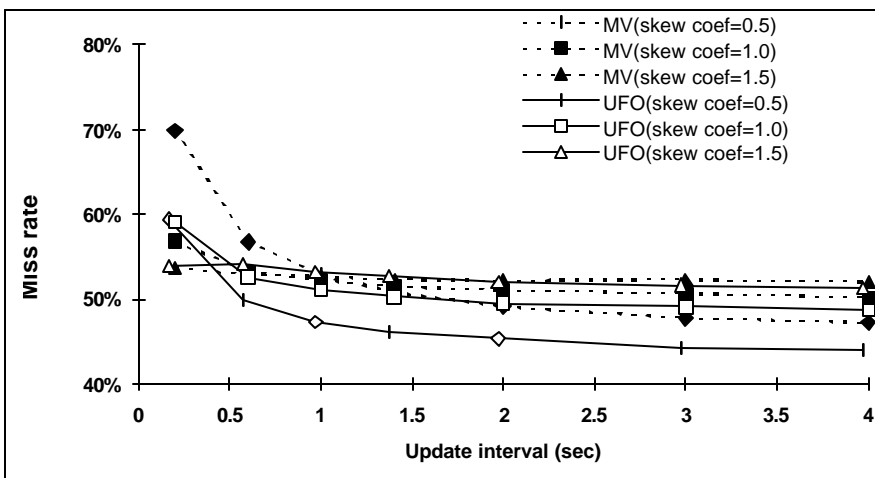


Figure 13. Miss rate (MT & U skewed access, offset = 10%)

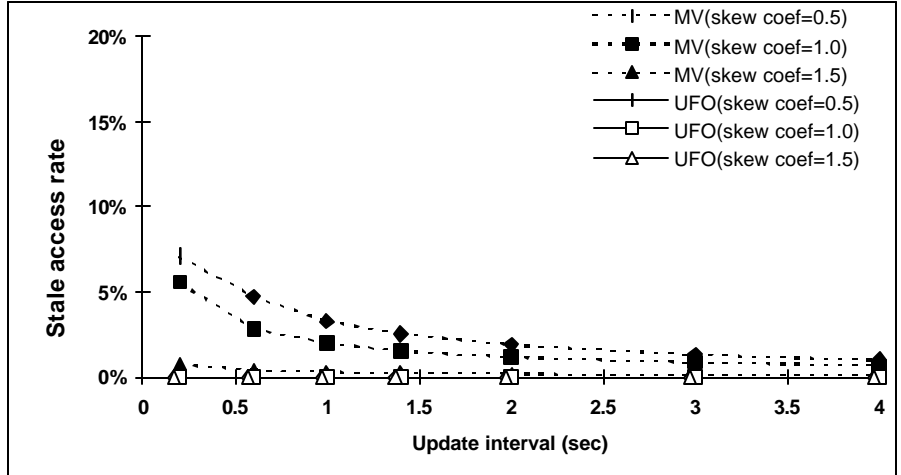


Figure 14. Stale access rate (MT & U skewed access, offset = 10%)

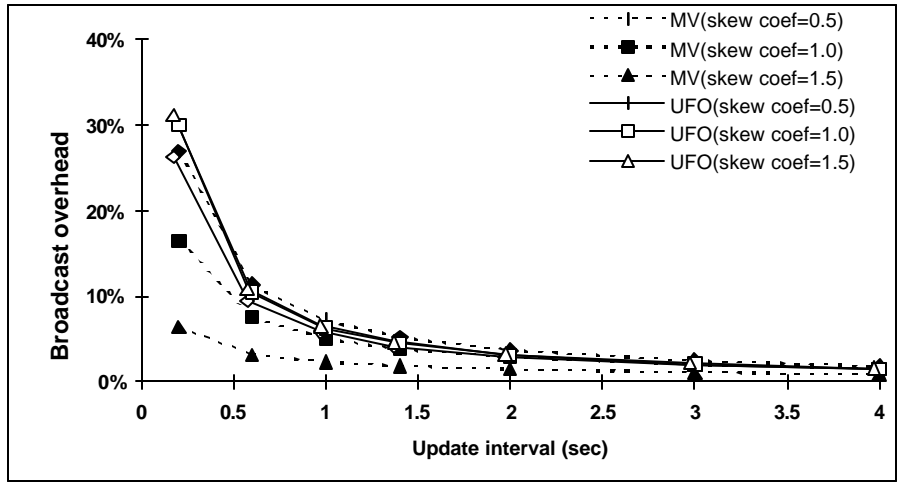


Figure 15. Broadcast overhead (MT & U skewed access, offset = 10%)

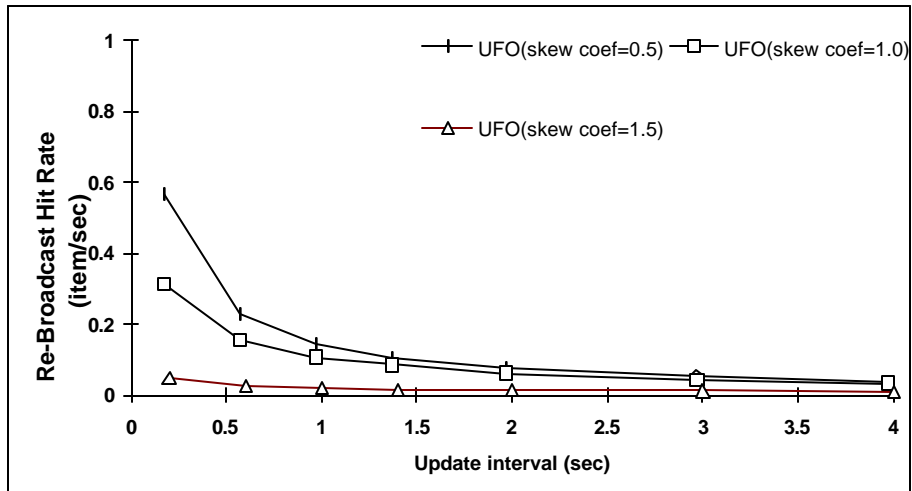


Figure 16. Re-broadcast hit rate (MT & U skewed access, offset = 10%)