A New Hybrid Broadcast Scheduling Algorithm for Asymmetric Communication Systems*

Yufei Guo
Dept. of Computer Sci. & Eng.
University of Texas at Arlington
Arlington, TX 76019-0015
yguo1@email.mot.com

M. Cristina Pinotti
Dept. of Mathematics
University of Trento
Povo, 38050 (TN) ITALY
pinotti@science.unitn.it

Sajal K. Das
Dept. of Computer Sci.& Eng.
University of Texas at Arlington
Arlington, TX 76019-0015
das@cse.uta.edu

It is believed that broadcast is an efficient way to transmit data in an asymmetric communication system. Most of the previous work focused on either pull-based or push-based scheduling. However, for systems with a very large number of data items, none of these schemes is efficient individually. We propose a novel hybrid scheduling algorithm which uses both pull- and push-based schemes. In our approach, data items are divided, by a suitable *cut-off point*, into two disjoint sets: one consisting of *more-popular* items and the other of *less-popular* items. The items in the former set are broadcast by a push-based schedule, while those in the latter set by a pull-based schedule. A cut-off point is said to be optimal when it minimizes the expected access time in the system. The optimal cut-off point is derived analytically and validated by simulation experiments. Results show that the proposed hybrid push- and pull-based scheduling performs always better than a pure push-based scheduling and improves on the expected access time of an existing hybrid scheduling scheme.

Keywords Broadcast scheduling, asymmetric communication systems, push-based scheduling, pull-based scheduling, cut-off point, hybrid algorithm

I. INTRODUCTION

Many computer and communication systems use a *client-server* architecture to transmit data items between the server and clients. In such a system, data transfer from the server to clients is called *downstream* communication, whereas the transfer in the opposite direction is called *upstream* communication. If the downstream communication capacity is much greater than the upstream communication capacity, the system is called an *asymmetric communication system*. For example, wireless networks, information dispersal systems, and video-on-demand services are all such systems. For which *broadcast* technology is an attractive and efficient way to satisfy clients' requirements in such systems.

There are two basic ways to broadcast data items. In a *pull-based* (or, *on-demand*) system, clients initiate the data transfers by sending requests to the server. The server then makes a schedule to satisfy the client's requests. In a *push-based* system, on the other hand, clients get the required data items by listening to the broadcast channel and capturing the data whenever it goes by. The server broadcasts data items on scheduled time no matter whether the particular item is required or not at that time.

A push-based system is modeled in Figure 1, where the server is located in the center. Many clients around the server listen to the channels to obtain data items a, b, c, and so on.

Figure 2 models a pull-based system. The server broadcasts only those items requested by some clients. Clients will be lined up in the order that requests are made.

No matter which scheme is used, a transmission schedule called *broadcast scheduling*, must be designed to determine when data items are to be transmitted by the server.

It will be shown that neither push-based nor pull-based scheduling alone can obtain the optimal performance. In fact, the best performance is achieved when the two scheduling are used together in an effective manner. In our approach, data items are divided into two disjoint sets - *more-popular* set and *less-popular* set. A push-based scheduling is used to broadcast the items in the more-popular set and a pull-based scheduling is used for the items in the less-popular set. This new hybrid scheduling algorithm leads to a lower expected access time than other existing scheduling schemes. The improvement in the expected access time depends on the selection of the optimal *cut-off* point between the two sets, which is also derived in this paper.

The rest of the paper is organized as follows. Section 2 reviews previous results in this area. Section 3 introduces definitions and scheduling performance metrics. Section 4 offers motivations behind using a hybrid scheduling. Section 5 discusses how to set up the cut-off point and Section 6 evaluates the new scheduling. Conclusions and future work are offered in Section 7.

*This work is partially supported by Texas Advanced Research Program and NSF grants EIA-0086280 and EIA-0115885, and by the Project "Resource Allocation in Wireless Network", MURST, ITALY.

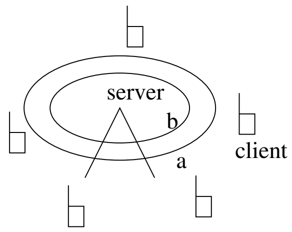

Figure 1: Push-based System.

II. BACKGROUND

Scheduling theory has been studied since 1960s. Most of the early research was focused on point-to-point systems, where an individual client makes a request and the server responds to the request individually. Then from 1980s, as wireless technology and information dispersal systems developed rapidly, the broadcast system technology received more and more attention.

Wong [17] studied the broadcast delivery problem in a teletext system. Three architectures were considered: the one-way broadcast (push-based), the two-way interaction (pull-based), and the hybrid one-way broadcast/two-way interaction. The first two architectures were better studied than the last one.

For push-based systems, as proved in [12], the broadcast schedule with minimum expected access time results when the instances of each item are equally spaced.

Acharya, Alonso, Franklin, and Zdonik [1] introduced the concept of *Broadcast Disks* and proposed an algorithm for a push-based scheme which improves performance for non-uniformly accessed data. In effect, the broadcast channel becomes a "disk". Data items are assigned to different disks of different sizes and speeds. Then these disks are multiplexed on a single broadcast channel.

Vaidya and Hameed [10] derived a scheme that obtains one of the best expected access time for push-based system. They also extended [16] the packet fair queuing scheduling to the broadcast schedule on multiple channels, and investigated the impact of transmission errors on scheduling.

Su, Tassiulas, and Tsotras [14] formulated the push-based broadcast schedule as a deterministic dynamic optimization problem. Their solution provides the optimal push-based schedule. Identifying the properties of the optimal solution, they derived a heuristic which achieves average response time very close to the lower bound.

For pull-based scheduling, Aksoy and Franklin [5] discussed an efficient algorithm which constantly measures, for each item, how frequently it is required and for how long it is missing on the air. The current item to be transmitted is selected as the one which is more popular and with the longest missing time. Rather than choosing an item based on the estimation of the access probability, they make a decision based on the current state of the system. This approach has the advantage of being easily adaptive to changes in the intensity and distribution of the workload.

Many researchers (e.g. [3, 6, 15, 17]) realized that caching and prefetching are very important criteria on the performance of broadcast scheduling, because they can save access time as well as power [11]. For example, in [15], the server follows a schedule that attempts to match the user access statistics. In case of non-homogeneous user populations with several different access profiles, the users have local cache to smooth out the mismatches of their profile with the broadcast schedule statistics. It turns out that a combined design of broadcast and cache management leads to a significant performance improvement.

In contrast, much less attention has been paid to developing hybrid approaches for broadcast scheduling. To the best of our knowledge, the only mixed push and pull approach is due to Acharya, Franklin, and Zdonik [2].

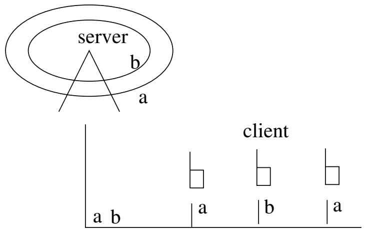

Figure 2: Pull-based System.

While the server transmits all the items based on a push-based schedule, clients can make requests for items which are missing on the air for a time interval longer than a given threshold. Pulled items are interleaved with pushed items. A parameter specifies how many pull pages may be sent after each single pushed page. The model in [2] assumes that the server has a bounded capacity for accepting requests from the clients. Hence, in a heavily loaded system, the requests from the clients are more likely to be dropped and the broadcast schedule will be the unique source for the distribution of such items. The results in [2] show that the integration of the push and pull approaches provides a significant performance improvement.

In this paper, we propose a new hybrid scheme which dynamically mixes push- and pull-based schedules, but differently from [2], an item is sent on demand if and only if it will never appear in the push-based scheduling. In fact, we partition the set of data items into two disjoint sets, depending on their *access probability*, *– the more-popular* and the *less-popular* items. The former can only be broadcast on a push-based scheduling, while the latter can only be sent on demand (pull-based). The *cut-off point* between the more-popular and the less-popular sets is selected in such a way that no more than one item can be required, on average, by clients during a single item transmission. Therefore, we are guaranteed that no request will be lost. Our approach saves bandwidth by avoiding duplicate transmissions for less-popular items. Finally, since the optimal cut-off point between the two sets can be computed analytically and off-line from time to time, the proposed algorithm may be considered *adaptive* to different workloads, which is highly desirable as shown by experiments in [5].

III. DEFINITIONS AND METRICS

We assume a server has a total of D data items, all of which are stored in memory, and can be retrieved immediately by the server.

From now on, we adopt the following terminology. A server *sends* an item if pulled on the air on-demand, whereas a server *broadcasts* an item if pushed on the air without any explicit client request. Moreover, we say that a client *accesses* (resp., *requests*) a data item broadcast (resp., sent) by the server.

Different items may be accessed or requested by a different number of clients. We call such a number the *degree of popularity* of the item. It is measured by the *access probability* in a given period of time T. Let the access probability for item i be denoted as p_i. Let M be the total number of accesses and/or requests to the system, and I be the number of accesses and/or requests for item i. A measure for the access probability is then:

$$p_i = \frac{I}{M}. \tag{1}$$

Assume that the item i has the i-th largest access probability. That is, listing the items as *1, 2, . . . , D*, their access probabilities $p_1, p_2, p_3, \ldots, p_D$ appear in non-increasing order as $p_1 \geq p_2 \geq p_3 \ldots \geq p_D$.

The item can be broken into several pieces. Each piece of unit length is called a *page*. To simplify our discussion, we assume that all items have unit length.

Moreover, the *server speed,* τ measures how fast a server transmits a single item of unit length. A *time slot* or *unit time* is the duration of the transmission of a page. Let N denote the average number of accesses and requests in the system (due to all clients) during a single time slot.

Finally, we recall that in a pure push-based system, the same schedule is cyclically repeated, and that a single repetition is called a *broadcast cycle*. During a broadcast cycle, some items may appear several times. Each appearance is referred to as an *instance* of the item. Denoting s as the space between two consecutive instances of an item, if all instances of item i are equally spaced, then the space between any two instances of item i will be denoted as s_i.

III.A. Scheduling Performance Metrics

There are several criteria that determine whether a scheduling algorithm is properly selected. Broadly speaking, a scheduling algorithm must balance both the individual client expectation and the overall system performance. The individual measures of scheduling performance include:

Access Time: the amount of time $T_{acc,i}$ that a client listens to the broadcast schedule for a data item i. The aim of the push-based scheduling is to keep the access time for each item i as small as possible. Note that, if all the instances of a pushed item i are equally distributed with space s_i in the broadcast cycle, the average access time $\overline{T_{acc,i}}$ yields $\overline{T_{acc,i}}{=}s_i/2$.

Response Time: the amount of time $T_{resp,i}$ that a client waits before its request for item i is served. The aim of the scheduling for the pull-based systems is to minimize the response time for each item i. Let $\overline{T_{resp,i}}$ denote the average response time for pulled item i.

In addition to each individual item measure, an overall measure of the scheduling performance is given by either the maximum or the average of the access times of all items. Many scheduling algorithms use average measure and, rather than a simple arithmetic average of the access times (or their average), they consider a weighted average which weights the access time of each item with its popularity.

For example, the most commonly used overall measure is called the *Expected Access Time*, T_{exp_acc}, defined as

$$T_{exp\_acc} = \sum_{i=1}^{D} p_i \bullet \overline{T_i} \tag{2}$$

where $\overline{T_i}$ is the average access time $\overline{T_{acc,i}}$ (resp., the average response time $\overline{T_{resp,i}}$) for item i if it is pushed (resp., pulled).

Many scheduling algorithms have been designed to minimize T_{exp_acc}. Among them, for a pushed-based system, the Packet Fair Scheduling [8, 9, 10] is a popular one, which will be used to implement the push-based schedule of the more-popular items in the experimental part of our work.

Finally, we adopt the expected access time as the performance measure of our new algorithm.

IV. NEW HYBRID SCHEDULING

This section describes our hybrid broadcast scheduling algorithm in details. But before that, let us discuss motivations behind hybrid scheduling.

IV.A. Motivations Behind Hybrid Scheduling

As pointed out earlier, most of the existing broadcast scheduling algorithms are either push-based or pull-based. However they suffer from drawbacks as outlined below.

In purely push-based systems, the server alone decides which data items to transmit without interacting with the clients. Therefore, to make sure that all client accesses can be satisfied, the server must send all data items in the system. When the number of data in the server is very large, each broadcast cycle will take a long time and thus the expected access time of the system will be large. Moreover, since the server has a huge number of data items, many items will have a very small access probability and will appear just once in the broadcast cycle. Therefore, their average access time will be half of the length of the broadcast cycle, which is very long too.

Pull-based scheduling can resolve this problem by sending just what clients request. However, a pure pull-based scheduling is inefficient when the system is highly loaded and the data items are very popular. In this case, due to the large number of requests – most of them being repeated – it is almost impossible to organize an efficient schedule. Moreover, the reverse channels from the clients to the server will soon become overloaded, consequently some requests will be dropped.

Our intuition is that an algorithm which sends on demand the less-popular items and broadcasts the more-popular ones, can overcome the drawbacks of the above two basic approaches. As increased number of less-popular items are sent on demand, the broadcast cycle becomes shorter and not much delay is introduced to serve requests on demand, so the overall expected access time will become smaller and smaller. However, when too many items are sent on demand, the system tends to be pull-based. The expected access time will start to increase again because as more-popular items are sent on demand, the load of the pull system becomes higher and higher. This suggests that there must be a way to choose the set of less-popular items to minimize the expected access time, which will be less than that of either a pure push-based or pull-based scheme.

Based on this observation, we propose a new hybrid scheme, which combines both pull-based and push-based schedules. Precisely, letting $Z = \{i|1 \leq i \leq D\}$ be the set of all the D items, we partition Z into two disjoint sets, $A = \{i|i \leq K, i \in Z\}$, and $R = \{i|i > K, i \in Z\}$, where K is termed the *cut-off point*. We use a push-based scheduling to broadcast the more-popular items in A(also called *access set*). The less-popular items in R (also called *request set*) are sent immediately on demand and interleaved with the push-based scheduled items. The cut-off point K, which separates the items in A from those in R, is chosen in such a way that the expected access time is minimized.

Before discussing the proposed algorithm, let us first discuss the method to select the optimal cut-off point K, termed K_{opt}.

IV.B. Criteria to Set the Cut-Off Point

We know that any push-based broadcast schedule is based on access probability p_i and assume that the sum of all access probabilities is equal to 1. Because of the cut-off point K, we have $\sum_{i=1}^{K} p_i \neq 1$. For scheduling purposes the access probability of item $i \in A$ is then modified as:

$$p_i^* = \frac{p_i}{\sum_{j=1}^{K} p_j} \tag{3}$$

The sum of probabilities for items in the request set R is $\left(1 - \sum_{i=1}^{K} p_i\right)$. If we define x as the average number of new requests for $i \in R$ in a unit time, then

$$x = N\left(1 - \sum_{i=1}^{K} p_i\right) \tag{4}$$

where N is the average number of accesses/requests for all the items in the system in a unit time. If K is too small, the items in R are requested with very high probability and it is possible that

$$x = N\left(1 - \sum_{i=1}^{K} p_i\right) > 1$$

That means, while the server is transmitting an item, there are more than one new requests generated. Since the server needs to send the next item $i \in R$ immediately, there will be no time to go back to the push-based schedule. The next item of the access set A to be broadcast is delayed: its access time becomes higher, and hence the total expected access time increases. In other words, when K becomes smaller and smaller, there will be more and more new requests for items in R generated in a unit time. Such pending requests could be stored in a queue and served in the First Come First Served (FCFS) order. We want to choose K such that no more than one item goes in such a queue during a time slot. Observe that when

$$x = N\left(1 - \sum_{i=1}^{K} p_i\right) = 1 \tag{5}$$

the queue starts to build. Hence, K will also be called *the build-up point* when

$$\sum_{i=1}^{K} p_i = 1 - \frac{1}{N}. \tag{6}$$

Since we do not want the FCFS queue to build up, the following relation must hold:

$$\sum_{i=1}^{K} p_i > 1 - \frac{1}{N}. \tag{7}$$

As explained before, when K moves away from the build-up point and becomes larger, at first the expected access time decreases but then again it starts to increase. Hence, there must be a value of K for which the expected access time is the minimum. We say that such a value is the *optimal cut-off point*.

To be able to compute the optimal cut-off point, we need to know, on an average, after how many units of time dedicated to send items requested on demand, there will be an empty slot to broadcast again a scheduled item.

For this purpose, let y define the space between two consecutive broadcast slots. If $x = 1$, there is always a requested item to send. Scheduled items never have a chance to be broadcast and $y = \infty$. If $x = 0$, which is the case when all items are broadcast, then $y = 0$. If $0 < x < 1$, we can think that each unit time is logically made up of two parts - time to broadcast and time to send.

Since we assume that all data items have unit length, the fraction of sending part will be x. Actually, x is the fraction of total time that is used to send requested items, and $1 - x$ is the fraction of total time that is used to broadcast scheduled items. Since among (1+y) units of time, y are used to send requested items (see Figure 3), it holds that $x = y/(1+y)$.

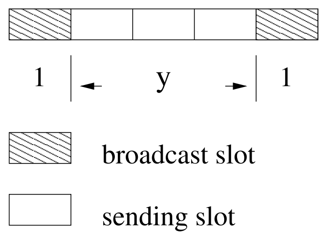

Figure 3: Fraction of Sending Time.

So, the space between two consecutive broadcast items will be

$$y = \frac{1}{1-x} - 1. \tag{8}$$

Let us define

$$U(K) = \sum_{i=1}^{K} p_i \tag{9}$$

$$V(K) = \sum_{i=1}^{K} \sqrt{p_i} \tag{10}$$

Recall that we assume the item i has the i-th largest access probability. So U is the summary of the first K largest access probabilities. By replacing x in equation (8) with equation (4), the space between two consecutive broadcast slots is obtained as

$$y = \frac{N(1-U(K))}{1-N(1-U(K))} \tag{11}$$

which means that between two broadcast slots, there will be y slots for sending items $i \in R$ as shown in Figure 4.

The optimal instance space for item $i \in A$ is given by (12)

$$s_i = \frac{\sum_{j=1}^{K} \sqrt{p_j^*}}{\sqrt{p_i}^*} \tag{12}$$

From equations (3), (9), and (10), we have

$$\sum_{j=1}^{K} \sqrt{p_j^*} = \sum_{j=1}^{K} \sqrt{\frac{p_j}{U(K)}} = \frac{V(K)}{\sqrt{U(K)}} \tag{13}$$

So

$$s_i = \frac{V(K)}{\sqrt{U(K)}\sqrt{p_i^*}} = \frac{V(K)\sqrt{U(K)}}{\sqrt{U(K)}\sqrt{p_i}} = \frac{V(K)}{\sqrt{p_i}} \tag{14}$$

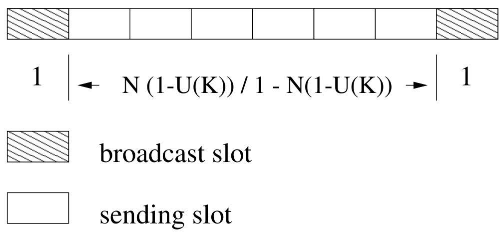

Figure 4: Sending Slots and Broadcast Slots.

However, broadcast is not continuous due to the interruptions by new requested items. Between s_i broadcast slots, there are

$$s_i \left(\frac{N(1-U(K))}{1-N(1-U(K))} \right) \tag{15}$$

sending slots. So the actual instance space for item $i \in A$ is given by

$$S_i^* = s_i + s_i \left[\frac{N(1-U(K))}{1-N(1-U(K))} \right] =$$

$$\frac{s_i}{1-N(1-U(K))} = \frac{V(K)}{1-N(1-U(K))} \bullet \frac{1}{\sqrt{p_i}} \tag{16}$$

The average access time for item $i \in A$ can now be computed as

$$\overline{T_{acc,i} \in A} = \frac{S_i^*}{2} \tag{17}$$

Because the new requested items are sent immediately, the average access time for item $i \in R$ satisfies:

$$\overline{T_{res,i} \in R} = \frac{1}{2} \tag{18}$$

The total expected access time is then

$$T_{exp\_acc}(K, N) = \sum_{i=1}^{K} \frac{S_i^*}{2} p_i + \sum_{i=K+1}^{n} \frac{1}{2} p_i = \frac{1}{2} \left[\frac{V(K)}{1-N(1-U(K))} \sum_{i=1}^{K} \frac{1}{\sqrt{p_i}} p_i + \sum_{i=K+1}^{n} p_i \right] =$$

$$\frac{1}{2} \left[\frac{V(K)}{1-N(1-U(K))} \sum_{i=1}^{K} \sqrt{p_i} + \left(1 - \sum_{i=1}^{K} p_i \right) \right] = \frac{1}{2} \left[\frac{V(K)^2}{1-N(1-U(K))} + 1 - U(K) \right].$$

So, in conclusion, the accurate formula for the total expected access time is then

$$T_{exp\_acc}(K, N) = \frac{1}{2} \left[\frac{V(K)^2}{1-N(1-U(K))} + 1 - U(K) \right] \tag{19}$$

Let us repeat that this formula is only good for computing expected access time after the build-up point. Because of the building up of the FCFS queue, there will be no time to broadcast item $i \in A$ before this point. So the access time for these items is ∞ and $T_{exp\_acc}(K, N) = \infty$.

Anyway, we can then compute $T_{exp\_acc}(K, N)$ by varying the cut-off point from the build-up point, obtained by equation (6), up to the last item D.

From Equation (19), we also know that the expected access time depends on the average number, N, of accesses and requests in a time slot. N depends on two factors: the server speed, τ and the access density d, which can be defined as the number of total accesses and requests in one second. Indeed, it holds $N = d\tau$.

When N decreases, the optimal expected access time decreases. So if the server speed increases (i.e., τ drops) or, if the access density decreases (i.e., d drops), the optimal expected access time also decreases.

IV.C. Proposed Algorithm

To implement the new hybrid scheme, we use two channels: the *index channel* to transmit the index numbers of the broadcast items, and the *data channel* to transmit the actual data.

On the server side, we suppose that the server knows access probability and access density. We also assume that the items are sorted in non-increasing order of access probability. That is, item 1 will be the one with the largest access probability, and in general, item i will have the i-th largest access probability. The server needs to compute the build-up point from Equation (6), and the optimal cut-off point K from Equation (19). Then it will broadcast the index on the index channel. At the same time, the scheduling algorithm is executed to find the broadcast schedule. The server broadcasts items on the data channel from 1 to K, that is, the server will broadcast the K items with the highest access probability. Any push-based scheduling algorithm, such as Packet Fair Scheduling [10], can be used for this purpose. Whenever the server receives a request from a client, it will

Cut-Off Point selection $(D, P = \{p_1, p_2, \ldots, p_D\}, N, K_{\text{opt}})$
/* D is the number of data items in the system,
P is the set of access probabilities of all data items,
N is the average number of accesses and requests during unit time,
K_{opt} is the optimal Cut-Off point */

Step 0. /* Compute the Build-Up Point $K^B(N)$. * /

Step 0.1 $B = 1$.

Step 0.2 $SUM = \sum_{i=1}^{B} p_i$

Step 0.3 If $SUM < 1 - \frac{1}{N}$, $B = B + 1$, go to Step 0.2,
else $K^B(N) = B$.

Step 1. Recalling that $K^B(N)$ is the minimum number of items to be pushed to avoid FCFS queue building up with N accesses and requests, and recalling that

$$U(K) = \sum_{i=1}^{K} p_i, \quad V(K) = \sum_{i=1}^{K} \sqrt{p_i}$$

From $K = K^B(N) + 1$ up to last item D:

- Compute $U(K)$ and $V(K)$;
- $T_{exp\_acc}(K, N) = \frac{1}{2}\left[\frac{V(K)^2}{1-N(1-U(K))} + 1 - U(K)\right]$;
- If $T_{exp\_acc}(K, N) > T_{exp\_acc}(K-1, N)$, then set the optimal `Cut-Off point` $K_{\text{opt}} = K - 1$, and exit.

Figure 5: Compute Cut-Off Point.

complete the current broadcasting and then send the requested item. Upon completion of the sending, the server will resume broadcasting.

On the client side, a client first listens to the index channel to check whether the needed data item is currently broadcast or not. If the requested item is in the index, then the client just waits until the data is broadcast. If the item is not in the index, the client asks the server for it.

The scheduling algorithm is described in Figure 6. Note that the procedure cut-off Point Selection in Figure 5 can be executed at the beginning and repeated from time to time to update the optimal cut-off point K_{opt} while workload changes.

The next section presents the results from the performance study of our scheduling algorithm for different cut-off points. For this purpose, a simulation program is developed to simulate the actual accesses and requests.

V. EXPERIMENTAL STUDY

In the simulation, a total of 100,000 instances of accesses and requests (out of a given D items) are generated. The instances of each item are assumed to be equally distributed. And instances of different items follow *Zipf's* distribution, which has been shown to closely approximate the access or request frequencies in asymmetric systems [9]. Many other researchers have also used *Zipf's* distribution in their broadcasting research in asymmetric systems [1, 10, 17, 18]. Note that the conclusions drawn in this paper also hold if another distribution is used.

Recalling that p_i is the access probability of item i, if p_i follows *Zipf's* distribution, then

Hybrid Scheduling Algorithm (P, K_{opt})
/* P is the set of access probabilities of all data items,
K_{opt} is the optimal Cut-Off point */

Step 1 Keep broadcasting, on the index channel, the indeces of the items which are in positions from 1 to K_{opt} in the sorted vector P of the access probabilities.

Step 2 Compute the next broadcast item by Push-Based Scheduling and broadcast it on the data channel if there is no client request. Else, send the requested item on data channel.

Step 3 Return to Step 2.

Figure 6: Hybrid Scheduling Algorithm

$$p_i = \frac{(1/i)^\theta}{\sum_{j=1}^{n} (1/j)^\theta}$$

where θ is called the *access skew coefficient* determining the distribution.

The simulation program reads N items at a time from the above list, which simulates that there are a total of N accesses and requests in unit time. Depending on the items read, the program either puts the item in some queue to simulate waiting, or sends the item immediately to simulate sending requests. At the same time the program will simulate broadcasting whenever there is no request. The access time is accumulated for each waiting accessed or requested item. At the end of the simulation, the accumulated access time of each item is computed to obtain the expected access time. The expected access time from Equation (19) is also computed. These two values are then compared.

V.A. Performance of the Proposed Algorithm

The number of total items in the server is set to D = 1,000. The cases for N = 2, 3, 4, and 5 are tested. Recall that N is the average number of accesses and requests in the system during a single time slot and that a time slot is the duration of the transmission of a page. For a fast system, a page transmission may take only few milliseconds. For example, about 1000 pages can be sent each minute. Thus, although the absolute value of N is small (N= 2,3,4,5), the system is overall serving a great number of access/requests. When N=5, the system can satisfy about 5000 accesses/requests each minute in this example. Also are tested the cases for N = 2, 5, 10, and 15. The cases for θ= 0.6, θ= 0.8, and θ= 1.0 are evaluated. Figures 7 - 10 display the performance results. In each figure, the solid line indicates the value obtained from the simulator results. The legend in the figures for this kind of lines is "N = (some number) simu.". On the other hand, the dashed lines represent the values computed from Equation (19). They are indicated by legends like "N = (some number) comp". In most cases, the computed results match so well with the respective simulation results that it is difficult to identify the solid lines from the dashed ones. Figure 7 shows the expected access time T_{exp_acc} over the whole range for θ= 0.6. The build-up points are clearly displayed in these figures for each case. For example, for N = 2, the build-up point corresponds to $K \approx 20$, which is indicated by the vertical dash line in Figure 7, and the performance drops dramatically. The other cases have similar results. For brevity, we do not show the plots for the whole range of other values of θ. Figures 8 - 10 show the expected access time (in the range of up to about 500) around the optimal cut-off point.

As seen from these figures, for a fixed value of N, when θ increases (i.e., the access probabilities become skewed), the set of the items to be pushed in the systems tends to be smaller and smaller. This is intuitively clear because, if θ becomes larger, a smaller number of items are accessed/requested by a large number of users. All the remaining items in the system are seldom accessed and requested and it is better to transmit them only by individual user request. In other words, while θ approaches 1, the number of items that need to be pushed in the system decreases, and the pull scheduling dominates the hybrid algorithm.

On the other hand, for a fixed a value of θ when N increases, the cut-off point increases too. Indeed, the hybrid scheduling uses a larger cut-off point to counterbalance the increasing number of accesses in the system. Pushing a larger number of items, most of the access-requests are satisfied without the individual transmission of the request item. Intuitively, increasing the cut-off point helps and enforces the strategy of having no queue for the user requests.

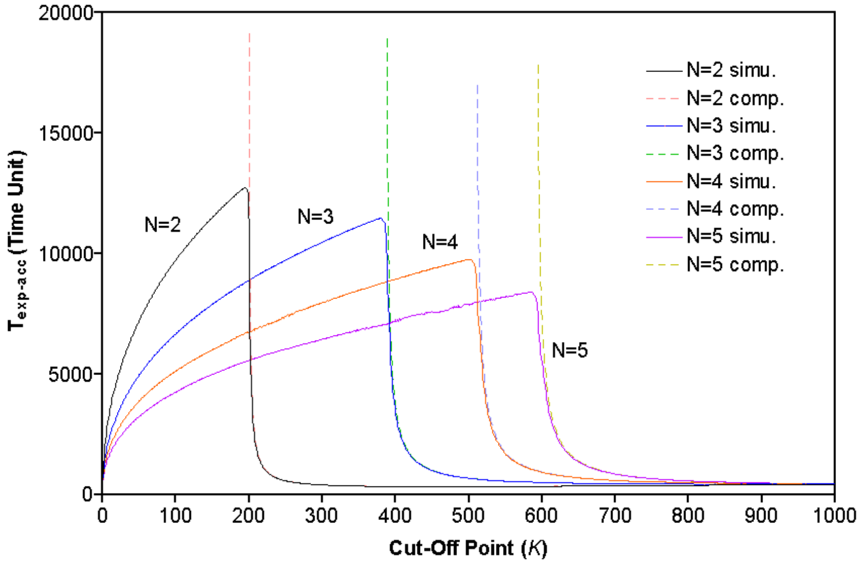

Figure 7: D = 1000, θ = 0.6 (whole range).

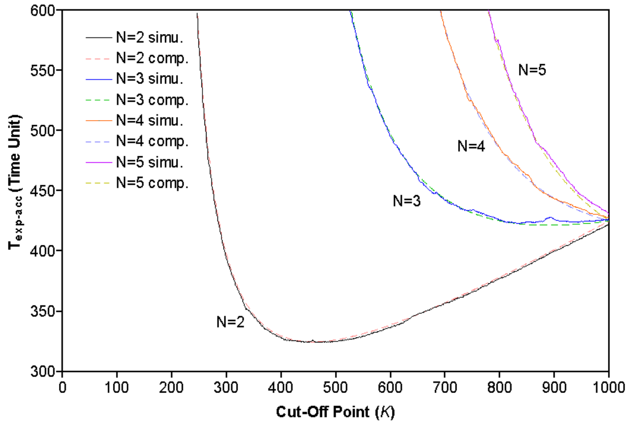

Figure 8: D = 1000, θ = 0.6 (in detail).

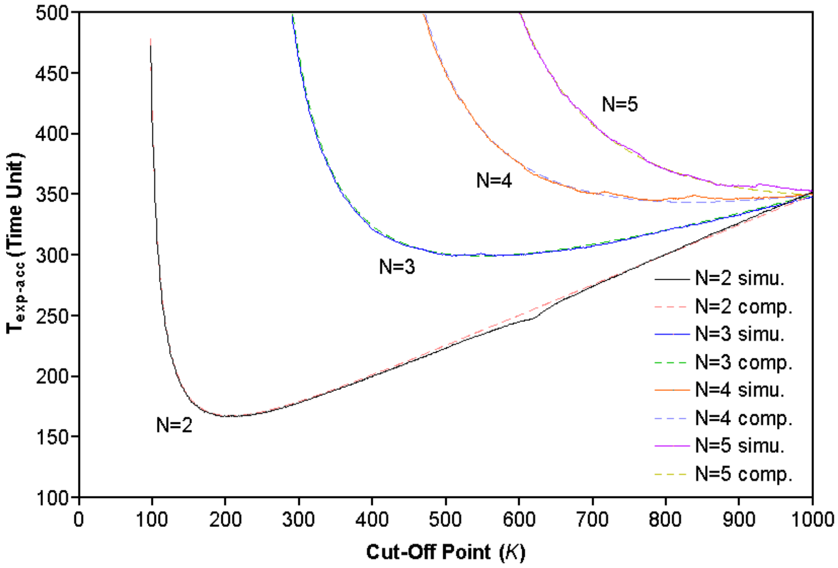

Figure 9: D = 1000, θ= 0.8 .

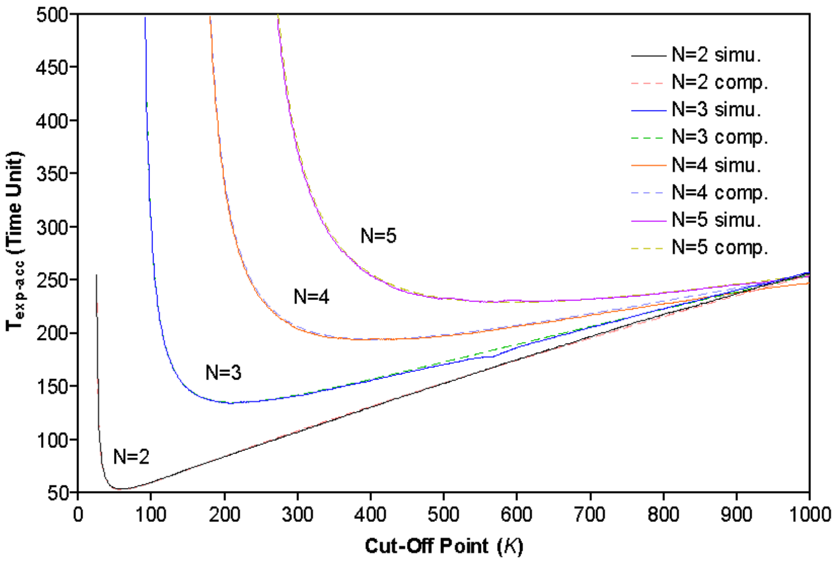

Figure 10: D = 1000, θ = 1.

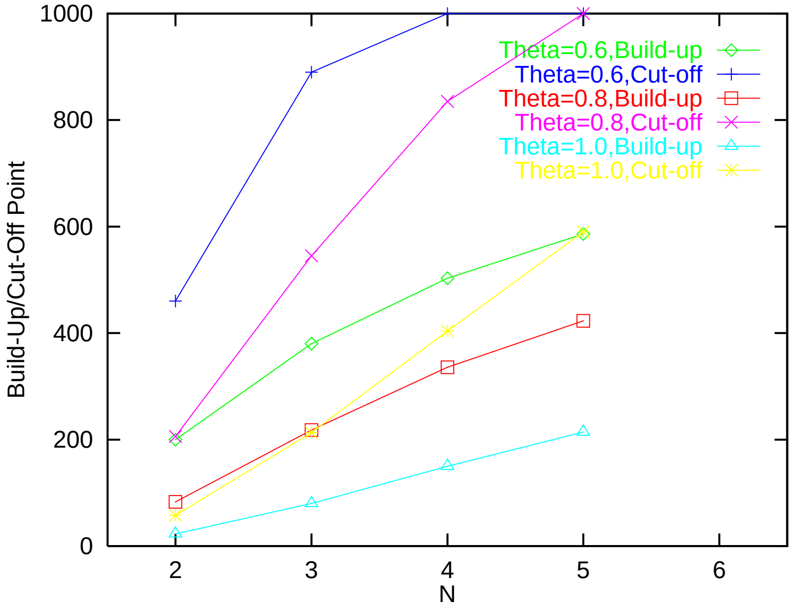

Figure 11: Distance between Build-Up and Cut-Off points.

The best performance of the hybrid algorithm is obtained if data items have good reasons to be divided into two groups. Thus, when the items in the system are not skewly accessed (small values of θ), the division of the items in more-popular and less-popular items is artificial, and the impact of the hybrid algorithm on the system performance is negligible. In fact, the optimal cut-off point becomes larger and the performance of the scheduling algorithm becomes close to the pure push-based algorithm. Eventually no meaningful optimal cut-off point can be found.

Moreover, the simulation experiments prove our idea that, when the number of data items in a broadcast system is very large, neither push-based nor pull-based algorithm is efficient. In addition, the curves of the expected access time from simulation experiments match very well with those analytically computed. This gives evidence that the analytical model for the cut-off point is meaningful and correct.

At this point, it is worthy to observe that the cut-off point and the build-up point are substantially different. The distance between build-up point and cut-off point varies with different N and θ as shown by Figure 11.

First observe that, as expected, both values – the build-up point and the cut-off point – grow with N. Moreover, it is worthy to point out that, for fixed value of θ, the distance between the build-up point and the cut-off point increases with N. That is, when the number of accesses/requests per unit time grows, the proposed approach is more effective (i.e the expected access time decreases) if the system pushes more items.

For fixed value of N, the distance between the cut-off point and the build-up point decreases when θ increases. When θ is smaller, the cut-off point balances the expected access time between the push- and the pull-schedule. When θ increases and it is close to 1, the cut-off point helps in avoiding the queuing of the requests. For example, when N=2 and $\theta = 0.6$, the curves show that the distance between the cut-off point and the build-up point is about 250 point away for a total of 1000 items. When N=2 and $\theta = 1$, the distance between the cut-off point and the build-up point is very small, about 30 point away. In synthesis, once again, the cut-off point plays a two folded role: (i) it makes stronger the condition given by the build-up point, and (ii) it also balances the push- and pull-based scheduling.

In the next two subsections, we compare our algorithm with a pure push-based scheduling namely, packet fair queueing and with another hybrid scheduling from the literature.

V.B. Comparison with Packet Fair Queuing

Packet Fair Queuing (PFQ) is used as the base algorithm to compute the performance of our proposed scheduling scheme. However, by using the pure PFQ, we do not consider N, the number of accesses and requests during a single time slot. For a specific N, the result can not be the minimum. Figure 12 shows the performance of the two schemes.

In this comparison, the performance when access skew coefficient is at 0.6, 0.8 and 1.0 are computed. The total number of total items is set to be D = 1,000. The curve that indicated by PFQ is the performance of the

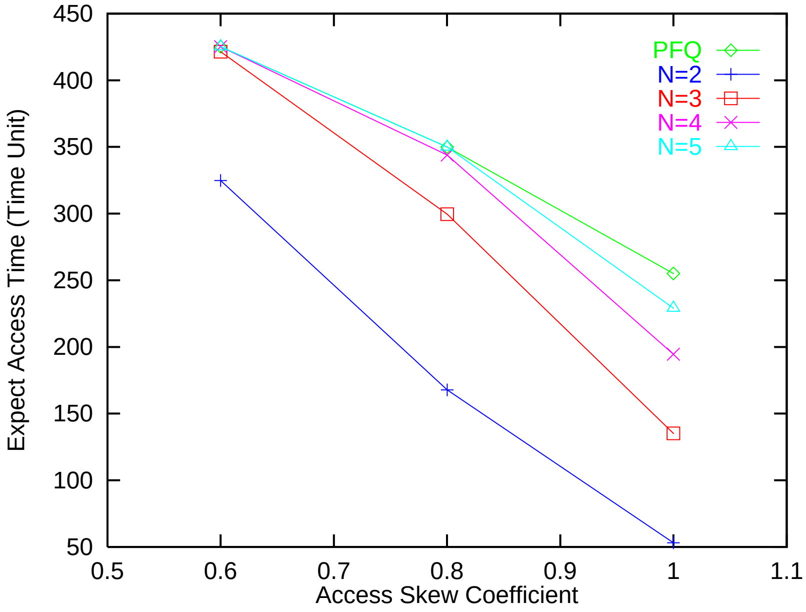

Figure 12: Comparison with the pure Packet Fair Queuing.

pure Packet Fair Queuing. Whereas, the curves indicated by N= 2, N= 3, N= 4, and N= 5 are the optimal performance of our proposed algorithm.

As explained before, when θ decreases the performance of out hybrid algorithm becomes closer to that of the pure-push scheduling. In fact, when $\theta = 1$, our approach is always better than the pure-push schedule, and therefore is able to take advantage from the pull-based scheduling.

For fixed θ, we see that if there are not so many accesses/requests, i.e., N is small, our proposal can obtain much better performance than Packet Fair Queuing. In fact, once again, when there are few accesses/requests, our approach takes advantage of the pure pull-based algorithm and avoids very long cyclic broadcast. On the other hand, when N increases, the performance of our hybrid scheduling will never be worse than the one of a pure push-based system.

V.C. Comparison with existing Hybrid Algorithm

Ref. [2] proposes another hybrid scheme combining pull and push scheduling algorithms. Instead of setting up a cut-off point, it uses *ThresPerc* as threshold for clients to decide if a pulling should be made. In addition, another parameter, called *PullBW*, is used to dedicate how much bandwidth of data channel is for pulling. This algorithm was proposed to reduce the access time for individual client in the worst case. However, the overall system performance will be degraded. This can be found by comparing T_{exp_acc} of our proposed algorithm with that of [2].

We use Packet Fair Queuing as the basic scheduling for both of them. Caching and noise are ignored, since they will have the same effect for both approaches. The access probability is set to be $\theta = 1.0$ and total number of data items is D = 1,000.

We first compute the overall system performance of the scheme in [2] by changing threshold. *PullBW* is set to 100% , which is the same as our proposal does. The results are shown in Figure 13. As we know, by using the algorithm proposed in this paper, at the optimal cut-off point, the T_{exp_acc} will be 53.16, 135.25, 194.55, and 229.17, when N = 2, 3, 4, and 5, correspondingly. However, the best overall performance for proposal in [2] is about 238 when threshold is about 1850. Actually, when threshold is so large, the system becomes pure push-based system. The overall performance for the scheme in [2] becomes better when *PullBW* is lower and lower. But they are never better than those in the pure push-based system and are far worse than those obtained by using our algorithm. This can be seen from Figure 14.

VI. CONCLUSIONS

In this paper the broadcast scheduling problem has been studied for asymmetric environments, where the downlink traffic capacity from the server to clients is much larger than the reverse channel capacity from clients to

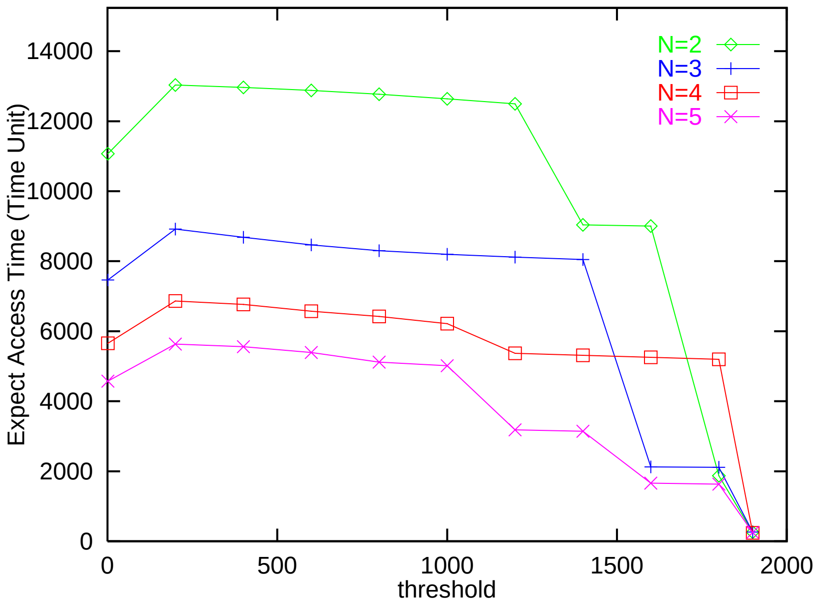

Figure 13: T_{exp_acc} as Threshold changes.

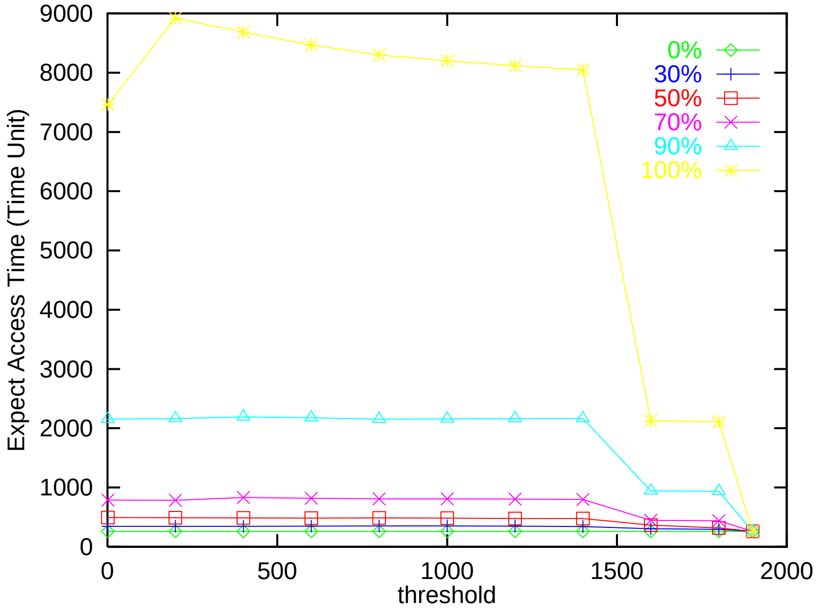

Figure 14: T_{exp_acc} as *PullBW* changes.

server. We propose a novel scheduling algorithm which is a hybrid of push-bash and pull-based techniques. This paper makes the following contributions:

- When the number of items in the server is very large, it is better not to broadcast all of them. By cutting the items at some point and broadcasting only the items before the cut-off point and sending on demand the requested items after this point, we may be able to reduce the expected access time.
- The optimal cut-off point can be derived analytically.
- The optimal expected access time is related to the push-based algorithm, cut-off point and the broadcast speed of the server.
- We conducted simulation experiments, the results of which support the proposed algorithms very well.

As further investigations, we intend to (a) study how the cut-off point varies when the system can keep some of the requests in a FCFS queue of bounded length; and (b) apply the cut-off point technique to distribute the data among several channels of communications.

References

[1] S. Acharya, R. Alonso, M. Franklin and S. Zdonik, “Broadcast disks – Data management for asymmetric communications environment”, Proceedings of ACM SIGMOD Int. Conference on Management of Data, pp. 199-210, May 1995.

[2] S. Acharya, M. Franklin and S. Zdonik, “Balancing Push and Pull for Data Broadcast”, Proceedings of ACM SIGMOD Int. Conference on Management of Data, pp. 183-193, 1997.

[3] S. Acharya, M. Franklin and S. Zdonik, “Prefetching from a broadcast disk”, Proceedings of 12th International Conference on Data Engineering, pp. 276-285, February 1996.

[4] S. Acharya and S. Muthukrishnan, “Scheduling on-demand broadcasts: New metrics and algorithms”, MOBICOM 98 Dallas Texas USA, pp. 43-54, 1998.

[5] D. Aksoy and M. Franklin, “R X W: A Scheduling Approach for large scale on-demand data broadcast”, IEEE/ACM Transac. On Networking, Vol. 7, No. 6, pp. 846-860, 1999.

[6] D. Barbara and T. Imielinski, “Sleepers and workaholics: Caching strategies in mobile environments”, Proceedings of ACM SIGMOD Conference, pp. 1-12, May 1994.

[7] A. Bar-Noy and Y. Shilo, “Optimal Broadcasting of Two Files Over an Asymmetric Channel”, Journal of Parallel and Distributed Computing, IDEAL, Vol. 60, No. 4, pp. 474-493, Apr. 2000.

[8] J.C.R. Bennett and H. Zhang, “Hierarchical packet fair queueing algorithms”, Proceedings of ACM SIGCOMM, pp. 43-56, 1996.

[9] J. Gecsei, “The architecture of videotex systems”, Englewood Cliffs, NJ: Prentice-Hall, 1983.

[10] S. Hameed and N.H. Vaidya, “Efficient algorithms for scheduling data broadcast”, Wireless Networks 5, pp. 183-193, 1999

[11] T. Imielinski, S. Viswanathan and B.R. Badrinath, “Energy efficient indexing on air”, Proceedings of International Conference on Management of Data, pp. 25-36, May 1994

[12] R. Jain and J. Werth, “Airdisks and airraid: Modelling and scheduling periodic wireless data broadcast (extended abstract)”, DIMACS Technical Report 95-11, Rutgers University, May 1995

[13] K. Stathatos, N. Roussopoulos, and J.S. Baras, “Adaptive data broadcast in hybrid networks”, Proceedings of 23rd International Conference on Very Large Data Bases, pp. 326-335, 1997.

[14] C. Su, L. Tassiulas, V.J. Tsotras, “Broadcast scheduling for information distribution”, Wireless Networks, Vol. 5, pp. 137-147, 1999.

[15] C. Su and L. Tassiulas, “Joint broadcast scheduling and user’s cache management for efficient information delivery”, Wireless Networks, Vol. 6, pp. 279-288, 2000.

[16] N.H. Vaidya and S. Hameed, “Scheduling data broadcast in asymmetric communication environments”, Wireless Networks 5, pp. 171-182, 1999.

[17] J.W. Wong, “Broadcast delivery”, Proceedings of IEEE, pp. 1566-1577, December 1988.

[18] J.X. Yu, T. Sakata, and K. Tan, “Statistical estimation of access frequencies in data broadcasting environments”, Wireless Networks 6, pp. 89-98, 20