Stretch-Optimal Scheduling for On-Demand Data Broadcasts

Yiqiong Wu and Guohong Cao
Department of Computer Science & Engineering
The Pennsylvania State University, University Park, PA 16802
E-mail: {ywu,gcao}@cse.psu.edu

Abstract—

In order to effectively utilize the broadcast bandwidth, it is necessary to have efficient on-line scheduling algorithms that can balance individual and overall performance and can scale in terms of database sizes and client populations. Moreover, the scheduling algorithm should be applicable to a heterogeneous environment where data items have different sizes. We address these issues in this paper. As stretch is widely adopted as a performance metric for variable-size data requests, we propose a broadcast scheduling algorithm to optimize the system performance in terms of stretch. We show by analysis that the proposed algorithm can indeed achieve the optimal performance in terms of stretch. Moreover, the proposed scheduling algorithm has very low decision overhead, which makes it a practical solution for on-demand broadcast scheduling. Simulation results demonstrate that our algorithm significantly outperforms existing scheduling algorithms under various scenarios.

I. Introduction

For dissemination-based applications, unicast data delivery as used by current web servers may not be scalable. With unicast, a data item must be transmitted for each client that requests it, and then the load on the server and the network increases with the number of clients. Broadcasting techniques have good scalability since a single broadcast response can potentially satisfy many clients simultaneously. A key consideration in the design of broadcast system is the algorithm used to schedule the broadcast. Scheduling algorithms have been extensively studied in the context of operating systems. Traditional scheduling algorithms such as first come first served (FCFS), round robin, shortest job first, are used in processor scheduling and disk task scheduling. These scheduling algorithms are used to optimize the response time, throughput, or fairness. Since most of these scheduling algorithms are designed in the point-to-point communication environments, they may not be applicable to the broadcasting environments.

A number of researchers addressed issues on broadcast scheduling. For example, Wong [12] studied several scheduling algorithms such as first-come-first served (FCFS), longest wait time (LWT), most requests first (MRF) in broadcasting environments. Researches in [6], [9] investigate techniques to index the broadcasted data to save power in mobile environments. Su and Tassiulas [10] formulate the broadcast scheduling as a dynamic optimization problem and propose efficient suboptimal solutions which can achieve mean access latency close to the lower bound. Hameed and Vaidya [8] apply existing fair queueing algorithms to broadcast scheduling. Most of the research [3], [7], [10], [11] on broadcast scheduling focus on *push-based* broadcasting; that is, the server delivers data using a periodic broadcast program based on precompiled access profiles and typically, without any active user intervention. Such schemes are limited in their ability to adapt to dynamic user needs.

There is another kind of broadcasting: *on-demand (pull-based)* broadcasting, where a large and dynamic client population request data from an information server which broadcasts data to the clients based on these requests. Aksoy and Franklin [4] initiate the study of on-demand broadcast scheduling. They proposed a $R \times W$ scheduling algorithm which provides balanced treatment of both hot and cold data resulting in a good overall performance. The algorithm combines MRF and FCFS, and uses a novel pruning technique to reduce the computation overhead. Unfortunately, the $R \times W$ algorithm assumes that each data item has the same size. Hence, it is not suitable for variable-sized requests because response time alone is not a fair measure to evaluate different sized data. Acharya and Muthukrishnan [1] addressed the broadcast scheduling problem in heterogeneous environments, where data items have different sizes. The solution is based on a new metric called $Stretch$, defined as the ratio of the response time of a request to its service time. Based on stretch, they proposed a scheduling algorithm, called longest total stretch first (LTSF) to optimize the stretch and achieve a balance between the worst case and the average case. A straightforward implementation of LTSF is not practical for a large system, as at each broadcast time, the server has to recalculate the total stretch for every data item with pending requests in order to decide which data to broadcast next, and hence the scheduling algorithm becomes a bottleneck due to the high computation overhead.

In this paper, we propose a stretch optimal scheduling algorithm for on-demand broadcast systems based on the observation that LTSF is not optimal in terms of overall stretch. The nice thing of the proposed algorithm is that it is extremely simple and the computation overhead can be significantly reduced compared to LTSF. We present analytical studies as well as detailed simulation experiments to back up these claims. Simulation results show that our algorithm can significantly reduce

This work was supported in part by the National Science Foundation CAREER Award CCR-0092770.

the overall stretch and have a comparative worst-case system response time compared to existing scheduling algorithms.

The rest of the paper is organized as follows. Section II develops necessary background. In section III, we present the stretch optimal scheduling algorithm and provide analytical models to demonstrate its optimality in terms of stretch. Section IV evaluates the performance of the proposed algorithm. Section V concludes the paper.

II. Preliminaries

A. The System Model

Similar to previous work on broadcast scheduling [4], [10], [12], we assume that the environment consists of a number of clients that use uplink channels to make requests/demands of data items from the server. The server uses a downlink (broadcast) channel to broadcast the responses. There is a single broadcast channel that is monitored by all clients and that the channel is fully dedicated to the data broadcast. When a client needs a data item that it cannot find locally, it sends a request to the server. These requests are queued up at the server upon arrival. Based on the scheduling algorithm, the server repeatedly chooses an item based on these requests, broadcasts it over the satellite link, and removes the associated requests form the queue. Clients continuously monitor the broadcast channel after they make a request. We do not consider the effects of transmission errors, and hence, when an item is broadcasted, it is received by all clients that are waiting for it.

B. Performance Metrics

The popular performance measure is the response time of a request, i.e., the time between sending a request and receiving the reply by a client. In heterogeneous settings, response time alone is not a fair measure given that the individual requests significantly differ from one another in their service time. In this paper, we adopt an alternate performance measure, namely the stretch of a request, defined to be *the ratio of the response time of a request to its service time*. Note that the service time is the time to complete the request if it were the only job in the system. The rationale for this choice is based on our intuition; i.e., clients with larger jobs should be expected to be in the system longer than those with smaller requests. The drawback of minimizing response time for heterogeneous workloads is that it tends to improve system performance for large jobs (since they contribute the most to the response time). Minimizing stretch, on the other hand, is more fair for all job sizes.

We use the average stretch to measure the overall system performance. Since it is important to ensure that the scheduling algorithm does not induce starvation of requests for unpopular items, we also measure the worst-case wait time, which is the maximum amount of time that a client request waits in the service queue before being served. To make a scheduling decision, there is a *decision overhead*. When the database size is very large, the decision overhead may be significant. As a result, some broadcast bandwidth may be wasted since the server spends a lot of time on making scheduling decisions. Thus, we also measure the decision overhead.

III. The Stretch Optimal Scheduling Algorithm

A. An Example

In LTSF, the server maintains a queue Q_i for each data item i. Since there may be multiple requests for the same item i, let $t_{k,i}$ denote the arrival time of the k^{th} request of item i. s_i denote the data size of item i, and B denote the broadcast bandwidth. Suppose the current time is t. Broadcasting item i at time t has the following stretch.

$$Stretch(i) = \sum_{k \in Q_i} \frac{t + \frac{s_i}{B} - t_{k,i}}{\frac{s_i}{B}} \quad (1)$$

The LTSF algorithm selects the data item with the maximum stretch (based on Equation 1) to broadcast. Although it is intuitively correct in a point-point communication environment, it may not be optimal in an on-demand broadcasting environment. For example, suppose a database has two data items such that $s_1 = 1$, $s_2 = 2$. Assume that the request queues of these items are: $Q_1 = \{t_{1,1} = 8\}$, $Q_2 = \{t_{1,2} = 5, t_{2,2} = 9\}$. For simplicity, let $B = 1$. At time $t = 10$, the server needs to decide which item to broadcast first. If the LTSF algorithm is used:

$Stretch(1) = \frac{10+1-8}{1} = 3$

$Stretch(2) = \frac{10+2-5}{2} + \frac{10+2-9}{2} = 5$

As a result, the second item is broadcasted first, and then the first item is broadcasted at $t = 12$ (broadcasting the second item needs 2 time units). Then, the overall stretch is:

$5 + \frac{12+1-8}{1} = 10$

However, if the first item is broadcasted first, and then the second item is broadcasted at $t = 11$. Then, the overall stretch is:

$3 + \frac{11+2-5}{2} + \frac{11+2-9}{2} = 9$

Thus, the LTSF algorithm cannot minimize the overall stretch.

B. The Stretch Optimal Scheduling Algorithm

Before presenting our stretch optimal scheduling algorithm, we first introduce a broadcast scheduling function for item i:

$$S(i) = \sum_{k \in Q_i} \frac{1}{s_i^2} \quad (2)$$

Theorem 1: The scheduling algorithm, which selects the item with maximum $S(i)$ (Equation 2) to broadcast, can minimize the overall stretch.

Proof. We give the sketch of the proof. Based on Equation 1, for any two data items i and j, the scheduling algorithm should determine to broadcast which one first. If the server broadcasts item i first, the overall Stretch is:

$$Stretch_{i \to j} = \sum_{k \in Q_i} \frac{t + \frac{s_i}{B} - t_{k,i}}{\frac{s_i}{B}} + \sum_{k \in Q_j} \frac{t + \frac{s_i}{B} + \frac{s_j}{B} - t_{k,j}}{\frac{s_j}{B}}$$

Similarly, if item j is broadcasted first, the overall Stretch is:

$$Stretch_{j \to i} = \sum_{k \in Q_j} \frac{t + \frac{s_j}{B} - t_{k,j}}{\frac{s_j}{B}} + \sum_{k \in Q_i} \frac{t + \frac{s_j}{B} + \frac{s_i}{B} - t_{k,i}}{\frac{s_i}{B}}$$

We should minimize the overall stretch. Let

$$Stretch_{i\to j} - Stretch_{j\to i} = \sum_{k\in Q_j} \frac{s_i}{s_j} - \sum_{k\in Q_i} \frac{s_j}{s_i}$$

To minimize the overall stretch, if $Stretch_{i\to j} - Stretch_{j\to i} < 0$, item i should be broadcasted first; otherwise, item j should be broadcasted first. Thus, in order to broadcast item i first,

$$Stretch_{i\to j} - Stretch_{j\to i} < 0 \iff \sum_{k\in Q_j} \frac{s_i}{s_j} < \sum_{k\in Q_i} \frac{s_j}{s_i}$$
$$\iff \sum_{k\in Q_j} \frac{1}{s_j^2} < \sum_{k\in Q_i} \frac{1}{s_i^2}$$

To minimize the overall stretch, item i is broadcasted when $\sum_{k\in Q_i} \frac{1}{s_i^2}$ has the maximum value. □

An example: Suppose a database has three data items such that $s_1 = 1$, $s_2 = 2$, and $s_3 = 3$. Assume that the request queues of these items are: $Q_1 = \{t_{1,1} = 8\}$, $Q_2 = \{t_{1,2} = 9, t_{2,2} = 9, t_{3,2} = 9, t_{4,2} = 9, t_{5,2} = 9\}$, $Q_3 = \{t_{1,3} = 2, t_{2,3} = 9\}$. At time $t = 10$, the server needs to decide which item to broadcast first. If Theorem 1 is followed, since $S(1) = 1$, $S(2) = \frac{5}{4}$, and $S(3) = \frac{2}{9}$, the data broadcasting order is 2-1-3, which has an average stretch of 1.81. The scheduling order of other algorithms and their average stretch costs are listed in Table I.

TABLE I
AN EXAMPLE

Scheduling algorithm	Broadcasting Order	Average Stretch
SSTF	1-2-3	1.875
	1-3-2	2.62
Our	2-1-3	1.81
MRF	2-3-1	2.1
FCFS	3-1-2	2.94
LWT, RxW, LTSF	3-2-1	2.88

The algorithm: Based on Equation 2, the overall stretch can be optimized. However, due to the removal of the time parameter in Equation 2, some cold data items may never be served, and the algorithm may suffer from the starvation problem. To address this problem, we add a scheduling deadline rule. When the first request for a data item i arrives, a scheduling deadline is assigned to this pending request. When the deadline passes, the data item has the highest priority to be broadcasted. The stretch optimal algorithm is as follows.

1. Scan the request queue of each data item to check if any request has reached the scheduling deadline. If the request for item i has reached the scheduling deadline, let $v = i$ and go to Step 3 directly. If multiple entries have passed their scheduling deadline, use their $S(i)$ to break the tie.
2. Scan the request queue of each data item to find the data item with maximum $S(i)$, and set $v = i$. If multiple entries have the same $S(i)$, use their request arrival time to break the tie.
3. Broadcast item v.

Even with the tie breaking rules, it is possible that multiple candidates have passed their scheduling deadline, and they have the same S value. In this case, the server just randomly picks one. To implement the algorithm, a priority queue can be used to track the deadline, and a heap can be used to track the function of $S(i)$. Hence, the scheduling overhead is $O(\log N)$, where N is number of nodes in the heap. Compared to the decision overhead of LTSF, which is $O(N)$, our scheduling algorithm can significantly reduce the decision overhead.

IV. PERFORMANCE EVALUATION

A. Simulation Model

In our simulation model, Each client is simulated by a process which generates a stream of queries. Clients send requests to the server whenever the query is generated. The aggregated stream of requests, generated by the clients, follows a Poisson process with mean λ. The density function for the inter-arrival times of accessing item i is:

$$f(t) = \lambda_i e^{-\lambda_i t}$$

where $\lambda_i = p_i * \lambda$, p_i denotes the probability of clients requesting for data item i. The access pattern follows $Zipf$ distribution (similar assumptions are made by other researchers as well [2], [5], [8]). In the $Zipf$ distribution, the access probability of the i^{th} data item is represented as follows.

$$P_i = \frac{1}{i^\theta \sum_{i=1}^{n} \frac{1}{i^\theta}}$$

where $0 \le \theta \le 1$, n is the database size. When $\theta = 1$, it is the strict Zipf distribution. When $\theta = 0$, it becomes the uniform distribution. The data item size randomly distributed between s_{min} and s_{max}. Most of the system parameters and their default values are listed in Table II.

TABLE II
DEFAULT SYSTEM PARAMETERS

Database items	2000 items
Number of clients	200
Broadcast bandwidth (B)	115Kbps
The minimum data item size (S_{min})	1KB
The data item size ratio $R = \frac{S_{max}}{S_{min}}$	100
Mean query generate rate λ	$\frac{1}{100s}$
Zipf distribution parameter θ	0.6

B. The Effects of the Access Pattern

Figure 1 shows the average stretch as a function of the access skew coefficient θ. As can be seen, our algorithm has the lowest average stretch. Since LTSF also considers stretch when making scheduling decisions, it has similar performance to ours. Since FCFS, LWT, MRF, and $R \times W$ do not consider stretch in

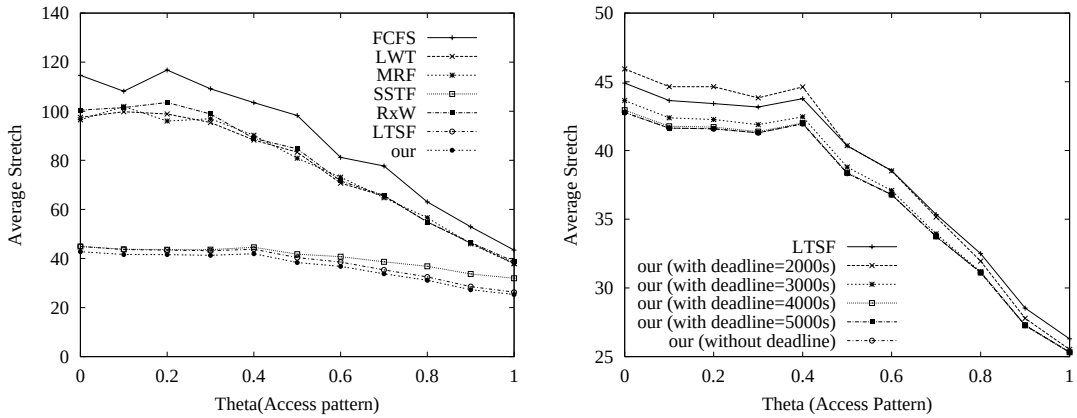

Fig. 1. The average stretch under different access patterns (random distribution)

broadcast scheduling, their average stretches are much higher than our algorithm and the LTSF algorithm. Note that SSTF has similar average stretch to our algorithm, since it broadcasts data in the order of data service time, which reflects the essence of the stretch-based algorithms.

Generally speaking, in order to reduce the average stretch, the server should try to schedule data items with small data size and high popularity. When $\theta = 0$, the access pattern is uniformly distributed, and then data items have similar popularity. As a result, the data size is the major factor. Since FCFS, LWT, MRF, and $R \times W$ do not consider the data size, they have much worse performance compared to our algorithm. The SSTF, however, considers the data size as major factor when making scheduling decisions, and it has similar average stretch to our algorithm when $\theta = 0$. When $\theta = 1$, the access pattern is much more skewed, and the data popularity becomes a major performance factor. Since FCFS, LWT, MRF, and $R \times W$ consider the data popularity when making scheduling decisions, the performance difference between these algorithms and our algorithm becomes much smaller as $\theta = 1$. The SSTF algorithm, however, does not consider the popularity when making scheduling decisions, and then its average stretch becomes much worse than our algorithm when $\theta = 1$.

The right graph of Figure 1 also shows the performance of our algorithm with different deadlines. It is easy to see that the average stretch reduces as the scheduling deadline increases. As expected, the worst-case waiting time decreases as the scheduling deadline increases (as shown in Figure 2). As shown in Figure 2, our algorithm has a comparative worst-case waiting time even without scheduling deadline. When the scheduling deadline is 2000, it has the best worst-case waiting time compared to other scheduling algorithms.

C. The Effects of the System Work Load

Figure 3 shows the average stretch as a function of the mean query generate rate, the database size, and the number of clients. Three figures have similar trends; that is, as the work load (in terms of query generate rate, database size, or number of clients) increases, the average stretch also increases. From the figure, we can see that our algorithm outperforms other scheduling algorithms in various scenarios.

D. The Scheduling Overhead

Figure 4 shows the scheduling overhead of the LTSF algorithm and our algorithm. In LTSF, the server has to recalculate the total stretch for every data item with pending requests in order to decide which data item to broadcast next, and hence the scheduling algorithm becomes a bottleneck due to its high computation overhead. By using heaps, the computation complexity of our algorithm can be reduced to $O(\log N)$. Results from Figure 4 also verifies that our algorithm can significantly reduce the computation overhead compared to the LTSF algorithm. Moreover, in the LTSF algorithm, the computation overhead increases linearly when the database size or the number of clients increases, whereas the computation overhead in our algorithm increases much slower, and then our algorithm is more scalable compared to the LTSF algorithm.

V. Conclusions

This paper has focused on the challenges of large-scale on-demand data broadcasts introduced by broadcast media such as satellite networks, cable networks, and cellular networks. In such an environment, the scheduling problem is different from the point-to-point communication environment and the push-based broadcast environment. Moreover, when variable-sized heterogeneous requests are considered, most of the previous scheduling algorithm fail to perform well. As stretch is widely adopted as a performance metric for variable-size data requests, we proposed a broadcast scheduling algorithm to optimize the system performance in terms of stretch. The nice thing of the proposed algorithm is that it is extremely simple and the computation overhead is very low. Analytical results described the intrinsic behavior of the algorithm. Simulation results demonstrated that our algorithm significantly outperforms existing scheduling algorithms under various scenarios.

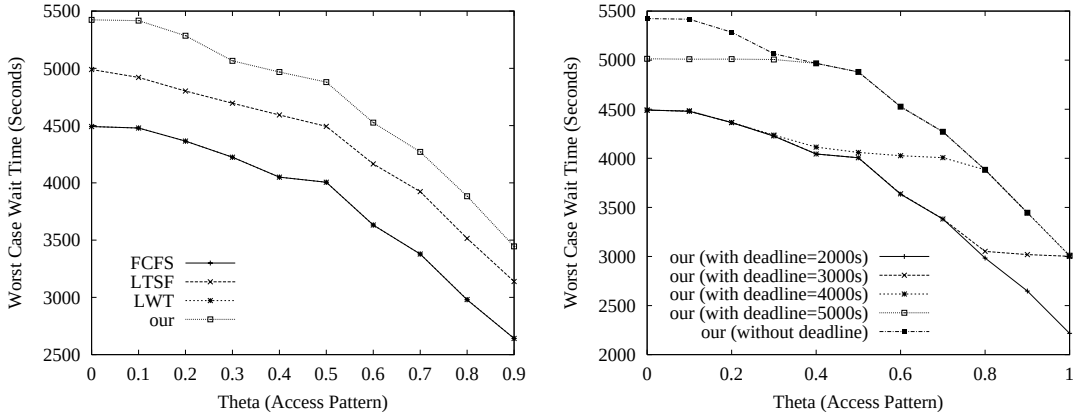

Fig. 2. The worst waiting time under different access patterns

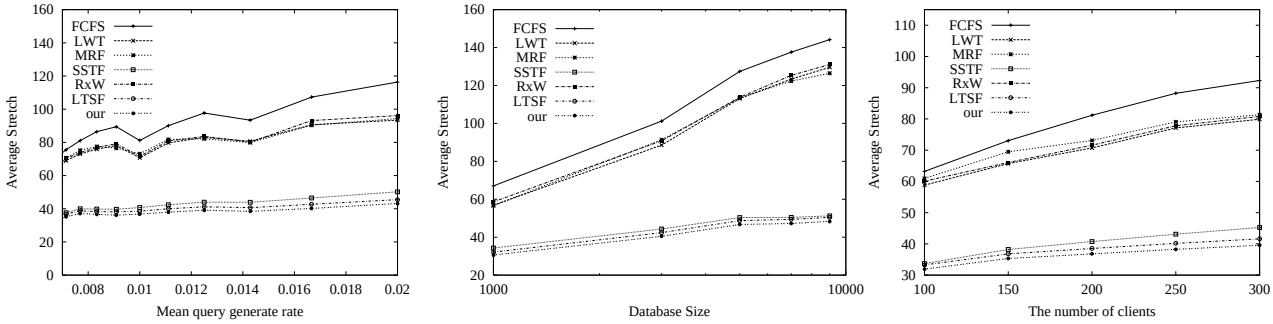

Fig. 3. The average stretch under different system work load

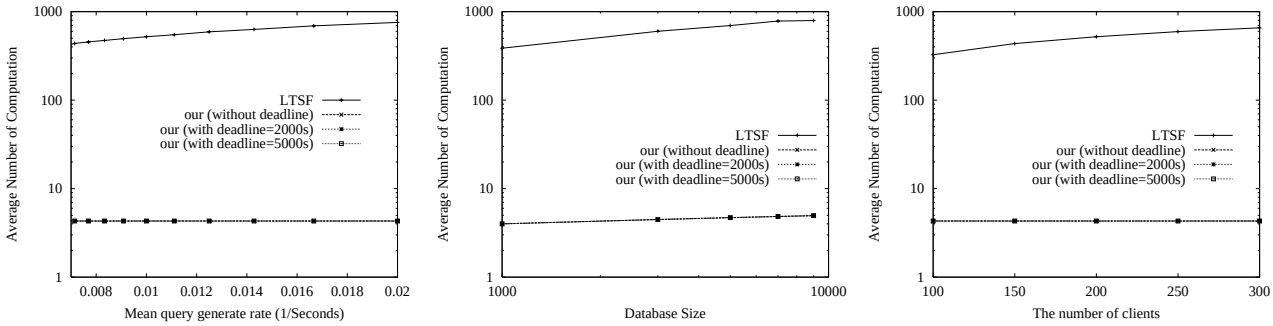

Fig. 4. The scheduling overhead under different system work load.

REFERENCES

[1] S. Acharya and S. Muthukrishnan, "Scheduling On-Demand Broadcasts: New Metrics and Algorithms," *ACM MobiCom'98*, pp. 43–54, Oct. 1998.

[2] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks," *IEEE Personal Communications*, pp. 50–60, Dec. 1995.

[3] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks: Data Management for Asymmetric Communication Environments," *Proc. ACM SIGMOD*, pp. 199–210, May 1995.

[4] D. Aksoy and M. Franklin, "RxW: A Scheduling Approach for Large-Scale On-Demand Data Broadcast," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, Dec. 1999.

[5] M. Ammar and J. Wong, "On the Optimality of Cyclic Transmission in Teletext Systems," *IEEE Transactions on Communications*, pp. 8–87, Jan. 1987.

[6] A. Datta, D. Vandermeer, A. Celik, and V. Kumar, "Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users," *ACM Transactions on Database Systems*, vol. 24, no. 1, pp. 1–79, March 1999.

[7] H. Dykeman, M. H. Ammar, and J. Wong, "Scheduling Algorithms for Videotext Systems under Broadcast Delivery," *Proc. International Conference of Communications*, pp. 1847–1851, 1996.

[8] S. Hameed and N. Vaidya, "Efficient Algorithms for Scheduling Data Broadcast," *ACM/Baltzer Wireless Networks (WINET)*, pp. 183–193, May 1999.

[9] T. Imielinski, S. Viswanathan, and B. Badrinath, "Data on Air: Organization and Access," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 3, pp. 353–372, May/June 1997.

[10] C. Su and L. Tassiulas, "Broadcast Scheduling for Information Distribution," *IEEE INFOCOM*, pp. 107–117, 1997.

[11] N. Vaidya and S. Hameed, "Scheduling Data Broadcast in Asymmetric Communication Environments," *ACM/Baltzer Wireless Networks (WINET)*, pp. 171–182, May 1999.

[12] J.W. Wong, "Broadcast Delivery," *Proceedings of the IEEE*, vol. 76, no. 12, pp. 1566–1577, Dec. 1988.