Concise Papers

Using Predeclaration for Efficient Read-Only Transaction Processing in Wireless Data Broadcast

SangKeun Lee, Chong-Sun Hwang, and Masaru Kitsuregawa, *Member*, *IEEE*

Abstract—Wireless data broadcast allows a large number of users to retrieve data simultaneously in mobile databases, resulting in an efficient way of using the scarce wireless bandwidth. However, the efficiency of data access methods is limited by an inherent property that data can only be accessed strictly sequentially by users. To properly cope with the inherent property, this paper presents three predeclaration-based transaction processing methods that yield a significant performance improvement in wireless data broadcast.

Index Terms—Wireless data broadcast, mobile data access, transaction processing, predeclaration, mutual consistency of data.

1 Introduction

In a wireless data broadcast environment, the server periodically broadcasts data items over one or more channels to a large client population, where data items correspond to database records (tuples). Data items are identified by their primary key and the total number of data items is relatively small (on the order of hundreds, sometimes thousands, of objects). The smallest logical unit of the broadcast, often called a *bucket*, is assumed to correspond to a single data item only for convenience. Each period of broadcast is called a broadcast cycle or *bcycle*, while the content of the broadcast is called a *bcast*. While data items are being broadcasted, update transactions are executed at the server that modify the values of data items broadcasted. Without any appropriate concurrency control, it is possible that wireless transactions generated by mobile clients may observe inconsistent data values.

The main challenge of this paper is to design a mechanism to provide consistent data items requested in a certain order by wireless read-only transactions. Clients will only receive the broadcast data and fetch individual items (identified by a key) from the broadcast channel. To properly cope with the inherent property of data broadcast that data can only be accessed strictly sequential, we explore a predeclaration-based query optimization and devise three predeclaration-based transaction processing methods. The proposed methods, presented in the context of broadcast disks [1] and a multiple channel environment [10], are particularly intended for applications like the online auction application, where the size of the database is relatively small, but the number of clients is very large. We also evaluate the performance of the proposed methods by an analytical study. The analytical results show that predeclaration-based query optimization can yield a significant performance improvement in wireless data broadcast.

• S. Lee and C.-S. Hwang are with the Department of Computer Science and Engineering, Korea University, Seoul, Korea. Email: yalphy@korea.ac.kr and hwang@disys.korea.ac.kr.
• M. Kitsuregawa is with the Institute of Industrial Science, The University of Tokyo, Tokyo, Japan. E-mail: kitsure@tkl.iis.u-tokyo.ac.jp.

Manuscript received 13 Feb. 2001; revised 16 Aug. 2002; accepted 16 Jan. 2003.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 113617.

1.1 Motivating Example

The three different broadcast organizations are illustrated in Fig. 1, where the server broadcasts a set of data items

$$\{d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$$

in one or two broadcast channel(s) according to broadcast schedules (d_1 is the relatively most frequently accessed item, d_2 and d_3 are lesser ones, and d_4, d_5, d_6, and d_7 are least ones).

To show that the order in which a transaction reads data affects the response time of the transaction, suppose that a client starts its transaction at the exactly middle of each bcycle in Fig. 1:

$$\text{IF } (d_3 < 0) \text{ THEN read}(d_1) \text{ ELSE read}(d_2)$$

That is, if the value of d_3 is negative, the transaction needs the value of d_1. Otherwise, it needs the value of d_2. For the uniform bcast, where the transaction starts at a half point within d_4, since both d_1 and d_2 precede d_3 in the bcast with respect to the client and access to data is strictly sequential, the transaction has to read d_3 first and wait to read the value of d_1 or d_2. Thus, the response time of the transaction is 11.5, i.e.,

$$d_4 \rightarrow d_5 \rightarrow d_6 \rightarrow d_7 \rightarrow d_1 \rightarrow d_2 \rightarrow \overbrace{d_3} \rightarrow$$
$$d_4 \rightarrow d_5 \rightarrow d_6 \rightarrow d_7 \rightarrow \overbrace{d_1},$$

or 12.5, i.e.,

$$d_4 \rightarrow d_5 \rightarrow d_6 \rightarrow d_7 \rightarrow d_1 \rightarrow d_2 \rightarrow \overbrace{d_3} \rightarrow$$
$$d_4 \rightarrow d_5 \rightarrow d_6 \rightarrow d_7 \rightarrow d_1 \rightarrow \overbrace{d_2}.$$

If, however, all data items that will be accessed *potentially* by the transaction, i.e., $\{d_1, d_2, d_3\}$, are predeclared in advance, a client can hold all necessary data items with a reduced response time of 6.5 (i.e., $d_4 \rightarrow d_5 \rightarrow d_6 \rightarrow d_7 \rightarrow \overbrace{d_1} \rightarrow \overbrace{d_2} \rightarrow \overbrace{d_3}$).

This is also true of the nonuniform bcast where the response time of the transaction can be reduced from 7 or 8 to 5. For the multiple channel bcast, albeit the reduction of response time is not observed from this specific example, we believe there is a performance benefit in most cases (and verify the argument in Section 3). Thus, the use of predeclaration allows the necessary items to be retrieved in the order in which they are broadcasted as opposed to the order specified in the transaction.

2 Proposed Methods

Three predeclaration-based transaction processing methods are devised here: P (Predeclaration), PA (Predeclaration with Autoprefetching), and PA^2 (PA/Asynchronous). The central idea is to employ predeclaration of readset in order to minimize the number of different bcycles from which transactions pick up data. The assumptions made in our proposed methods are listed below:

- The client population and their access patterns do not change, so the location of each data item in the broadcast channel(s) remains fixed. Clients also know a priori the contents of the channel(s). Clients simply tune into the broadcast channel and filter all the data until the required items are downloaded. In practice, however, some index information must be made available to clients for selective tuning. We make this simplifying assumption because the

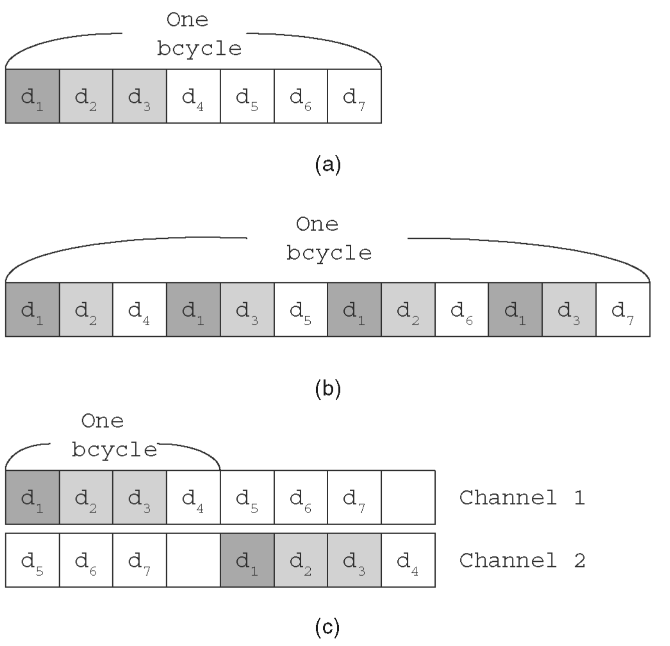

Fig. 1. Broadcast organizations. (a) Uniforma bcast, (b) nonuniform bcast, and (c) multiple channel bcast.

indexing problem is orthogonal to transaction processing methods.[1]

- All buckets have an *offset*, as well as data items, to the beginning of the next bcast. This is to guide the clients that have to tune in the next bcast.
- The information about the readset of a transaction is available at the beginning of transaction processing by using a preprocessor on a client, e.g., to identify all the items appearing on a transaction program before being submitted to the client system.
- *Serializability* is adopted as our correctness criterion. We will show that serializability is *not* expensive to achieve in the proposed methods, although it has widely been considered to be expensive to achieve for asymmetric communication environments [8].

We define the predeclared readset of a transaction T, denoted by $Pre\_RS(T)$, to be a set of data items that T reads *potentially*. For all methods, each client processes T in three phases: *preparation*, *acquisition*, and *delivery*.

2.1 Method P

For method P, we assume that the following always hold [6], [8]:

Server Requirement for P. The server broadcasts data values produced by only serializable, committed update transactions in each bcycle.

One obvious way to satisfy this requirement is to make each bcycle represent the (serializable) state of the database at the beginning of the cycle. Here, the size of database being transmitted cannot be too large as poor currency may be experienced at clients. For many applications, like online auctions and traffic control, however, the number of data items being transmitted is not too large.

With the above server requirement, the execution of each read-only transaction is clearly serializable if a client can fetch all data items within a single bcycle. Since, however, a transaction is expected to start at some point within a bcycle, its acquisition phase may therefore be across more than one bcycle. To remedy this problem, in P, a client starts the acquisition phase synchronously, i.e., at the beginning of the next bcycle. Since all data items for its transaction are already identified, the client will complete the acquisition phase within a single bcycle. More specifically, a client processes its transaction T_i as follows:

```
1. On receiving $Begin(T_i)$ {
       get $Pre\_RS(T_i)$ by using preprocessor;
       $Acquire(T_i) = \emptyset$;
       tune in at the beginning of the next bcast;
   }
2. While ($Pre\_RS(T_i) \neq Acquire(T_i)$) {
     for $d_j$ in $Pre\_RS(T_i)$ {
         download $d_j$;
         put $d_j$ into local storage;
         $Acquire(T_i) \Leftarrow d_j$;
   }
3. Deliver data items to $T_i$ according to the order in which $T_i$
   requires, and then commit $T_i$.
```

Theorem 1. *P generates serializable execution of read-only transactions.*

2.2 Methods PA and PA^2

A client tends to repeatedly query a subset of database items with a high degree of locality. This subset is thus a "hot spot" for the client. A client can cache data items in the hot spot locally to reduce access latency. Caching reduces the latency of transactions since transactions find data of interest in their local cache and, thus, need to access the broadcast channel for a smaller number of times. In our work, clients use their available hard disks as local caches and caching technique is employed in the context of transaction processing.

In order to keep the clients' caches consistent with the updated data values, the client-cached copies of modified items must be invalidated or updated. Among various approaches to communicating updates to the clients, it has been shown in the work [2], [3] that the client cache coherency can be effectively maintained by exploiting a periodic *invalidation report*, which is a list of the items that have been updated recently. Broadcasting identifiers of updated items, however, may consume a large portion of the broadcast channel, which is a scarce resource, especially when a large portion of items in the database are updated. Furthermore, in the context of the serializability consistency model, consistency must be preserved across reads of multiple data items.

In our work, the server is required to broadcast an *invalidation bit pattern* which is followed by a bcast. In an invalidation bit pattern, each bit corresponds to a single data item in the database. A bit is set to 1 if its corresponding data item has been updated during the previous bcycle, but not installed into the previous bcast. The remaining bits are set to 0s. This way, compared to invalidation reports, the size of invalidation information broadcasted by the server can be significantly reduced, especially when a large portion of items in the database are updated.

Server Requirement for PA **and** PA^2. In each bcycle, the server broadcasts an invalidation bit pattern which is followed by data values produced by only serializable, committed update transactions.

At the beginning of each bcycle, a client tunes in and reads the invalidation bit pattern broadcasted by the server. For any data item d_i in its local cache, if a bit corresponding to d_i is 1 in the invalidation bit pattern, the client marks d_i as "invalid" and gets d_i again from the current bcast and puts it into local cache. Cache management in our scheme is therefore an invalidation combined with a form of autoprefetching [2]. Invalidated data items remain

1. Our work, however, is also applicable to the case where some form of directory information is broadcasted along with data items without loss of generality. Some techniques for broadcasting index information are given in [4], [7], [9] for each broadcast organization.

TABLE 1
Parameter Settings

Parameter	Meaning	Value(s)
D	num. of items in the DB	1000
C	num. of physical channels in a multiple channel environment	2
μ	update rate per item	varying (5×10^{-4} per unit)
n	num. of data partitions in a nonuniform bcast	3
λ_1, λ_2, λ_3	broadcast frequency of each partition in a bcycle	4, 2, 1
$\|P_1\|$, $\|P_2\|$, $\|P_3\|$	size of each partition	50, 150, 800
k (MA only)	num. of old versions per item included in a bcast	2
m	num. of items actually accessed by a transaction	varying (10)
m_p (P, PA, PA^2 only)	num. of items appearing on a transaction program	$\frac{3}{2}m$
$p(P_1)$, $p(P_2)$, $p(P_3)$	access probability of each partition	0.7, 0.2, 0.1
cache invalidation	client cache invalidation scheme	invalidation with autoprefetching

in cache to be autoprefetched later. In particular, at the next appearance of the invalidated data item in the bcast, the client fetches its new value and replaces the old one.

There are two choices on when to start the acquisition phase: One is a synchronous approach where, as is the case with P, a client fetches data items from the beginning of the next bcycle. We call this method PA. More specifically, PA works as follows:

```
1. On receiving $Begin(T_i)$ {
       get $Pre\_RS(T_i)$ by using preprocessor;
       $Acquire(T_i) = \emptyset$;
       tune in at the beginning of the next bcast;
   }
2. Fetch an invalidation bit pattern;
   For every item $d_i$ in local cache {
       if (a corresponding invalidation bit is set to 1)
           { mark $d_i$ as "invalid";}
   }
   For every "valid" item $d_i$ in local cache {
       if ($d_i \in Pre\_RS(T_i)$) { $Acquire(T_i) \Leftarrow d_i$; }
   }
   While ($Pre\_RS(T_i) \neq Acquire(T_i)$) {
       for $d_j$ in $Pre\_RS(T_i) - Acquire(T_i)$ {
           download $d_j$;
           put $d_j$ into local cache;
           $Acquire(T_i) \Leftarrow d_j$;
       }
   }
3. Deliver data items to $T_i$ according to the order in which $T_i$
   requires, and then commit $T_i$.
```

Theorem 2. *PA generates serializable execution of read-only transactions.*

The synchronous approach of method PA may incur unnecessary response time latency to short transactions in a rarely updated database. For example, if most of the data items reside in local cache and all missed items can be retrieved from the current bcast, then a transaction would be completed within a single bcycle in which it is initiated.

To get over the disadvantage of method PA, a client can take an asynchronous way, i.e., it fetches data items immediately without waiting for the next bcycle. Unlike synchronous approaches, the acquisition phase may span across two different bcasts in this case. This method is referred to as PA^2. It goes as follows:

```
1. On receiving $Begin(T_i)$ {
       get $Pre\_RS(T_i)$ by using preprocessor;
       $Acquire(T_i) = \emptyset$;
   }
2. For every "valid" item $d_i$ in local cache {
       if ($d_i \in Pre\_RS(T_i)$) { $Acquire(T_i) \Leftarrow d_i$; }
   }
   While ($Pre\_RS(T_i) \neq Acquire(T_i)$) {
       for $d_j$ in $Pre\_RS(T_i) - Acquire(T_i)$ {
           download $d_j$;
           put $d_j$ into local cache;
           $Acquire(T_i) \Leftarrow d_j$;
           if (it is time to receive an invalidation bit pattern) {
               tune in and fetch an invalidation bit pattern;
               for every item $d_i$ in local cache {
                   if (a corresponding invalidation bit is set to 1) {
                       mark $d_i$ as "invalid";
                       $Acquire(T_i) = Acquire(T_i) - \{d_i\}$;
                   }
               }
           }
       }
   }
3. Deliver data items to $T_i$ according to the order which $T_i$
   requires, and then commit $T_i$.
```

Theorem 3. *PA^2 generates serializable execution of read-only transactions.*

3 Performance Evaluation

We have developed analytical models [5] to compare predeclaration-based transaction processing methods with other

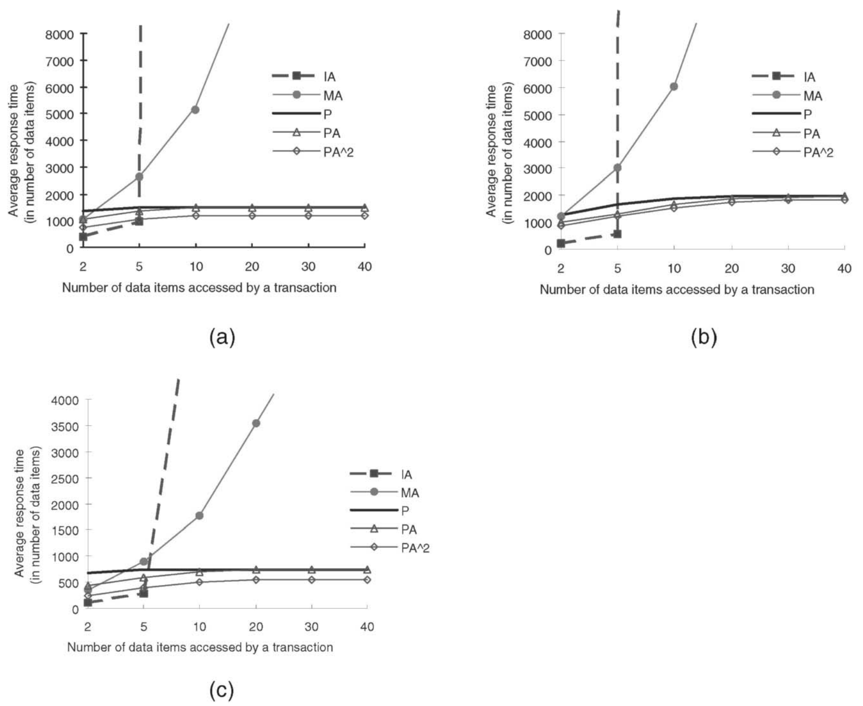

Fig. 2. Effect of transaction size. (a) Uniform bcast, (b) nonuniform bcast, and (c) multiple channel bcast.

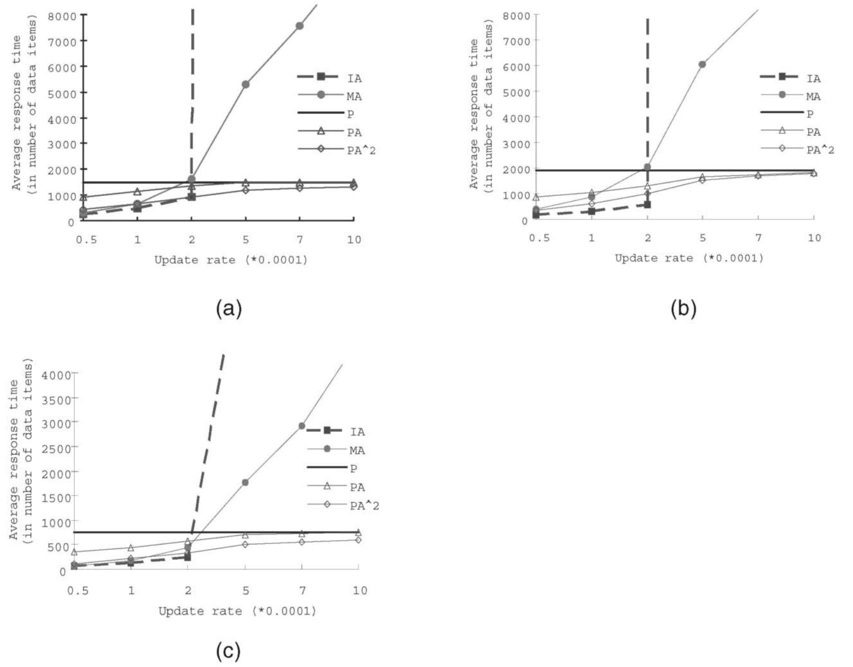

Fig. 3. Effect of update rate. (a) Uniform bcast, (b) nonuniform bcast, and (c) multiple channel bcast.

two methods, Invalidation with Autoprefetching (IA) and Multiversion with Autoprefetching (MA), which are slightly modified versions from ones proposed by [6]. We report two analytical results in this section. In the results, one time unit corresponds to the physical time taken to broadcast a single item on the server side. Table 1 summarizes the parameter settings for the server (the top half) and a client (the bottom half), where values in parentheses are default ones. Throughout the performance evaluation, in particular, the additional reads overhead in determining a predeclared readset is set to 50 percent in terms of the number of data items.

3.1 Effect of Transaction Size

Fig. 2 shows the performance behavior as the number of data items accessed by a transaction is increased in each bcast when update rate per item is set to 5×10^{-4} per unit.[2] We see that, for long transactions (the number of accessed items are greater than 5 in our analysis), the response time of IA is increased rapidly. This is because a large value m decreases the probability of a transaction's commitment. As a result, a transaction suffers from many restarts

2. This value indicates, when D is set to 1,000, that about 40 , 50, and 22 percent of database items are updated in the uniform, nonuniform, and multiple channel bcasts during a bcycle.

until it commits. MA avoids this problem by making a client access old versions on each bcast, thereby increasing the chance of a transaction's commitment. We can observe that the performance of MA is less sensitive to the number of items than IA. However, the increased size of bcast affects the response time negatively in each bcast. This explains why MA is inferior to IA for small data items. With our P, PA, and PA^2 methods, as a transaction can access data items in the order they are broadcasted, the average response time is almost independent of transaction size. As a result, our methods outperform MA, which in turn outperforms IA, when the number of items is large.

3.2 Effect of Update Rate

Fig. 3 shows the effect of update rate on the performance of various methods when m is set to 10. A higher update rate means a lower cache hit ratio and also a higher probability of cache invalidation. This explains why the response time of IA deteriorates so rapidly. In particular, if $\mu > 2 \times 10^{-4}$ per unit,[3] IA results in unacceptable performance (in case of the multiple channel bcast, when $\mu > 5 \times 10^{-4}$ per unit). MA also degenerates as the update rate increases. This is because a higher update rate leads to a larger number of updated items in the database, resulting in a larger bcast size. Unlike IA, however, with MA, a transaction can proceed and commit by reading appropriate versions of items which are on the air. This difference of commitment probability is the main reason why MA beats IA for high update rate (in our analysis, when $\mu > 2 \times 10^{-4}$ per unit). With P, PA, and PA^2, the response time is not significantly affected by the update rate . As expected, it is observed that ours are superior to MA and IA when the update rate is high. Only when a small portion of items in the database is updated during a bcycle, is the response time of ours worse than that of MA and IA.

4 Concluding Remarks

In this paper, we have proposed three simple yet robust predeclaration-based methods to speed up processing of wireless read-only transactions without sacrificing serializability. To the best of our knowledge, this work is the first approach to query optimization on the client side for reducing transaction response time in wireless data broadcast. Our scheme allows transactions to retrieve data items in the order they are broadcasted as opposed to the order specified in the transaction.

3. This value indicates, when D is set to 1,000, that more than 20 and 23 percent of database items are updated in the uniform and nonuniform bcasts during a bcycle.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments," *Proc. ACM SIGMOD Conf. Management of Data,* pp. 199-210, 1995.

[2] S. Acharya, M. Franklin, and S. Zdonik, "Disseminating Updates on Broadcast Disks," *Proc. 22nd Int'l Conf. Very Large Data Bases,* pp. 354-365, 1996.

[3] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching in Mobile Environments," *Proc. ACM SIGMOD Conf. Management of Data,* pp. 1-12, 1994.

[4] T. Imielinski, S. Viswanathan, and B. Badrinath, "Data on Air: Organization and Access," *IEEE Trans. Knowledge and Data Eng.,* vol. 9, no. 3, pp. 353-372, 1997.

[5] S. Lee, M. Kitsuregawa, and C.-S. Hwang, "Efficient Processing of Wireless Read-Only Transactions in Data Broadcast," *Proc. 12th Int'l Workshop Research Issues on Data Eng.: Eng. E-Commerce/E-Business Systems,* pp. 101-111, 2002.

[6] E. Pitoura and P. Chrysanthis, " Exploiting Versions for Handling Updates in Broadcast Disks," *Proc. 25th Int'l Conf. Very Large Data Bases,* pp. 114-125, 1999.

[7] K. Prabhakara, K. Hua, and J. Oh, "Multi-Level Multi-Channel Air Cache Designs for Broadcasting in a Mobile Environment," *Proc. 16th IEEE Int'l Conf. Data Eng.,* pp. 167-176, 2000.

[8] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham, "Efficient Concurrency Control for Broadcast Environments," *Proc. ACM SIGMOD Conf. Management of Data,* pp. 85-96, 1999.

[9] K.-L. Tan and J.X. Yu, "Energy Efficient Filtering of Nonuniform Broadcast," *Proc. 16th Int'l Conf. Distributed Computing Systems,* pp. 520-527, 1996.

[10] N.H. Vaidya and S. Hameed, "Scheduling Data Broadcast in Asymmetric Communication Environments," *Wireless Networks,* vol. 5, no. 3, pp. 171-182, 1999.