Data Broadcast Scheduling : On-line and Off-line Algorithms*

Nitin H. Vaidya Sohail Hameed
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
E-mail: {vaidya,shameed}@cs.tamu.edu
Phone: (409) 845-0512
FAX: (409) 847-8578

Technical Report 96-017 †

July 1996

Abstract

With the increasing popularity of portable wireless computers, mechanisms to efficiently transmit information to such *clients* are of significant interest. The environment under consideration is *asymmetric* in that the information server has much more bandwidth available, as compared to the clients. In such environments, often it is not possible (or not desirable) for the clients to send explicit requests to the server. It has been proposed that in such systems the server should broadcast the data periodically. One challenge in implementing this solution is to determine the *schedule* for broadcasting the data, such that the wait encountered by the clients is minimized. A *broadcast schedule* determines what is broadcast by the server and when. In this report, we present three algorithms for determining broadcast schedules that minimize the wait time. Simulation results are presented to demonstrate that our algorithms perform well. Variations of our algorithms for environments subject to errors, and systems where different clients may listen to different number of broadcast channels are also considered.

*Research reported is supported in part by Texas Advanced Technology Program grant 009741-052-C.
†This report supersedes most material in report 96-012 [13] and contains more recent results.

Contents

1 Introduction

Mobile computing and wireless networks are fast-growing technologies that are making ubiquitous computing a reality. With the increasing popularity of portable wireless computers, mechanisms to efficiently transmit information to such *clients* are of significant interest. For instance, such mechanisms could be used by a *satellite* or *base station* to communicate information of common interest (e.g., stock quotes, sports scores, etc.) to wireless hosts. Approaches for determining what to transmit and when, is the subject of this report. In the environment under consideration, the *downstream* communication capacity, from server to clients, is relatively much greater than the *upstream* communication capacity, from clients to server. Such environments are, hence, called *asymmetric* communication environments [2]. In an asymmetric environment, *broadcasting* the information is an effective way of making the information available simultaneously to a large number of users. For asymmetric environment, researchers have previously proposed algorithms for designing *broadcast schedules* [4, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16]. Two metrics are used to evaluate these algorithms:

- Access time: This is the amount of time a client has to wait for some information that it needs. It is important to minimize the *access time* so as to decrease the idle time at the client. Several researchers have considered the problem of minimizing the access time [4, 6, 10, 11, 12, 7, 3, 2, 15, 16]
- Tuning time: This is the amount of time a client must *listen* to the broadcast until it receives the information it needs. It is important to minimize the *tuning time*, because the power consumption of a wireless client is higher when it is *listening* to the transmissions, as compared to when it is in a *doze* mode. Previous work has tried to minimize the tuning time by providing the clients with an *index* that provides hints to the client of when the required data is scheduled to be broadcast [9, 10, 11, 14]. Without this information, the client must listen continually until the necessary information is received.

This report presents an approach to minimize the *access time*. Our results can be combined with the schemes for reducing *tuning time* to obtain a comprehensive solution. We consider a database that is divided into *information items* (or *items* for short). Thus, a broadcast schedule specifies when each item is to be transmitted. Contributions of this report are as follow:

- **Square-root Rule:** We show that the access time is minimized when the frequency of an item (in the broadcast schedule) is proportional to the *square-root* of its demand (characterized as *demand probability*) and inversely proportional to *square-root* of its length. This result is a generalized version of a result presented in [4, 15].
- We present three algorithms for scheduling broadcasts, based on the above *square-root rule*.
  - The first algorithm is an "on-line" algorithm that can be used by a server to determine which item to transmit next [13]. We demonstrate that the proposed on-line algorithm achieves performance comparable to an optimal off-line algorithm. Also, the proposed on-line algorithm reduces the average access time by a factor of 2, as compared to a previously proposed *on-line* algorithm [14].

    On-line algorithms are of significant interest as they are easy to adapt to time-varying demands for the data items.

- The second algorithm is also an on-line algorithm obtained by modifying the first algorithm. An on-line algorithm must use some *decision-making mechanism* to determine which information item is to be transmitted next. For the first on-line algorithm, the time complexity of the decision mechanism is linear in the number of information items in the database. The second algorithm reduces this complexity significantly but may result in somewhat poorer performance than the first algorithm. This algorithm provides the ability to trade performance with time complexity.
- The third algorithm generates cyclic schedule for a *specified* cycle size. This algorithm is off-line in that it *a priori* produces an entire schedule, which is then used repeatedly.

The ideas presented here can be combined with previous work [10, 14] to minimize *tuning* time as well as *access* time.

- Impact of *errors* on the scheduling policy is evaluated. In an asymmetric environment, when a client receives a data item containing errors (due to some environmental disturbance), it is not always possible for the client to request retransmission of the data. In this case, the client must wait for the next transmission of the required data item. We evaluate how optimal broadcast frequencies of the items are affected in presence of errors [13].

- We consider systems where different clients may listen to different number of broadcast channels, depending on how many they can afford. In such an environment, the schedules on different broadcast channels should be designed so as to minimize the access time for most (if not all) clients. This report presents preliminary results on this problem.

The rest of the report is organized as follows. Section 2 introduces some terminology. Section 3 derives the *square-root* rule, and presents our broadcast scheduling algorithms. The impact of errors is analyzed in Section 4. Section 5 considers an environment where different clients may be listening to different number of channels (depending on what they can afford). Section 6 evaluates the performance of our schemes. Section 7 briefly discusses how our schemes can be made adaptive. Related work is summarized in Section 8. A summary is presented in Section 9.

2 Preliminaries

This section introduces much of the terminology and notations to be used in rest of the report.

- Database at the server is assumed to be divided into many *information items*. The items are not necessarily of the same size.

- The time required to broadcast an item of unit length is referred to as one *time unit*. Hence time required to broadcast an item of length l is l time units. Note that unit of length and time unit may be used interchangeably because of the way they are defined.

- l_i represents length of item i.

- To develop a theoretical foundation for our algorithms, we assume that the broadcast consists of cycle of size N time units, The results presented in the report also apply to *non-cyclic* schedules (for non-cyclic schedules, effectively, $N \rightarrow \infty$).
- M = total number of information items in the server's database. The items are numbered 1 through M.
- *Instance of an item* : An appearance of an item in the broadcast cycle is referred to as an *instance* of the item. There may be multiple instances of an item in the cycle.
- *Schedule* : *Schedule* for the broadcast cycle is an order of the items in the cycle.
- *Frequency of an item* : *frequency* f_i of item i is the number of instances of item i in the broadcast cycle. The f_i instances of an item are numbered 1 through f_i. Size of the cycle is, therefore, given by $N = \sum_{i=1}^{M} f_i l_i$, where l_i is the length of item i.
- *Spacing* : The *spacing* between two instances of an item is the time it takes to broadcast information from the beginning of the first instance to the beginning of the second instance. s_{ij} denotes the spacing between j-th instance of item i and the next instance of item i $(1 \leq j \leq f_i)$. Note that, after the f_i-th instance of an item in a transmission of the broadcast cycle, the next instance of the same item is the *first* instance in the next transmission of the broadcast cycle.

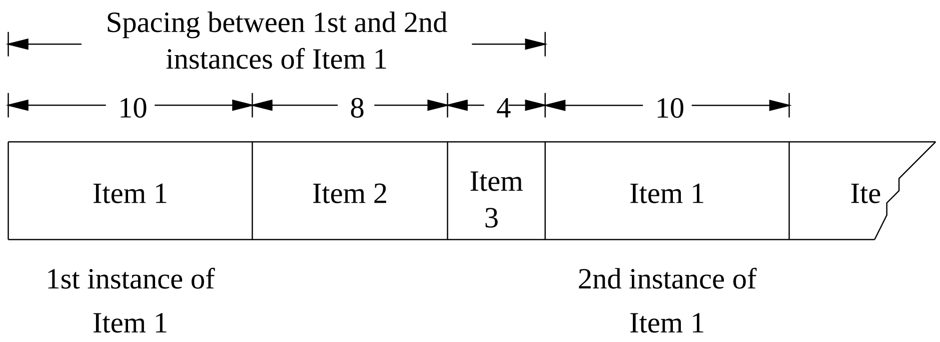


Figure 1: Showing a part of broadcast cycle (Example 1)

Example 1: As an example, refer to Figure 1. The figure shows a part of a broadcast cycle, which contains two instances of *item 1*, and one instance each of *items 2* and *3*. The lengths of the items are 10, 8 and 4 time units respectively. The spacing between the two instances of item 1 is the time from the beginning of first instance of item 1 until the beginning of second instance, which is equal to $10 + 8 + 4 = 22$ time units. Thus, if a client needs item 1 some time (uniformly distributed) between the two instances of item 1, then the average wait is $22/2 = 11$ time units. To reduce this wait, item 1 will have to be transmitted sooner, however, doing so will require one of items 2 or 3 to be transmitted later, causing an increase in the access time of a client needing that item. This example illustrates the need for appropriate scheduling of items in the broadcast. $\square$

- *Item Mean Access Time* : *Item Mean Access Time* of item i, denoted t_i, is defined as the average wait by a client needing item i until it starts receiving item i from the server. Provided that a client is equally likely to need an item i at any instant of time (uniform distribution), t_i can be obtained as,

$$\begin{aligned} t_i &= \sum_{j=1}^{f_i} \frac{s_{ij}}{2} \frac{s_{ij}}{N} = \frac{1}{2} \sum_{j=1}^{f_i} \frac{s_{ij}^2}{N} \\ \text{where} \quad N &= \sum_{i=1}^{M} f_i l_i \end{aligned}$$

  If all the f_i instances of item i are equally spaced, that is, for some constant s_i, $s_{ij} = s_i$ $(1 \le j \le f_i)$, then, it follows that,

$$s_i = \frac{N}{f_i}$$

  In this case, the expression for t_i can be simplified as follows:

$$t_i = \frac{1}{2} \sum_{j=1}^{f_i} \frac{s_{ij}^2}{N} = \frac{1}{2} \sum_{j=1}^{f_i} \frac{s_i^2}{N} = \frac{1}{2}\, f_i \left(\frac{s_i^2}{N} \right) = \frac{1}{2} s_i \quad \text{as } s_i = N/f_i \tag{1}$$

- *Demand probability* : *Demand probability* p_i denotes the probability that an item needed by a client is item i.

- *Overall Mean Access Time* : *Overall Mean Access Time*, denoted t, is defined as the average wait encountered by a client (averaged over all items). Thus,

$$t = \sum_{i=1}^{M} t_i\, p_i = \sum_{i=1}^{M} \left(\frac{1}{2} \sum_{j=1}^{f_i} \frac{s_{ij}^2}{N} \right) p_i$$

  When $s_{ij} = s_i$ $(1 \le j \le f_i)$, that is, when all items are distributed in the cycle with equal spacing, the above equation reduces to

$$t = \frac{1}{2} \sum_{i=1}^{M} s_i\, p_i \tag{2}$$

<u>Note:</u>

All results presented here remain valid if p_i in the above expression is replaced by w_i, where w_i may be interpreted as *weight* of item i. The weights of all items do not have to add to 1 (the fact that the p_i's add to 1 does not have any impact on our algorithms). For instance, weight w_i may be obtained as a product of the "priority" of item i and the demand probability of item i. When p_i is replaced by w_i, t is interpreted as a cost metric.

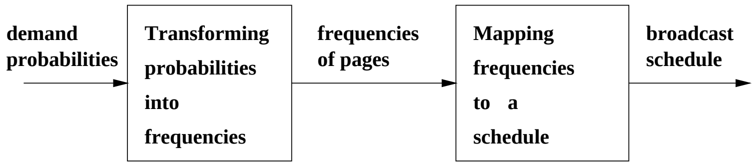


Figure 2: Constructing a Broadcast Cycle

3 Proposed Scheduling Schemes

Figure 2 depicts an abstract view of the procedure for constructing a broadcast schedule. The first block in Figure 2 maps the demand probability distribution into "optimal" item frequencies. Recall that frequency of an item is the number of times the item is to be broadcast in a cycle. Having determined the optimal frequencies, second block in Figure 2 uses the frequencies to determine the broadcast schedule. Our goal is to perform the functions of the two blocks in such a way that *overall mean access time*, t, is minimized. Note that Figure 2 gives a low-level abstraction of the procedure. This helps in obtaining an expression for *Optimal Overall Mean Access Time*. Algorithms presented here do not use this two-step procedure, however, they are formulated based on the results obtained in this discussion.

3.1 Mapping Demand to Frequencies

We first present theoretical results that motivate our scheduling schemes. The first observation stated in Lemma 1 below is intuitive. This observation also follows from a result presented in [12], and has been implicitly used by others (e.g., [3, 4, 15]).

Lemma 1 *The broadcast schedule with minimum* overall mean access time *results when the instances of each item are equally spaced.*

Proof of the lemma is omitted here for brevity. In reality, it is not always possible to space instances of an item equally. However, the above lemma provides a basis to determine a lower bound on achievable overall mean access time. Note that, while Lemma 1 suggests that spacing between consecutive instances of item i should be constant, say s_i, s_i need not be identical to the spacing s_j between instances of another item j.

The objective in this section is to determine the optimal frequencies (f_i's) as a function of the probability distribution (p_i's) and the length distribution (l_i's). We assume the ideal situation, as implied by Lemma 1, where instances of all items can be equally spaced. This assumption, although often difficult to implement, does lead to a useful result that the minimum possible *overall mean access time* is achieved when

$$f_i \propto \frac{\sqrt{p_i}}{\sqrt{l_i}}$$

Theorem 1 Square-root Rule: *Given the demand probability p_i of each item i, the minimum* overall mean access time*, t, is achieved when frequency f_i of each item i is proportional to $\sqrt{p_i}$ and* *inversely proportional to* $\sqrt{l_i}$*, assuming that instances of each item are equally spaced. That is,*

$$f_i \propto \frac{\sqrt{p_i}}{\sqrt{l_i}}$$

To put it differently, for items i *and* j*,*

$$\frac{f_i}{f_j} = \sqrt{\frac{p_i}{p_j}}\,\sqrt{\frac{l_j}{l_i}}$$

Proof: Appendix A presents the proof. □

A special case of Theorem 1 when all items are of identical size was derived in [4, 15]

For cycle size N, $\sum_{j=1}^{M} f_j l_j = N$. Therefore, the above theorem implies that, $f_i = \left(N\sqrt{p_i/l_i}\right) / \left(\sum_{j=1}^{M}\sqrt{p_j l_j}\right)$. Also, as spacing $s_i = N/f_i$, a consequence of the above result is that, for *overall mean access time* to be minimized, we need

$$s_i \propto \frac{\sqrt{l_i}}{\sqrt{p_i}}$$

As shown in Appendix A, from Theorem 1 it follows that, the optimal *overall mean access time*, named t_{optimal}, is:

$$t_{\text{optimal}} = \frac{1}{2}\left(\sum_{i=1}^{M}\sqrt{p_i l_i}\right)^2 \qquad (3)$$

t_{optimal} represents a *lower bound* on achievable overall mean access time. As the lower bound is derived assuming that instances of each item are equally spaced, the bound, in general, is not achievable. However, as shown later, it is possible to achieve performance almost identical to the above lower bound.

Now we present three scheduling algorithms. The first "on-line" algorithm determines which item should be broadcast next by the server. The second on-line algorithm distributes the items into different "buckets" and applies the first algorithm to these buckets instead of individual items, thereby reducing the computation cost. The third algorithm uses the first algorithm to generate a broadcast cycle with length approximately equal to a given size N.

3.2 On-line Scheduling Algorithm

Whenever the server is ready to transmit a new item, it calls the *on-line* algorithm presented here [13]. The on-line algorithm determines the item to be transmitted next using a *decision rule* – this decision rule is motivated by the result obtained in Theorem 1. As noted previously, Theorem 1 implies that, for optimal performance, instances of an item i should be equally spaced with spacing s_i, where

$$s_i \propto \frac{\sqrt{l_i}}{\sqrt{p_i}}$$

This can be rewritten as

$$s_i^2 \frac{p_i}{l_i} = \text{constant} \tag{4}$$

The above observation is used in our algorithm, as presented below. In the following, let Q denote the current time and $R(j)$ denote the time at which an instance of item j was most recently transmitted. $R(j)$ is initialized to -1,[1] for all j, and Q is initialized to 0. Note that, $R(j)$ is updated whenever item j is transmitted.

ON-LINE algorithm:

`broadcast item` i `at time` Q
`if`

$$(Q - R(i))^2 \; p_i/l_i \;\; \geq \;\; (Q - R(j))^2 \; p_j/l_j, \quad 1 \leq j \leq M$$

(if there is a tie, that is, if the above condition is satisfied for more than one item, any one of those items may be chosen arbitrarily.) □

$Q - R(i)$ is the spacing between the current time, and the time at which item i was previously transmitted. Note that, the term

$$(Q - R(i))^2 \; \frac{p_i}{l_i}$$

is similar to the term $s_i^2 \, p_i/l_i$ in Equation 4 above. The motivation behind our algorithm is to attempt to achieve this equality, to the extent possible.

Example 2: Consider a database containing 3 items such that $p_1 = 1/2$, $p_2 = 3/8$, and $p_3 = 1/8$. Assume that items have lengths $l_1 = 1$, $l_2 = 2$ and $l_3 = 4$ time units. Figure 3 shows the items recently broadcast by the server (up to time < 100). The above *on-line* algorithm is called to determine the item to be transmitted at time 100. Thus, $Q = 100$. Also, from Figure 3, observe that $R(1) = 95$, $R(2) = 93$, and $R(3) = 96$. The *on-line* algorithm evaluates the term $(Q - R(j))^2 p_j/l_j$ for $j = 1, 2, 3$ as 12.5, 147/16 (=9.1875) and 0.5, respectively. As this term is the largest for item 1, item 1 is transmitted at time 100. □

Performance measurements for the above algorithm are presented in Section 6. Our algorithm improves performance by a factor of 2 as compared to the *probabilistic* on-line algorithms presented in [14, 15].

While, in general, the proposed on-line algorithm performs close to optimal, it is possible to construct examples where the schedule produced by the algorithm is not exactly optimal.

[1]The choice of initial value will not affect the mean access time much, unless the broadcast is for a very short time. For broadcasts that last a short time, other initial values may perform better. For instance, $R(j)$ may be initialized to $-\sqrt{l_j}$.

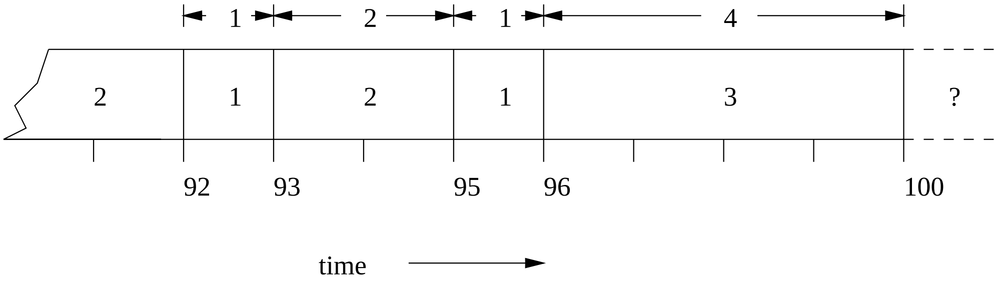


Figure 3: Illustration of the on-line algorithm (Example 2)

Example 3: Consider the following parameters: $M = 2$, $l_1 = l_2 = 1$, $p_1 = 0.2 + \epsilon$, $p_2 = 1 - p_1$, and $\epsilon < 0.05$. In this case, the on-line algorithm produces the cyclic schedule (1,2), i.e., 1,2,1,2,..., which achieves an overall mean access time of 1.0. On the other hand, the the cyclic schedule (1,2,2) achieves overall mean access time $2.9/3 + 2\epsilon/3 < 1$. Thus, in this case, the on-line algorithm is not optimal. However, the overall mean access time 1.0 of the on-line algorithm is within 3.5% of that achieved by the cyclic schedule (1,2,2). □

The on-line algorithm produces a cyclic schedule, if the ties are resolved deterministically. Appendix B presents an argument to show that the schedule produced by the on-line algorithm is cyclic. However, size of the cycle may be large, and hard to predict accurately.

3.3 On-line Algorithm with Bucketing

The main disadvantage of *On-line* Algorithm is the high computational cost of *O(M)* required to decide the next item to broadcast. This cost can be reduced by partitioning the database into *buckets* of items, as follows.

Suppose that we divide the database into k buckets, each bucket B_i contains m_i items, $m_i \geq 0$, such that $\sum_{i=1}^{k} m_i = M$, the total number of items in database. We maintain the items in each bucket in a cyclic queue. At any time, only items at the front of the buckets are candidates for broadcast at that time. Define $q_j = \sum_{i \epsilon B_j} p_i / m_j$ as the average demand probability of the items in bucket B_j, and $d_j = \sum_{i \epsilon B_j} l_i / m_j$ as the average length of the items in bucket B_j. Note that $\sum_{i=1}^{k} m_i q_i = 1$. Let Q be the current time and $R(i)$ be the time when item i was most recently broadcast. Let I_j denote the item at the front of Bucket B_j. As shown in Appendix C, for optimality, the following condition must hold when bucketing is used: If item i is in bucket B_j, then

$$s_i \propto \sqrt{d_j} / \sqrt{q_j} \qquad (5)$$

$$\text{and} \qquad f_i \propto \sqrt{q_j} / \sqrt{d_j} \qquad (6)$$

The on-line algorithm with bucketing is obtained from the above result.

ON-LINE WITH BUCKETING algorithm:

```
broadcast item $I_m$ at the front of bucket $B_m$
  if
```

$$(Q - R(I_m))^2\ q_m/d_m \ \geq\ (Q - R(I_j))^2\ q_j/d_j, \quad 1 \leq j \leq k$$

(if the condition is satisfied for more than one item,
any one of those items may be chosen arbitrarily.)

```
dequeue item $I_m$ from the front of the bucket $B_m$ and enqueue at the end □
```

The above algorithm is quite similar to the original *on-line* algorithm except that the decision rule is applied only to items at the front of each bucket. Hence, the algorithm needs to compare values for only k items giving the complexity of $O(k)$. Observe that all items within the same bucket are broadcast with the same frequency. This suggests that the p_i/l_i values of all items in any bucket should be close for good results.

The *Optimal Overall Mean Access Time* resulting from the above algorithm as shown in Appendix C, is given by

$$t_{\text{opt\_bucket}} = \frac{1}{2}\left(\sum_{j=1}^{k} m_j \sqrt{q_j d_j}\right)^2 \tag{7}$$

Similar to t_{optimal}, $t_{\text{opt\_bucket}}$ is a lower bound on performance achievable with bucketing.

The above equation shows that $t_{\text{opt\_bucket}}$ is dependent upon the selection of values for m_j's under the constraint that $\sum_{j=1}^{k} m_j = M$. Optimizing the bucketing scheme for a given number of buckets k requires that the m_j's be chosen appropriately, such that the above equation is minimized.

For our simulations, we use a heuristic to determine the membership of items to the buckets. The heuristic for determining the membership of an item i to a bucket B_j is as follows:

Let $R_{min} = \min_i \sqrt{p_i/l_i}$ and $R_{max} = \max_i \sqrt{p_i/l_i}$. Let $\delta = R_{max} - R_{min}$. If, for item i, $\sqrt{p_i/l_i} = R_{min}$, then item i is placed in bucket B_1. Any other item i is placed in bucket B_j $(1 \leq j \leq k)$ if $(j-1)\delta/k \ < \ (\sqrt{p_i/l_i} - R_{min}) \leq (j\,\delta/k)$. This is pictorially depicted in Figure 4. The above heuristic executes in $O(M)$ time.

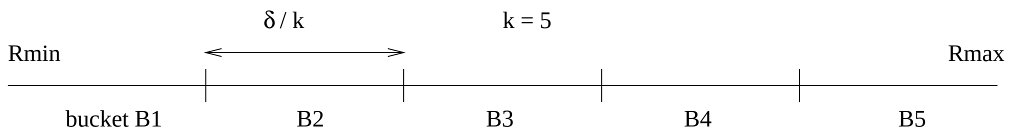


Figure 4: Heuristic for assigning items to k buckets: The interval (R_{min}, R_{max}) is divided into k equal-sized sub-intervals. An item i whose $\sqrt{p_i/l_i}$ value belongs to the j-th sub-interval is assigned to bucket B_j $(1 \leq j \leq k)$.

3.3.1 Comparison of Buckets and Multi-disk [3]

The notion of a *bucket* is similar to that of a *broadcast disk* in the multi-disk approach proposed by Acharya et al. [3]. Therefore, the result in Equation 6 can be used to determine suitable frequencies for the broadcast disks. The differences between the two approaches are as follows: (a) Our algorithm is on-line in that the broadcast schedule is not predetermined. This allows our algorithm to quickly react to any changes in parameters (such as demand probabilities). (b) The algorithm in [3] imposes a constraint that the instances of each item be equally spaced at the risk of introducing "holes" in the schedule (which may be filled with other information). Our algorithm does not create such holes. (c) Our algorithm works well with items of arbitrary sizes. [3] is constrained to fixed size items. (d) Acharya et al. do not have a way of determining the optimal frequencies for the different disks, whereas, our algorithm automatically tries to use the optimal frequencies.

3.4 Cyclic Scheduling Algorithm

We can use the *on-line* algorithm to generate *off-line* a schedule of size approximately equal to a specified size N. The schedule may then be used as a cycle and broadcast repeatedly. Here is a sketch of the algorithm:

```
while size of schedule < N
{
  generate the next item to schedule using On-line algorithm
  append it to the schedule
}
```

Example 4: Suppose there are 3 items in database. Their probabilities of access are $p_1 = 1/2, p_2 = 1/4$ and $p_3 = 1/4$ respectively and their lengths are $l_1 = 6, l_2 = 6$ and $l_3 = 8$ units respectively. Let the specified cycle size N be 80 time units.

Let $R(i)$ for $i = 1, 2, 3$ be initialized to -1. Start with current time $Q = 0$. The $(Q - R(i))^2 p_i/l_i$ values for the three items would be $0.08, 0.04$ and 0.03 respectively. Hence we select item 1 having maximum value, include it in our broadcast schedule and set $R(i)$ to current time $Q = 0$. Next we increment Q by the length of item 1, l_1, making $Q = 6$. We again determine the $(Q - R(i))^2 p_i/l_i$ term for $i = 1, 2, 3$ which come out to be $2.99, 2.04$ and 1.53 respectively. Hence we again select item 1 and append it to the schedule. We increment $R(i)$ to current value of time $Q = 6$ and then increment Q by the length of item 1 making $Q = 12$. We again determine $(Q - R(i))^2 p_i/l_i$ for $i = 1, 2, 3$ which come out to be $2.99, 7.04$ and 5.28 respectively and hence we select item 2 this time. The process is repeated until the size of schedule, Q equals or exceeds the specified value of $N = 80$ time units. The resulting schedule is shown in Figure 5. $\square$

It is evident that the size of the schedule from the above algorithm may end up in being greater than the given size N. However, the error is bounded by the size of the largest item in the database whose demand probability is non-zero.

1	1	2	1	3	2	1	3	2	1	3	2

Q = 0 6 12 18 24 32 38 44 52 58 64 72 80

Figure 5: Showing the broadcast cycle of Example 4. Note that initially the algorithm does not generate "sub-cycles", but after the initial *transients* are damped, the algorithm generates sub-cycles of items 3,2,1.

The *On-line* algorithm requires the knowledge about the time of last broadcast $R(i)$ of each item i to determine their spacing and hence the next item to broadcast. If $R(i)$ values are not properly initialized, it will take some iterations in the beginning to bring $R(i)$ values to *steady state*. Hence *On-line* algorithm shows a *transient* behavior at the beginning. The transients can have an adverse impact on the schedule, if the initial transient forms a significant fraction of the cyclic schedule.

One way of mitigating the effect of the transients could be to run the *On-line* algorithm to generate items (and discarding them) until each item is generated at least a specified number of times. This is the point when, hopefully, all the transients have been damped and now the above *Cyclic Scheduling Algorithm* may be used to generate the cyclic schedule.

It should be noted that the given size N should be large enough such that all the items with non-zero demand probabilities are scheduled at least once. Also, in general, larger N results in lower access time.

The main advantage of *Cyclic Scheduling Algorithm*, as in case of any off-line schedule, is that only a simple table look-up is needed each time the server needs to transmit an item (in contrast, the on-line algorithms presented earlier incur $O(M)$ or $O(k)$ overhead). For the off-line algorithm, initially, $O(MN)$ overhead is incurred in generating the cyclic schedule. The initial overhead is amortized over the period of time for which the schedule is used.

4 Effect of Transmission Errors on Scheduling Strategy

In the discussion above, we assumed that each item transmitted by the server is always received correctly by each client. As the wireless medium is subject to disturbances and failures, this assumption is not necessarily valid. Traditionally, in an environment that is subject to failures, the data is encoded using error control codes (ECC). These codes enable the client to "correct" some errors, that is, recover data in spite of the errors. However, ECC cannot correct large number of errors in the data. When such errors are detected (but cannot be corrected by the client), the server is typically requested to retransmit the data.

In the asymmetric environment under consideration here it is not always possible for the client to ask the server to retransmit the data.[2] In this section, we evaluate the impact of *uncorrectable* errors on the scheduling strategy for broadcasts [13].

[2]Even if it were possible for a client to send a retransmit request to the server, it is not clear that a broadcast scheme should allow such requests, because it is possible that many clients receive the original broadcast correctly, but only a few do not due to some localized disturbance.

Suppose that uncorrectable errors occur in an item of length l with probability $E(l)$.[3] Appendix D shows that the *overall mean access time*, t, for this case, assuming that instances of item i are equally spaced with spacing s_i, is given by

$$t \;=\; \sum_{i=1}^{M} s_i p_i \left(\frac{E(l_i)}{1 - E(l_i)} + \frac{1}{2} \right) \;=\; \frac{1}{2} \sum_{i=1}^{M} s_i p_i \left(\frac{1 + E(l_i)}{1 - E(l_i)} \right) \tag{8}$$

The *Square Root Rule* in Theorem 1 needs to be modified to take errors into account as follows :

Theorem 2 *Given that the probability of occurrence of uncorrectable errors in an item of length l is $E(l)$, the* overall mean access time *is minimized when*

$$f_i \propto \frac{\sqrt{p_i}}{\sqrt{l_i}} \left(\frac{E(l_i)}{1 - E(l_i)} + \frac{1}{2} \right)^{1/2}$$

and

$$s_i \propto p_i^{-1/2} \; l_i^{1/2} \left(\frac{E(l_i)}{1 - E(l_i)} + \frac{1}{2} \right)^{-1/2}$$

Proof : See Appendix D. □

The lower bound on overall mean access time now becomes,

$$t_{opt\_error} = \left(\sum_{i=1}^{M} \sqrt{p_i l_i} \left(\frac{E(l_i)}{1 - E(l_i)} + \frac{1}{2} \right)^{1/2} \right)^2$$

Equation 8 and Theorem 2 can be applied to many random error models.[4] For the purpose of demonstration, we now consider a simple error model. Let *uncorrectable* errors occur according to a Poisson process with rate λ per unit time. Thus, the probability that item i transmitted by the server will be received by a client without uncorrectable errors is $e^{-\lambda l_i}$. If the item contains uncorrectable errors (with probability $E(l_i) = 1 - e^{-\lambda l_i}$), the item is discarded by the client, and the client cannot use that instance of the item. The client must wait for the next instance of the item.

Substituting $E(l_i) = 1 - e^{-\lambda l_i}$ in Equation 8, and simplifying yields

$$t = \sum_{i=1}^{M} s_i \, p_i \left(e^{\lambda l_i} - \frac{1}{2} \right)$$

Similarly using Theorem 2, the optimal frequency f_i for an item i should be

[3] Now, l_i denotes length of item i after encoding with an error control code.

[4] Burst error models need somewhat different analysis. As burst errors are likely in wireless channels, this is an important subject for future research.

$$f_i \propto \frac{\sqrt{p_i}}{\sqrt{l_i}} \left(e^{\lambda l_i} - \frac{1}{2}\right)^{1/2}$$

and

$$s_i \propto p_i^{-1/2} \, l_i^{1/2} \left(e^{\lambda l_i} - \frac{1}{2}\right)^{-1/2}$$

The scheduling algorithms presented previously can be trivially modified to take into account the new square-root rule (Theorem 2), to design good schedules for environments subject to failure. The above result suggests that large items should be transmitted in smaller chunks (this is usually true when dealing with errors). However, doing so requires that the client to be able to piece together an information item using chunks that may arrive out of order.

5 Multiple Broadcast Channels

The discussion so far assumed that the server is broadcasting items over a single channel and all the clients are tuned to this channel. One can also conceive an environment in which the server has a large available bandwidth which is divided into multiple channels, the channels being numbered 1 through c. The clients can then subscribe to as many channels as they want (and can afford). It is apparent that proper use of these channels should result in better performance. Here we give two different schemes to exploit these channels to improve performance.

5.1 Channel Schedule Staggering

This scheme is basically an extension of cyclic scheduling algorithm. Here we assume that a schedule is created before hand using some *off-line* algorithm such as the *cyclic scheduling algorithm* described earlier. The same schedule is broadcast on every available channel – the schedule on channel i is delayed $(i-1)\,\tau$ time units as compared to channel 1. We refer to this as "staggering". Thus, if an instance of item i appears on channel j at time t, then on channel $j+1$, the corresponding instance of item i would appear at time $t+\tau$. A client, if tuned to more than one channel, should try to listen to the channels which are equidistant from each other. For example, if there are six channels used by server for data broadcast and a client can tune to three different channels, it should listen to channels 1, 3 and 5 or channels 2, 4 and 6. Of course, the total number of channels may not always be an integral multiple of the number of channels a client is listening to. In general, the *subscribed* channels may be selected to be as equidistant as possible.

It can be shown that for <u>two</u> channels ($c = 2$), *optimal* overall mean access time is achieved when the offset τ is the weighted average of the spacing, s_i, over all items, that is,

$$\tau = \frac{1}{2} \sum_{i=1}^{M} p_i \, s_i$$

s_i is the optimal spacing between consecutive instances of item i. The above result assumes that Lemma 1 is satisfied by all the items. While equal spacing is not always achievable, it can be approximated (as done by our scheduling algorithms). Similar results may be obtained for larger number of channels as well.

5.2 On-line scheme for multiple channels

This scheme is the modification of the *on-line* algorithm, described in Section 3.2, to accommodate multiple channels. In this case, a client capable of listening to n broadcast channels, listens to channels 1 through n (unlike previous section where the channels should be as equidistant as possible).

The on-line scheduling scheme works as follows. Generate items to broadcast over channel 1 using the normal *on-line* algorithm (as in Section 3.2). For channel 2, generate the items using the same *on-line* algorithm, but with different interpretation of $R(i)$ for item i. Let us define $R_j(i)$ as the instant when item i was last broadcast over channel j. Here $R(i)$ is defined as the maximum of $R_1(i)$ and $R_2(i)$, $1 \leq i \leq M$, and use it in the *on-line* algorithm to generate items for channel 2. Similarly, we define $R(i)$ to be maximum of $R_1(i)$, $R_2(i)$ and $R_3(i)$ for channel 3 and use the *on-line* algorithm to generate channel 3 items, and so on. More formally, the algorithm for the generation of items in channel k may be devised as follows :

ON-LINE algorithm for channel k:

$R(i) = max_{1 \leq j \leq c}\{R_j(i)\}, \quad \forall i, \;\; 1 \leq i \leq M$
`broadcast item` i `at time` Q
  `if`
    $(Q - R(i))^2 \; p_i/l_i \;\; \geq \;\; (Q - R(j))^2 \; p_j/l_j, \quad 1 \leq j \leq M$

(if there is a tie, that is, if the above condition is satisfied for more than one item, any one of those items may be chosen arbitrarily.)

$R_k(i) = Q$`, where` i `is the item broadcast` □

The above algorithm can be easily modified to use bucketing. A complete analysis of the algorithm and its simulation is one of the topics of our on-going research.

6 Performance Evaluation

We consider two classes of demand probability distributions:

- Skewed distribution: In this case, half the data items are requested more frequently than the other half. The distribution is characterized by a parameter B, where $0.5 \leq B \leq 1$. The distribution can be formally defined as below:

$$p_i = \begin{cases} \frac{B}{\lfloor M/2 \rfloor}, & 1 \leq i \leq \lfloor M/2 \rfloor \\ \\ \frac{1-B}{M - \lfloor M/2 \rfloor}, & \lfloor M/2 \rfloor < i \leq M \end{cases}$$

  Varying the value of B yields distributions with different amount of demand "skew".

- Exponential distribution: In this case, item 1 is requested most frequently, and item M is requested least frequently, the probabilities (p_i's) being determined by an *exponential* function of i. This distribution is characterized by a parameter γ. The distribution can be formally defined as below:

  $$p_i = p_1 \; e^{-\gamma \, (i-1)}$$

  where p_1 is evaluated by solving the equality $\sum_{i=1}^{M} p_i = 1$. Thus, $p_1 = (1 - e^{-\gamma})/(1 - e^{-M\gamma})$. Larger values of γ cause p_i to decrease more rapidly with increasing i.

Each of these two probability distributions are evaluated with the following three length distributions :

- Uniform length distribution: Length l_i of item i is defined as $l_i = L$, where L is a constant. (We select $L = 10$ for our simulations.)
- Increasing length distribution: Length l_i of item i, in this case, is defined as $l_i = i$, $1 \le i \le M$.
- Decreasing length distribution: In this case, length l_i of item i is defined as $l_i = M - i + 1$, where M is the total number of items in server's database, and $1 \le i \le M$.

We compare the *overall mean access time* t achieved by two of our algorithms, *on-line* and *on-line with bucketing*, with the "optimal" t given by Equation 3

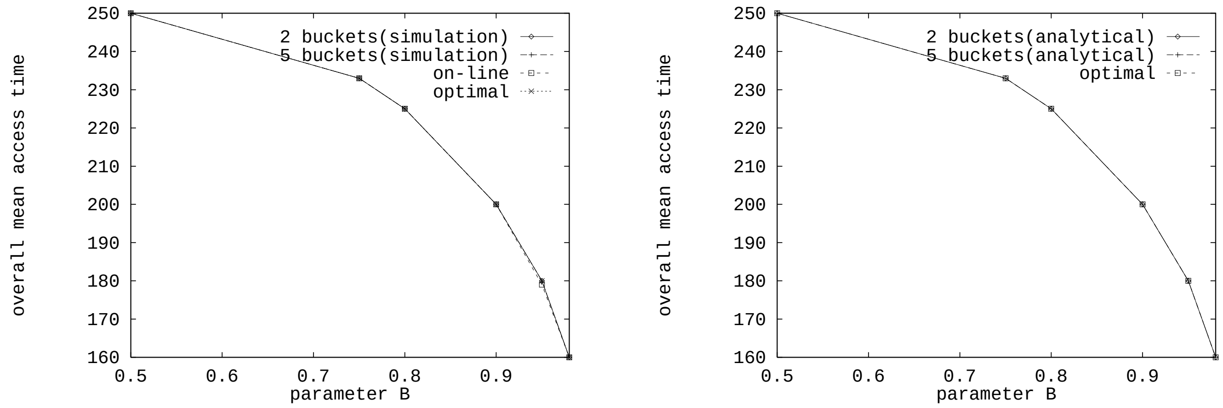
$$t_{\text{optimal}} = \frac{1}{2} \left(\sum_{i=1}^{M} \sqrt{p_i l_i} \right)^2$$

In the figures to be presented below, the curves corresponding to our *on-line* algorithm, and *on-line with bucketing* algorithm, are labeled as *on-line* and *n buckets*, respectively, where n shows the number of buckets used. Whereas, the curve corresponding to t_{optimal} is labeled *optimal.*

It should be noted that the theoretical optimum values of t may not be achievable for all probability distributions. However, the theoretical values do provide a measure to determine how close to optimal our algorithms are.

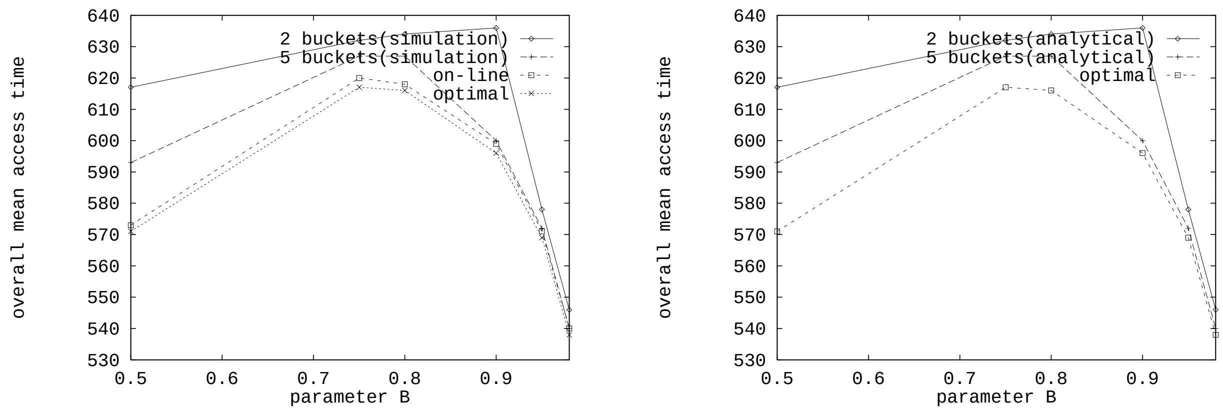
For the simulation results presented here, number of items (M) is 50. Similar results can be obtained for larger values as well. The reason for choosing a small M was to enable us to run experiments long enough to obtain simulation results with high confidence. Each schedule was simulated long enough until 300,000 requests were satisfied. Uncorrectable transmission errors (as in Section 4) are not considered in these simulations.

Figures 6, 7 and 8 plot *overall mean access time* against different values of B, the parameter in skewed distribution, for uniform length, decreasing length and increasing length distributions, respectively. In each of these figures, (a) plots simulation results and (b) plots analytical results. For *on-line with bucketing algorithm* curves, the analytical value was calculated using Equation 7 for different number of buckets. Note that *on-line* curve is a special case of *on-line with bucketing* curves with number of buckets= M. The assignment of various items to buckets is performed using the heuristic presented in Section 3.3.



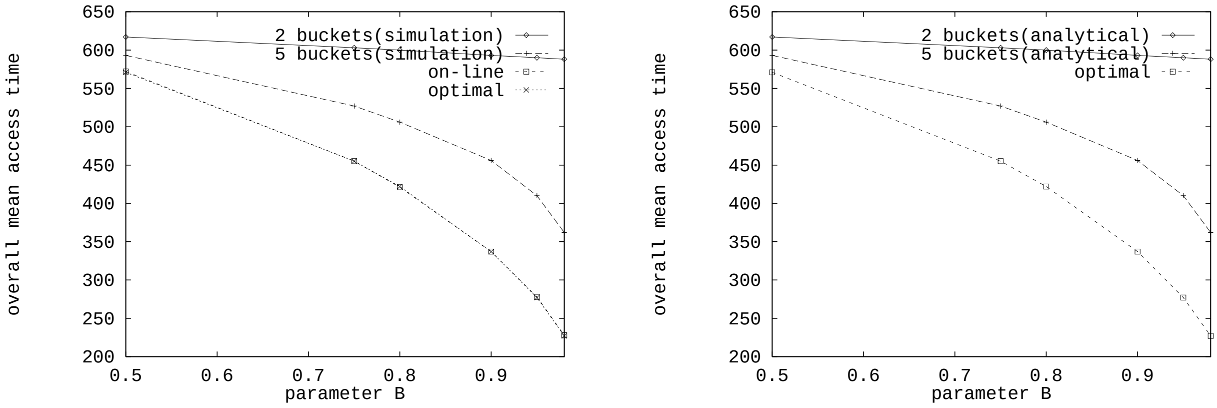
(a) Simulation results (b) Analytical results

Figure 6: *Overall mean access time* with skewed probability distribution (for different values of B) and using uniform length distribution $l_i = 10$. All curves coincide in each graph.



(a) Simulation results (b) Analytical results

Figure 7: *Overall mean access time* with skewed probability distribution (for different values of B) and using decreasing length distribution $l_i = M - i + 1$, where $M = 50$.



(a) Simulation results

(b) Analytical results

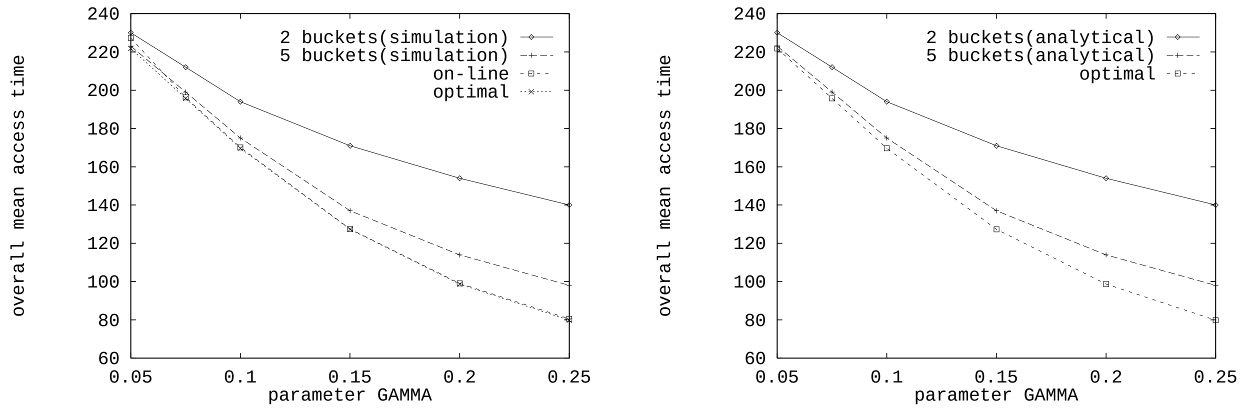
Figure 8: *Overall mean access time* with skewed probability distribution (for different values of B) and using increasing length distribution $l_i = i$. *On-line* and *optimal* curves in (a) coincide.

In all these figures, note that the simulation results almost match the analytical values for bucketing case. *On-line* algorithm, though not most time-efficient, performs pretty close to *optimal*. Note that the performance tends to improve as the number of buckets is increased. It may also be noted that most of the time, the *on-line with bucketing* algorithm with 5 buckets perform fairly close to (pure) *on-line* algorithm which is a special case with number of buckets=M (here 50). It is worth noting that this small loss in performance wins us a speed-up in computation by a factor of $50/5 = 10$.

Figure 6 is interesting in that all the curves coincide with each other. The reason is that the length of all the items is uniform (here $l_i = 10, \forall i$) and hence the *overall mean access time* is dependent only upon probability distribution, which has only two possible values of probabilities. Hence, there are only two possible values of $(p_i/l_i), \forall i$. This results in only two non-empty buckets, even if we use the assignment algorithm to create 5 buckets. This is why all the curves coincide with each other and with the *optimal* curve.

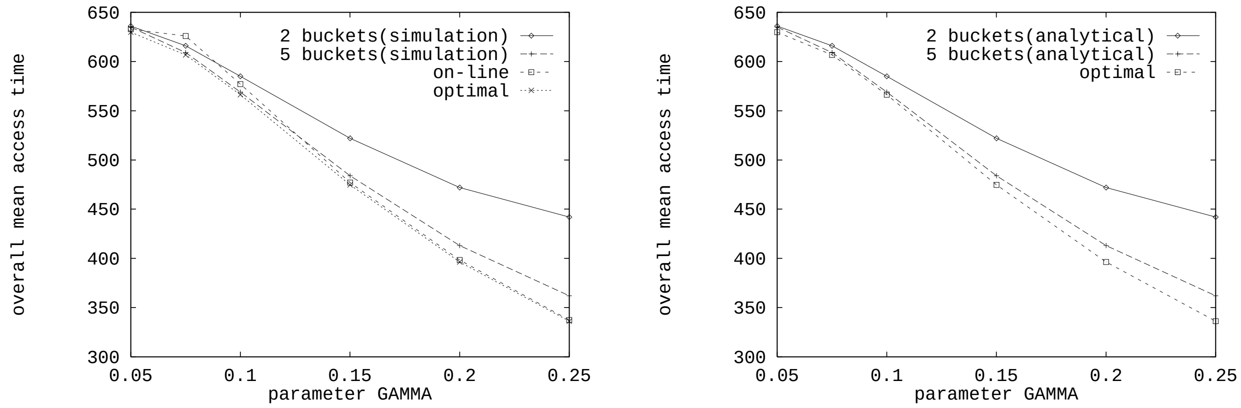
Figures 9, 10 and 11 plot *overall mean access time* against different values of γ, the parameter of exponential distribution, for uniform length, decreasing length and increasing length distribution, respectively. The observations made earlier for figures 6, 7 and 8 are valid here as well.

In order to emphasize the necessity of taking lengths of items into account, we also plot in Figure 12 a modified version of on-line algorithm in which we use the equation $s_i^2 p_i$=constant, instead of $s_i^2 p_i / l_i$=constant. The figure plots *overall mean access time* against different values of B, for decreasing length distribution, $l_i = M - i + 1$. In the figure, the normal *on-line* algorithm is labeled as *on-line*, whereas the modified version which does not take lengths into account is labeled *without lengths*. Note that some of the previous work [4, 13, 15, 14] does not take into account lengths of different items (or assumes lengths to be uniform). It is evident from Figure 12 that their results cannot be applied to the case where lengths of different items are different without encountering loss in performance.



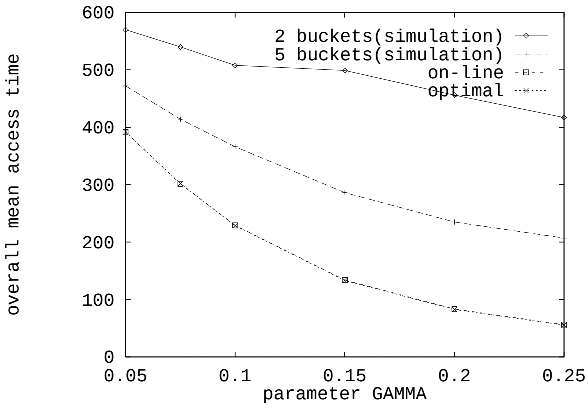
(a) Simulation results (b) Analytical results

Figure 9: *Overall mean access time* with exponential probability distribution (for different values of γ) and using uniform length distribution $l_i = 10$. *On-line* and *optimal* curves in (a) coincide.

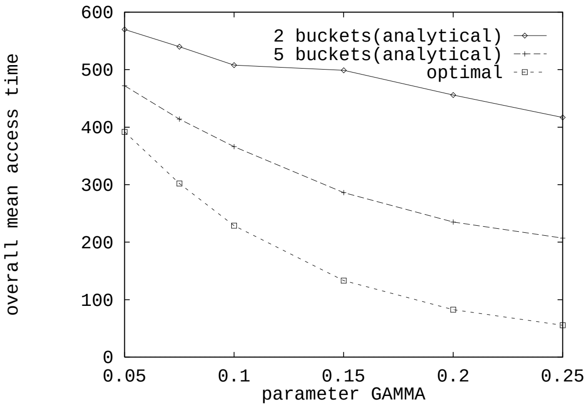


(a) Simulation results (b) Analytical results

Figure 10: *Overall mean access time* with exponential probability distribution (for different values of γ) and using decreasing length distribution $l_i = M - i + 1$, where $M = 50$. *On-line* and *optimal* curves in (a) coincide.



(a) Simulation results



(b) Analytical results

Figure 11: *Overall mean access time* with exponential probability distribution (for different values of γ) and using increasing length distribution $l_i = i$

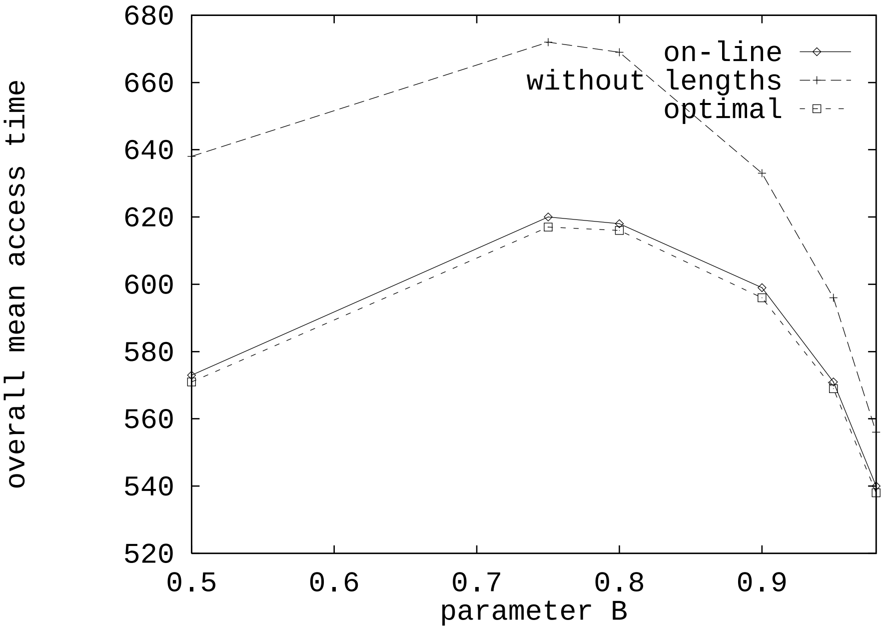


Figure 12: *Overall mean access time* plotted against different values of B (skewed distribution) for decreasing length distribution. The figure shows that not considering lengths results in poor performance if the lengths are not uniform.

We conclude from this section that *on-line* algorithm almost always performs very close to *optimal*. Simulation results demonstrate that the *on-line with bucketing* algorithm allows a trade-off between run-time overhead of the on-line algorithm, and its performance. We also saw that ignoring difference in lengths of various items can have a significant detrimental impact on performance.

7 Static and Adaptive Broadcasts

In the asymmetric communication environment under consideration, the clients may not be capable of sending requests for data to the server (or, it may not be desirable for the clients to send the requests to the server). In this case, the server must be provided with information (such as user profiles) which will allow the server to estimate the demand probability for each item. The broadcast schedule used by the server is determined completely by this estimate of the demand. The broadcast schedule will not change unless the user profiles are updated. Such broadcast schedules are said to be *static*.

An *adaptive* scheme will continually gather statistics on the demand for various items, and modify the broadcast schedule to best meet the demand. While we did not evaluate the adaptive behavior of our algorithms, we believe that our on-line schemes can quickly adapt to any changes in the demand probabilities.

To achieve adaptive behavior, the server must have some way to update its estimates of *demand probability* (p_i). Viswanathan [14] describes several ways in which this information can be conveyed to the server, including (i) when a mobile client moves from one cell to another, and (ii) when the client sends some other message, the access frequency information is piggybacked on these messages. The idea is to minimize the need to send messages just to update the user access profiles at the server. Viswanathan [14] divides items into two classes: (i) items that are broadcast by the server, and (ii) items that are sent to clients only on-demand (i.e., when a request is received from the client for the item). Thus, clients send requests for items from class (ii), and simply wait for the server to transmit an item from class (i).

We suggest another approach that does not divide items into two classes as above [13]. Instead, when a client needs some item, say i, it waits for some time bounded by a *time-out* τ. If an interval of τ is exceeded, then the client sends a request to the server. Thus, if an item is broadcast by the server within τ, no explicit request is sent. Parameter τ would affect the mean access time, as well as the number of requests that are sent by the clients. The server, based on the number of requests that are received, can update its estimate of demand probabilities (p_i). These new estimates can be used to determine the appropriate broadcast schedule. Performance evaluation of the above adaptive approach is a subject of future research.

8 Related Work

The problem of broadcasting data efficiently has received much attention lately. The existing schemes can be roughly divided into two categories (some schemes may belong to both categories, we have listed them in the most appropriate category): Schemes attempting to reduce the *access time* [4, 3, 2, 1, 8, 12, 7, 6, 15, 16] and schemes attempting to reduce the *tuning time* [10, 9, 11].

Ammar and Wong [4, 15] have performed extensive research on broadcast scheduling and obtained many interesting results. Our square root rule is a generalization of that obtained by Ammar and Wong. Wong [15] and Imielinski and Viswanathan [8, 14] present an on-line scheme that uses a *probabilistic* approach for deciding which item to transmit. Our on-line algorithm results in an improvement by a factor of 2 in the mean access time as compared to the probabilistic on-line algorithm in [8, 15]. Chiueh [6] and Acharya et al. [3, 2, 1] present schemes that transmit the more frequently used items more often. However, they do not use optimal degree of replication, unlike our schemes.

Jain and Werth [12] note that reducing the variance of spacing between consecutive instances of an item reduces the mean access time. The two schemes presented in this report do attempt to achieve a low variance. Similar to our discussion in Section 4, Jain and Werth [12] also note that errors may occur in transmission of data. Their solution to this problem is to use error control codes (ECC) for forward error correction, and a RAID-like approach (dubbed airRAID) that stripes the data. The server is required to transmit the stripes on different frequencies, much like the RAID approach spreads stripes of data on different disks [5]. This requires the clients to receive broadcasts on multiple frequencies. ECC is not always sufficient to achieve forward error correction, therefore, uncorrectable errors remains an issue (which is ignored in the past work on data broadcast).

9 Summary

This report considers *asymmetric* environments wherein a server has a much larger communication bandwidth available as compared to the clients. In such an environment, an effective way for the server to communicate information to the clients is to broadcast the information periodically. Contributions of this report are as follows:

- We propose three on-line and off-line algorithms for scheduling broadcasts, with the goal of minimizing the *access time*. Simulation results show that our algorithms perform quite well (very close to the theoretical optimal). The proposed on-line algorithms are suitable for *adaptive* broadcasts, as they can quickly adapt to any changes in demand probability distributions.
- The report considers the impact of errors on optimal broadcast schedules.
- When different clients are capable of listening on different number of broadcast channels, the schedules on different broadcast channels should be designed so as to minimize the access time for all clients. This report presents preliminary results on this problem.

More work is needed on some problems discussed in this report. Future work will also include design of strategies for caching and updates that attempt to achieve optimal performance while incurring low overhead. Further results will be made available at our web site at

`http://www.cs.tamu.edu/faculty/vaidya/mobile.html` .

A Appendix: Proof of Theorem 1

Theorem 1: Square-root Rule: *Given the demand probability p_i of each item i, the minimum* overall mean access time, *t, is achieved when frequency f_i of each item i is proportional to $\sqrt{p_i}$ and inversely proportional to $\sqrt{l_i}$, assuming that instances of each item are equally spaced. That is,*

$$f_i \propto \frac{\sqrt{p_i}}{\sqrt{l_i}}$$

To put it differently, for items i and j,

$$\frac{f_i}{f_j} = \sqrt{\frac{p_i}{p_j}}\sqrt{\frac{l_j}{l_i}}$$

Proof: As instances of item i are spaced equally, the spacing between consecutive instances of item i is N/f_i, where $N = \sum_{j=1}^{M} f_j l_j$ is the number of slots in the broadcast cycle. Also, in this case, the *item mean access time* is $t_i = s_i/2$. Therefore, $t_i = \frac{N}{2f_i}$. Now, *overall mean access time* $t = \sum_{i=1}^{M} p_i t_i$. Therefore, we have,

$$t = \sum_{i=1}^{M} p_i \frac{N}{2f_i} = \frac{1}{2}\sum_{i=1}^{M} p_i \frac{N}{f_i} \tag{9}$$

Define "supply" of item i, $q_i = \frac{f_i l_i}{N}$. Thus, q_i is the fraction of time during which item i is broadcast. Now note that, $\sum_{i=1}^{M} q_i = \sum_{i=1}^{M} \frac{f_i l_i}{N} = \frac{N}{N} = 1$. Now, Equation 9 can be rewritten as,

$$t = \frac{1}{2}\sum_{i=1}^{M} \frac{p_i l_i}{q_i} \tag{10}$$

As $\sum_{i=1}^{M} q_i = 1$, only $M-1$ of the q_i's can be changed independently. Now, for the optimal values of q_i, we must have $\frac{\partial t}{\partial q_i} = 0, \forall i$. We now solve these equations, beginning with $0 = \frac{\partial t}{\partial q_1}$.

$$\begin{aligned}
0 &= \frac{\partial t}{\partial q_1} \\
&= \frac{1}{2}\frac{\partial}{\partial q_1}\left(\sum_{i=1}^{M}\frac{p_i l_i}{q_i}\right) \\
&= \frac{1}{2}\frac{\partial}{\partial q_1}\left(\frac{p_1 l_1}{q_1} + \sum_{i=2}^{M-1}\frac{p_i l_i}{q_i} + \frac{p_M l_M}{(1-\sum_{i=1}^{M-1} q_i)}\right) \\
&= \frac{1}{2}\left(-\frac{p_1 l_1}{q_1^2} + \frac{p_M l_M}{(1-\sum_{i=1}^{M-1} q_i)^2}\right) \\
\Longrightarrow \frac{p_1 l_1}{q_1^2} &= \frac{p_M l_M}{(1-\sum_{i=1}^{M-1} q_i)^2} \qquad (11)
\end{aligned}$$

$$\text{Similarly} \quad \frac{p_2 l_2}{q_2^2} = \frac{p_M l_M}{(1 - \sum_{i=1}^{M-1} q_i)^2} \tag{12}$$

From Equations 11 and 12, we get

$$\frac{p_1 l_1}{q_1^2} = \frac{p_2 l_2}{q_2^2} \implies \frac{q_1}{q_2} = \sqrt{\frac{p_1 l_1}{p_2 l_2}}$$

$$\text{Similarly it can be shown that} \quad \frac{q_i}{q_j} = \sqrt{\frac{p_i l_i}{p_j l_j}}, \quad \forall i, j$$

This implies that, the optimal values of q_i's must be linearly proportional to $\sqrt{p_i l_i}$'s. It is easy to see that constant of proportionality $a = \frac{1}{\sum_{j=1}^{M} \sqrt{p_j l_j}}$ exists such that $q_i = a\sqrt{p_i l_i}$ is the *only* possible solution for the equations $\frac{\partial t}{\partial q_i} = 0$. From physical description of the problem, we know that a non-negative minimum of t must exist. Therefore, the above solution is unique and yields the minimum t.

Substituting $q_i = \frac{\sqrt{p_i l_i}}{\sum_{j=1}^{M} \sqrt{p_j l_j}}$ into Equation 10, and simplifying, yields optimal *overall mean access time* as

$$t = \frac{1}{2} \left(\sum_{i=1}^{M} \sqrt{p_i l_j} \right)^2$$

Also the optimal frequency of item i, f_i may be obtained as $q_i = \frac{f_i l_i}{N} \propto \sqrt{p_i l_i}$,

$$\implies f_i \propto \sqrt{\frac{p_i}{l_i}}$$

□

B Appendix: Cyclic Nature of the On-line Schedule

Assume that the ties in the on-line algorithm (Section 3.2) are resolved using a deterministic rule that is time-independent. With this assumption, for given (fixed) probability and length distributions, the on-line algorithm produces a cyclic schedule (as shown here). The length of the cycle, however, is hard to predict. (NOTE: The initial transient may not be identical to the cycle produced by the algorithm. We are focusing on the steady state behavior of the algorithm here.)

The proof that the on-line algorithm produces a cyclic schedule is based on the observation that the maximum spacing between consecutive instance of any item is finite and bounded. We present an informal proof. *To simplify the proof here we assume that $l_i = 1$ for all i. The proof can be generalized to arbitrary item sizes.*

Assume that the items are sorted such that $p_i \leq p_j$, if $i < j$. Thus, item 1 has the smallest demand probability, and item M has the largest demand probability. Observe that, due to the decision rule used in our on-line algorithm, any item may be broadcast at *most* once between consecutive instances of item M. (Thus, item M is broadcast at least once each M time units.) Similarly, between consecutive instances of item 1 each other item must be broadcast at *least* once.

The time at which latest instance of item 1 was transmitted is $R(1)$. Consider the time instances, say $\tau_1, \tau_2, \cdots, \tau_i, \cdots$ where item M is broadcast after the latest instance of item 1, before item 1 is broadcast again. (Recall that item M is transmitted at least once each M time units.) Now we show that the next broadcast of item 1 must occur within finite and bounded time. That is the sequence $\tau_1, \tau_2, \cdots, \tau_i, \cdots$ is finite. As item M is broadcast at τ_{i-1} and τ_i, without an intervening broadcast of item 1, it follows that

$$(\tau_i - \tau_{i-1})^2 p_M \geq (\tau_i - R(1))^2 p_1$$

(Recall that here we have assumed $l_i = 1, \forall i$.) As $\tau_i - \tau_{i-1} \leq M$, the above inequality implies that

$$M^2 p_M \geq (\tau_i - R(1))^2 p_1$$

However, the above inequality cannot hold for arbitrarily large t_i. The left-hand-side of the inequality is constant, while the right-hand-side increases monotonically with increasing i. Therefore, $\tau_i - R(1)$ cannot be larger than $M\sqrt{p_M/p_1}$. Thus, there can be at most $M\sqrt{p_M/p_1}$ instance of item M between consecutive instances of item 1. Similarly, the number of instances of each item i $(2 \leq i \leq M)$ between consecutive instances of item 1 are finite and bounded. Therefore, the spacing between consecutive instances of item 1 is bounded. Thus, the distinct *item sequences* that can appear between consecutive instances of item 1 are bounded in number. As each item appears at least once between consecutive instances of item 1, if any *item sequence* is repeated twice (and some sequence will eventually have to be repeated), the rest of the schedule will repeat itself (due to deterministic nature of the algorithm). Thus, after an initial transient, the on-line algorithm will produce a cyclic schedule. This argument can also be extended to variable size items.

C Appendix: Proof of Optimal Overall Mean Access Time for On-line with Bucketing Algorithm

Proof: Suppose that each bucket B_j, $1 \leq j \leq k$, contains m_j items, $m_j > 0$, $\sum_{j=1}^{k} m_j = M$. Let $q_j = \sum_{i \epsilon B_j} p_i/m_j$ and $d_j = \sum_{i \epsilon B_j} l_i/m_j$ be the average access probability and average length of the items in bucket B_j, respectively.

The proof is similar to the proof in Appendix A. With bucketing, the frequency of all items in the same bucket is identical. We define F_i as the frequency of items in bucket B_i. For optimal solution, the items should be equally spaced. Therefore, spacing between consecutive instances of an item in bucket i is N/F_i. Let S_i denote the spacing N/F_i.

Now, $N = \sum_{j=1}^{k} F_j d_j m_j$. Therefore, $S_i = N/F_i = \sum_{j=1}^{k} F_j d_j m_j / F_i$. Let T_i denote the *item mean access time* of an item in bucket B_i. Then, $T_i = \frac{1}{2} S_i = \frac{1}{2} N/F_i = \frac{1}{2}(\sum_{j=1}^{M} F_j d_j m_j)/F_i$. Note that, with the equal spacing assumption, *item mean access time* is identical for all items in the same bucket.

The *Overall Mean Access Time* is now given by

$$t = \sum_{j=1}^{k} \left(\sum_{i \in B_j} p_i \right) T_j$$

Since $\sum_{i \in B_j} p_i = m_j q_j$, the above equation can be written as

$$t = \sum_{j=1}^{k} m_j q_j T_j$$

or

$$t = \frac{N}{2} \sum_{j=1}^{k} \frac{q_j m_j}{F_j}$$

We define *supply* of bucket B_j, denoted r_j, as $r_j = F_j d_j m_j / N$. Observe that $\sum_{j=1}^{k} r_j = 1$. The above equation for t can be rewritten as

$$t = \frac{1}{2} \sum_{j=1}^{k} \frac{q_j m_j^2 d_j}{r_j} \tag{13}$$

If we denote $q_j m_j^2 d_j$ as X_j, the above equation becomes

$$t = \frac{1}{2} \sum_{j=1}^{k} \frac{X_j}{r_j}$$

where $\sum_{j=1}^{k} r_j = 1$. This equation has the same form as Equation 10. Therefore, from the proof in Appendix A it follows that, with bucketing, to minimize t the following condition must be true:

$$r_j \propto \sqrt{X_j} \tag{14}$$

As $\sum_{j=1}^{k} r_j = 1$, $r_j = \frac{\sqrt{X_j}}{\sum_{i=1}^{M} \sqrt{X_j}}$. Substituting this into Equation 13, replacing $X_j = q_j m_j^2 d_j$, and simplifying, we get

$$t_{\text{opt\_bucket}} = \frac{1}{2} \left(\sum_{j=1}^{k} m_j \sqrt{q_j d_j} \right)^2$$

Substituting $X_j = q_j m_j^2 d_j$ in the above proportionality (14), we get

$$r_j \propto \sqrt{q_j m_j^2 d_j} = m_j \sqrt{q_j} \sqrt{d_j}$$

As $r_j = F_j d_j m_j / N$, we now get $\frac{F_j d_j m_j}{N} \propto m_j \sqrt{q_j} \sqrt{d_j}$. On simplifying, this yields,

$$F_j \propto \frac{\sqrt{q_j}}{\sqrt{d_j}}$$

As $F_j = N/S_j$, we have,

$$S_j \propto \frac{\sqrt{d_j}}{\sqrt{q_j}}$$

Finally, note that, if item i is in bucket B_j, then $f_i = F_j$ and $s_i = S_j$.

□

D Appendix: Overall Mean Access Time in Presence of Errors

Here we are not assuming bucketing. The result below can be easily generalized to the case where items are divided into buckets. Consider item i, instances of which are spaced s_i time units apart. The total time required to transmit the cycle is N. Then, $f_i = N/s_i$. Also, as size of item i is l_i, we have $\sum_{i=1}^{M} f_i l_i = N$.

First, let us determine the *item mean access time*, t_i, for item i. Observe that *average* time until the first instance of item i is transmitted, from the time when a client starts waiting for item i, is $s_i/2$ time units. If the first instance of item i transmitted after a client starts waiting is corrupted, then an additional s_i time units of wait is needed until the next instance. Thus, each instance of item i that is received with uncorrectable errors adds s_i to the waiting time. Given that the probability that an instance of item i of length l_i contains uncorrectable errors is $E(l_i)$, the expected number of consecutive instances with uncorrectable errors is obtained as

$$\frac{E(l_i)}{1 - E(l_i)}$$

Thus, the *item mean access time* is obtained to be

$$t_i \;=\; \frac{s_i}{2} + s_i \left(\frac{E(l_i)}{1 - E(l_i)} \right) \;=\; s_i \left(\frac{E(l_i)}{1 - E(l_i)} + \frac{1}{2} \right) \;=\; \frac{1}{2} s_i \left(\frac{1 + E(l_i)}{1 - E(l_i)} \right)$$

Thus,

$$t = \sum_{i=1}^{M} p_i t_i = \sum_{i=1}^{M} p_i \, s_i \left(\frac{E(l_i)}{1 - E(l_i)} + \frac{1}{2} \right)$$

Proof of Theorem 2 : As $s_i = N/f_i$, the above expression for t can be rewritten as,

$$t = \sum_{i=1}^{M} p_i \, \frac{N}{f_i} \left(\frac{E(l_i)}{1 - E(l_i)} + \frac{1}{2} \right) \;=\; \sum_{i=1}^{M} \frac{p_i l_i \left(\frac{E(l_i)}{1 - E(l_i)} + \frac{1}{2} \right)}{r_i}$$

where, $r_i = f_i \, l_i/N$. Now, $\sum_{i=1}^{M} r_i = \sum_{i=1}^{M} f_i l_i/N = N/N = 1$. Thus, the above expression for t has a form similar to Equation 10, and can be minimized similarly. The optimization procedure yields the result stated in Theorem 2.

References

[1] S. Acharya, M. Franklin, and S. Zdonik, "Prefetching from a broadcast disk," in *12th International Conference on Data Engineering*, Feb. 1996.

[2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks - data management for asymmetric communications environment," in *ACM SIGMOD Conference*, May 1995.

[3] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination-based data delivery using broadcast disks," *IEEE Personal Communication*, pp. 50–60, Dec. 1995.

[4] M. H. Ammar and J. W. Wong, "On the Optimality of Cyclic Transmission in Teletex Systems", in *IEEE Transactions on Communications*, Vol. COM-35 No. 1, pp. 68-73, Jan. 1987.

[5] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–185, 1994.

[6] T. Chiueh, "Scheduling for broadcast-based file systems," in *MOBIDATA Workshop*, Nov. 1994.

[7] V. A. Gondhalekar, "Scheduling periodic wireless data broadcast," Dec. 1995. M.S. Thesis, The University of Texas at Austin.

[8] T. Imielinski and S. Viswanathan, "Adaptive wireless information systems," in *Proceedings of SIGDBS (Special Interest Group in DataBase Systems) Conference*, Oct. 1994.

[9] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Power efficient filtering of data on air," in *4th International Conference on Extending Database Technology*, Mar. 1984.

[10] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy efficient indexing on air," May 1994.

[11] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on the air - organization and access," submitted for publication.

[12] R. Jain and J. Werth, "Airdisks and airRAID : Modelling and scheduling periodic wireless data broadcast (extended abstract)," Tech. Rep. DIMACS Tech. Report 95-11, Rutgers University, May 1995.

[13] N. H. Vaidya and S. Hameed, "Data Broadcast Scheduling (Part I)," Tech. Report 96-012, Computer Science Dept., Texas A&M University, May 1996.

[14] S. Viswanathan, *Publishing in Wireless and Wireline Environments.* PhD thesis, Rutgers, Nov. 1994.

[15] J. W. Wong, "Broadcast Delivery", in *Proceedings of IEEE*, pp. 1566-1577, Dec. 1988,

[16] Z. Zdonik, R. Alonso, M. Franklin, and S. Acharya, "Are disks in the air, just pie in the sky?," in *IEEE Workshop on Mobile Comp. System*, Dec. 1994.