Adaptive Broadcast Protocols to Support Efficient and Energy Conserving Retrieval from Databases in Mobile Computing Environments

Anindya Datta, Debra E. VanderMeer, Jeong Kim, Aslihan Celik, Vijay Kumar

May 8, 1997

TR-13

A TimeCenter Technical Report

Title	Adaptive Broadcast Protocols to Support Efficient and Energy Conserving Retrieval from Databases in Mobile Computing Environments
Author(s)	Anindya Datta, Debra E. VanderMeer, Jeong Kim, Aslihan Celik, Vijay Kumar
Publication History	April 1997, ICDE (A short version of this paper) A TimeCenter Technical Report

TIMECENTER Participants

Aalborg University, Denmark
Christian S. Jensen (codirector)
Michael H. Böhlen
Renato Busatto
Heidi Gregersen
Kristian Torp

University of Arizona, USA
Richard T. Snodgrass (codirector)
Anindya Datta
Sudha Ram

Individual participants
Curtis E. Dyreson, James Cook University, Australia
Kwang W. Nam, Chungbuk National University, Korea
Keun H. Ryu, Chungbuk National University, Korea
Michael D. Soo, University of South Florida, USA
Andreas Steiner, ETH Zurich, Switzerland
Vassilis Tsotras, Polytechnic University, New York, USA
Jef Wijsen, Vrije Universiteit Brussel, Belgium

The TimeCenter icon on the cover combines two "arrows." These "arrows" are letters in the so-called *Rune* alphabet used one millennium ago by the Vikings, as well as by their precedessors and successors, The Rune alphabet (second phase) has 16 letters. They all have angular shapes and lack horizontal lines because the primary storage medium was wood. However, runes may also

be found on jewelry, tools, and weapons. Runes were perceived by many as having magic, hidden powers.

The two Rune arrows in the icon denote “T” and “C,” respectively.

Abstract

Mobile computing has the potential for managing information globally. Data management issues in mobile computing have received some attention in recent times, and the design of *adaptive broadcast protocols* has been posed as an important problem. Such protocols are employed by database servers to decide on the content of broadcasts dynamically, in response to client mobility and demand patterns. In this paper we design such protocols and also propose efficient retrieval strategies that may be employed by clients to download information from broadcasts. The goal is to design *cooperative* strategies between server and client to provide access to information in such a way as to minimize energy expenditure by clients. We evaluate the performance of our protocols analytically.

1 Introduction

One of the most important information dissemination strategies in the current fast paced corporate and personal environments is to employ *wireless networks*. Furthermore, the data-hungry users who access such networks are expected to be mobile. In fact, a widely accepted information access model envisions millions of highly mobile users carrying small, battery powered palmtop terminals equipped with wireless connectivity. The types of information that may potentially be accessed are virtually boundless, e.g., financial, weather, traffic, sports and airline schedules.

While the challenges of designing and implementing wireless networks are important ones for the telecommunications research community, data management issues related to organizing and retrieving information from wireless channels have posed challenges for the database community as well. Recent advances in wireless technology have made data access possible for *mobile users*, who are geographically dispersed. However, there are a number of hardware and software problems which must be resolved before the capabilities of mobile computing can be fully utilized. Some of the software problems, such as data management, transaction management, database recovery, etc., have their origins in distributed database systems. In mobile computing, however, these problems become more difficult to solve, mainly because of the narrow bandwidth of the wireless communication channels, the relatively short active life of the power supplies (battery) of mobile units, and the changing locations of required information due to client mobility.

A widely accepted paradigm of disseminating information in a disconnected and mobile environment, is by the use of broadcasts [19, 4, 17, 16]. In this model a "server" broadcasts information and "clients" tune in to the broadcast to retrieve their data of interest. The widespread acceptance of the broadcast phenomenon is mainly due to the notion of *broadcast asymmetry*, where the "downstream" (server to client) communication capacity is relatively much greater than the "upstream" (client to server) communication capacity. The two major issues of interest in this scenario are (a) how and what does the server broadcast, and (b) how does the client retrieve? Of particular interest are solutions that enable clients to retrieve their data of interest *efficiently*, as well as with a *minimum* of energy expenditure. Of course, the two above questions is composed of several different sub-issues, as explained soon in this paper.

In this paper we deal with the problem of data retrieval by mobile units from wireless broadcasts. The goal is to design and evaluate adaptive broadcast protocols to support efficient information retrieval while simultaneously attempting to minimize energy consumption by the mobile units. The paper is organized as follows. In section 1.1, we differentiate the contribution of this paper from other prior work on data retrieval in mobile contexts, followed by a brief review of existing work

in the mobile computing area in section 2 and a description of a mobile computing architecture in section 3. We provide a concrete problem statement in section 4, discuss some preliminary information in section 5 and describe our protocols in section 6. Subsequently, we analyze the performance of our protocols in section 7 and evaluate and discuss the performance in section 8. We conclude in section 9.

1.1 Contribution of this Paper

In recent times there has been some interesting work in investigating data issues in mobile environments (see section 2 for extended discussion). In particular, very interesting research in the area of broadcasts in mobile environments is reported by Imilienski and Badrinath et al. in Rutgers (see e.g., [15, 4, 10, 17, 16]) and Vaidya et al. in Texas A & M (see e.g., [19, 20]). Most of this work is concerned with *access methods*, i.e., how to organize the broadcast data. In particular Imilienski, Badrinath et al. have done extensive treatment of indexing in broadcasts, and how to organize data and index in broadcasts. Vaidya et al. report nice results on broadcast scheduling, i.e., when to broadcast particular items, such that response times (termed "access times" in these papers) are minimized. In reporting this work, researchers have suggested various methods of indexing information, or have designed algorithms to maintain cache consistency at clients. This work presupposes that the composition of the broadcast is known. In other words, given the composition of a broadcast, there has been lot of nice work in organizing data and index on air, as well as scheduling broadcasts. Our concern is different. In [15], while discussing relevant future work, the authors mention that the most likely mode of retrieval in mobile environments is going to be *mixed*, i.e., the most frequently accessed items are broadcast and other items are provided on demand. To support such a mixed technique, the authors remark, *adaptive broadcast protocols* need to be designed. Such protocols will dynamically alter the broadcast content depending upon client demand patterns. In this paper we propose and analytically evaluate the performance of two such protocols. To the best of our knowledge, there does not exist, in the published literature, other examinations of the same topic.

Aside from the dynamic determination of broadcast content, which is concerned with the server side, this paper also delves deeply into the client side of things. Specifically, we address the problem of efficient and power conservent retrieval by clients. To this end, this paper proposes a number of retrieval protocols that a client might use to download data from broadcasts. Thus, this paper offers an *integrated* treatment of broadcast content determination and retrieval. This is achieved by designing *cooperative* protocols between the server and client sides. This is also a novel aspect of this work. We would like to note however, that we draw heavily upon existing work. Actually, this work, together with work cited above, forms a comprehensive theory of organizing, managing and retrieving from broadcasts in a wireless, mobile environment.

2 Related Work

The primary reference area of this paper is data management in wireless networks in general and in mobile computing in particular. There exists much work in the area of mobile computing, particularly from a telecommunications perspective. This work is somewhat orthogonal for our purposes. Therefore, rather than cite these papers, we prefer to refer the reader to the following excellent web site: http://maui.ee.tu-berlin.de/bibl/. This site, at last count, contained over 200 papers on

mobile networking and is an excellent source of background material. Our primary interest however, is the area of data management in the context of mobile computing. This area has received considerable research attention in the recent past. Nice discussions of several data management issues in mobile computing scenarios may be found in [2, 7, 11, 8]. While the aforementioned papers are somewhat high level, there are also papers that deal in detail with specific issues.

Architecture of database systems in mobile environments receives detailed treatment in [18, 1]. In [18], the authors propose a dynamic programming strategy to optimize the mapping of a database hierarchy to the network configuration. In [1], a complete architecture for wireless communication is proposed, which includes a hierarchical database structure to implement the HLR/VLR users location scheme.

Location Management is yet another important topic in this area, which is primarily concerned with designing strategies to keep track of mobile users. In [12] a number of search strategies are proposed to locate users. Another nice paper in location management is [5], which proposes a framework for modeling and evaluating the performance of location management schemes for a Personal Communication Services (PCS) network. In [14], user profile replication is proposed as a mechanism for rapid location lookup. A recent paper [13] considers the "span of access" under changing location conditions and offers a summary of research in this area.

Data Replication is considered in [3], which studies cache consistency for replicated data. In [21], dynamic data replication algorithms are proposed for tree and star networks. Another interesting paper in this area is [9] which considers the problem of data allocation in a large number of databases accessed through a wireless network, and proposes strategies to optimize communication costs.

Finally *data retrieval* from broadcasts is the topic of this paper. A number of papers have appeared on this subject, including [4, 17, 16, 15, 19, 20]. This set of papers forms the background of our work reported in this document. The papers from Rutgers [4, 17, 16, 15] primarily deal with data organization in a broadcast, e.g., index organization and structure and cache invalidation strategies. We, on the other hand, are concerned with how to determine optimal broadcast content and design "good' retrieval strategies for users. Note that these two (i.e., their and our work) are complementary. Our strategies and their strategies put together form a comprehensive broadcasting framework. The papers by Vaidya et al. deal with, among other issues, the notion of broadcast scheduling, i.e., when to broadcast particular items. The metric optimized is access time, i.e., response times. Aside from a basic difference in the broad objectives (we are concerned with broadcast content determination and client retrieval strategies), another point of departure of our work is that we are concerned not only with access time, but also with "tuning time" (the duration of time clients actively listen). Thus, both efficiency as well as power conservation, are of interest to us.

3 Architecture Issues

In this section we give a brief overview of the architectural assumptions made in this paper, both with regard to the underlying networked mobile environment as well as to the database.

3.1 Mobile Environment

We have used a general architecture of a mobile platform, which is given in Figure 1 [7]. It is a distributed architecture where a number of computers, *fixed hosts* and *base stations*, are intercon-

nected through a high speed *wired* network. Fixed hosts are general purpose computers which are not equipped to manage mobile units, but can be configured to do so. Base stations are equipped with wireless interfaces and communicate with mobile units to support data access.

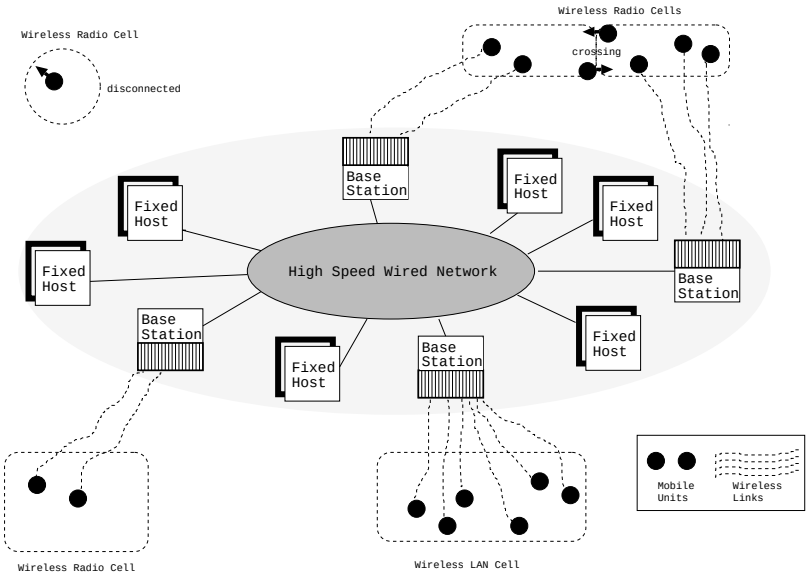

Figure 1: A General Architecture of a Mobile Platform

Mobile units are battery-powered portable computers, which move around freely in a restricted area, referred to here as a *geographic mobility domain*. The size restriction on a unit's mobility is mainly due to the limited bandwidth of wireless communication channels. To manage the mobility of units the entire geographic mobility domain is divided into smaller domains called *cells*. The mobile discipline requires that the movement of mobile units be unrestricted within the geographic mobility domain (inter-cell movement).

The mobile computing platform can be effectively described under the *client/server* paradigm. Thus sometimes we refer to a mobile unit as a *client* and sometimes as a *user*. The base stations are identified as *servers*. Each cell is managed by a base station, which contains transmitters and receivers for responding to the information processing needs of clients located in the cell. We assume that the size of a cell is such that the average query response time is much smaller than the time required by the client to traverse it. Therefore, it will seldom occur that a user submits a query and exits the cell before receiving the response.

Clients and servers communicate through wireless channels. The communication link between a client and a server may be modeled as multiple data channels, or a single channel [15]. We assume a single channel for reasons outlined in [15]. We further assume that this channel consists of both an uplink for moving data from client to server and a downlink for moving data from server to client.

3.2 Database Architecture and its Characteristics

Data replication and partitioning are broad areas of research themselves; they are not the concern of this paper. Here, we have assumed full database replication. The data is characterized as *rapidly*

changing [6]; users often query servers to remain up-to-date. More specifically, they will often want to track every broadcast for their data item of interest. Typical examples of this type of data are stock, weather, and airline information. We assume the following for fully characterizing our mobile database.

1. The database is updated asynchronously, i.e., by an independent external process, which does not affect the protocols designed in this paper. Also, such updates arrive with high frequency, signifying that the database is rapidly changing. Examples of such information are stock, weather, etc.

2. Users are highly mobile and randomly enter and exit from cells. There is a parameter called *Residence Latency* (RL) which characterizes the average duration of a user's stay in the cell. We provide a full explanation of RL in section 5.

3. User reference behavior is localized; e.g., some stocks are more popular than others.

4. Servers are *stateless*, i.e., they maintain neither client arrival and departure patterns nor client-specific data request information [15]. We assume a stateless server because we believe that the cost of maintaining a *stateful* server in a mobile environment would be prohibitively expensive. We want to emphasize, however, that our scheme will work with stateful servers as well.

4 Problem Statement

Wireless networks differ from wired networks in many ways. Database users over a wired network remain connected not only to the network, but also to a continuous power source. Thus, response time is the key performance metric. In a wireless network, however, both the response time and the active life of the user's power source (battery) are important. While a mobile unit is listening or transmitting on the line, it is considered to be in *active* mode. Assuming a power source of 10 AA batteries and a laptop equipped with CD-ROM and display, estimated battery life in active mode is approximately 2.7 hours [15].

In order to conserve energy and extend battery life, a clients slips into *doze* (stand by) mode, in which she is not actively listening on the channel. Clients expend significantly less energy in doze mode than in active mode. Therefore, one of the major goals of our scheme is to minimize the amount of time a client must spend in active mode to retrieve the data items she requests.

The problem addressed in this paper may be captured by the following question: *given that users are highly mobile in their mobility domain, what are good strategies that the server can use to decide on what to broadcast?* The assumption is that such strategies need to adapt to user demand patterns in the highly mobile environment. We are also interested in the question of retrieval strategies: *given that good broadcast strategies are found, what are good retrieval algorithms by which users can retrieve/download data from broadcast, with a minimum of energy expenditure?* The basic idea, inspired by suggestions in [15], is one of "mixed broadcasting", i.e., automatic, as well as on-demand broadcasting.

We follow a similar approach to that presented in [15] for characterizing the performance of our algorithms. We define the following terms:

Access Time (AT): Access time refers to the time elapsed between query submission and receipt of the response.

Tuning Time (TT): Tuning time is the duration of time that the client spends actively listening

on the channel.

The meaning of these terms is illustrated in figure 2. Consider a client who submits a request at time T_0 and receives the response at time T_7.

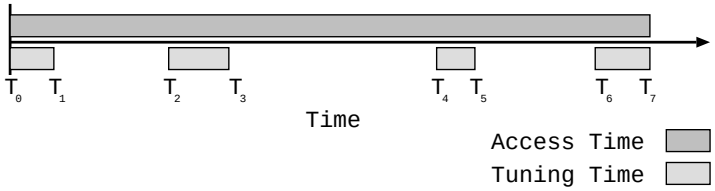

Figure 2: Access and Tuning Times

In this scenario, if the client listens continuously from the time a query is submitted until the response is received, then $AT = TT = (T_7 - T_0)$. On the other hand, if the client slips into doze mode intermittently, then TT is noticeably less than AT, significantly reducing battery usage. In this case $AT = (T_7 - T_0)$, and $TT = (T_7 - T_6) + (T_5 - T_4) + (T_3 - T_2) + (T_1 - T_0)$. This results in energy conservation, as the client is in active mode for only short periods of time. The question, of course, is how to determine the smallest possible tuning intervals. An ideal approach appears to be providing the client with *precise* knowledge of when her requested information will be broadcast.

Our aim is to find optimal points in the two dimensional space of AT and TT. This becomes difficult because there appears to be a trade-off between AT and TT; attempts to reduce one tend to increase the other. For example, access time is directly related to the size of the broadcast, i.e., AT is smaller for a smaller broadcast size. On the other hand, providing information for *selective auto-tuning*, i.e., informing the user precisely where its required data is located in the broadcast, reduces tuning time. However, inclusion of such tuning information would increase the overall size of the broadcast by including overhead, which in turn could increase AT. Conversely, eliminating this overhead will reduce AT at the expense of an increased TT, because the user will not know precisely when to tune in.

5 Preliminaries

Before delving into details of our proposed strategies, we first describe certain notions that we will use in developing those strategies.

Broadcast Set and Content: *Broadcast set* refers to the set of data items included in a broadcast. *Broadcast content* refers to the composition of a broadcast, i.e., the data tuples along with access and overhead information included in a broadcast. Note that while the broadcast set of two successive broadcasts may be identical, the content may change due to changes in the actual data values that may have occurred in the meantime. For example, while IBM stock prices at various markets may be broadcast repeatedly, the actual data tuples corresponding to the IBM prices may change. A major goal of this paper is the devise strategies to determine what the broadcast set should be, given the high degree of mobility as well as particular demand patterns of clients. In particular the server must make two decisions: (a) identify items to include in broadcast, and (b) determine the temporal duration that a particular data item needs to be broadcast, once it is included.

Broadcast Periodicity: The periodicity of broadcast is important as it affects both the access

and tuning times. We consider both periodic and aperiodic broadcasts in our strategies.

Data Organization: Data organization refers to how the broadcast content is organized, e.g., how data and indices are interleaved. In [15], data organization has been addressed very nicely and is not a research issue in our work. We make use of the results presented in [15]. In particular, for our strategies, we assume a `(1,m)` indexing strategy as presented in [15]. This strategy posits that a complete index for the broadcast is repeated every $(\frac{1}{m})^{th}$ of the broadcast. In other words, the entire index occurs m times during a broadcast, where m is a parameter.

Residence Latency (RL) and Estimated Departure Time (EDT): When the server decides to include an item in its broadcast, it also needs to decide the length of time this item will remain in its broadcast set. In order to identify the residency duration of a data item in a broadcast, we propose to associate a *Residence Latency* (RL) value with each cell. RL identifies the duration an item would remain in a broadcast for that cell. The RL value for a specific cell is the average length of time a mobile user resides there. RL for a cell could be computed *a priori* based on the advance knowledge of user movement patterns and cell geography. We believe this is feasible based on information currently available. A data item's *Expected Departure Time* (EDT) from a broadcast is computed by adding the item's entry time into the broadcast and the cell's RL.

Popularity Factor: The popularity factor of item X at time T, denoted by PF_X^T, identifies the number of clients in the cell at time T who are interested in X. Henceforth we shall drop the use of the time superscript in the PF expression without any loss of generality. The PF of item X is maintained in the system as follows. When a client requests X, PF_X is increased by 1. However, every time PF_X is incremented, the system records the corresponding time. Let the timestamp of the i^{th} increment to PF_X be denoted by T_X^i. Then, a corresponding decrement of 1 is performed on the value of PF_X at time $(T_X^i + RL)$. This reflects the (anticipated) departure of the client whose request caused the i^{th} increment.

Dynamism of the Underlying Database: It has been widely suggested (e.g., see [15]) that certain information that people are likely to access in a mobile fashion would be dynamic, e.g., stock information, weather and traffic. This *rapidly changing* nature makes these databases more difficult to manage. In this paper we are interested in these types of systems.

The *rapidly changing* nature of the database has some effects on the broadcasting strategy. In particular, in this scenario, there is a reasonable likelihood that the database state will change from one broadcast to another. Such changes often occur in bursts, e.g., a large fraction of stock trades occur in the last hour of trading. At these times, clients are particularly interested in retrieving current values for their data items of interest. Moreover, at these times, clients would often download data items from every broadcast to keep themselves apprised of the rapidly changing current state. It is easily seen that this can be quite expensive, as the client needs to tune into every broadcast. We are particularly interested in minimizing a client's cost (i.e., energy expenditure) in such scenarios. Our basic strategy may be explained through the following example. Assume that a client is interested in a particular data item , corresponding to which, there exist B buckets in a broadcast. Assume further that the client would like to download the same information in the following broadcast. Now, if only B_{dirty} buckets have changed, where $B_{dirty} < B$, it is then advantageous for the client to simply download the dirty (i.e., modified since the previous broadcast) buckets from the subsequent broadcast, rather than the full set of B buckets. One can easily see that such savings quickly accumulate over a few successive broadcasts. In order to execute this

strategy, we shall maintain certain special metadata in buckets which will allow clients to selectively tune in to dirty buckets. That, and our entire broadcast structure, is explained below.

5.1 Broadcast Structure

Broadcast structure refers to the specific broadcast organization that we assume in developing our strategies. It is important to understand this structure in order to properly appreciate our protocols. As mentioned before, we assume a `(1,m)` indexing strategy outlined in [15]. In this scheme, index information is provided at regular intervals in the broadcast. More specifically, a complete index is inserted m times in a broadcast at regular intervals.

Figure 3 illustrates our broadcast structure. A broadcast is a sequence of *data blocks* (containing data) and *index segments* (containing access information) as shown in figure 3A. Using the `(1,m)` data organization methodology, an index segment appears every $(\frac{1}{m})^{th}$ of the broadcast, i.e., there are m index segments. Clearly, each of the m data blocks is also of equal size. Each data block is composed of one or more *data clusters* as shown in figure 3B, where each data cluster consists of a collection of tuples of a single data item of interest. For example, assume the broadcast consists of stock information, and each tuple is a triple <*stock_id, price, market*>. In such a scenario, the data items of interest would be represented by stock ids. Consider a particular stock id, e.g., IBM. All IBM records would comprise a data cluster. A data cluster may span more than one data block.

Data clusters are composed of data buckets (figure 3C) which contain data records as well as some tuning information (denoted by the 4-tuple $< X, Y, Z, E_B >$ in the figure) explained below. We assume that each client has her own item of interest (e.g., clients are not interested in all stocks, but instead in specific ones). For the purposes of this study, we assume a client has a single data item of interest. As explained above, all records pertaining to this item appear in a specific data cluster which we refer to as the client's *Data Cluster of Interest* (DCI). Within the broadcast, the data clusters are organized in order of decreasing popularity based on *PF*, such that the most popular item will be broadcast first, and the least popular item will be broadcast last. This helps to reduce the access times for popular items.

An index segment is a series of buckets containing index tuples and some other special tuning information. We first describe the index tuples. Each index tuple consists of a 4-tuple, $< K, B, C, E_C >$, that not only informs the client precisely where the `DCI` appears in the broadcast, but also provides information about updates to the cluster since the previous broadcast. The structure of the index segment is shown in figure 3D. *K, B, C* and E_C are defined below.

K: the cluster's key value (e.g., for an IBM cluster, the key value is IBM).
B: the id of the first bucket of the data cluster.
C: the offset from B to the first dirty bucket (bucket where changes have occurred since the last broadcast) of the data cluster. If all buckets in the data cluster are clean, C takes a default value of -1.
E_C: the *EDT* of the cluster key, i.e., when the cluster is scheduled to be dropped from the broadcast.

The dirty/clean information (i.e., B and C) are included to handle the *rapidly changing data* scenario explained earlier in this section. We assume a B-tree structure for the index. Thus, clients must begin reading the index at the root in order to find the pointers to their `DCI`s.

As mentioned above and shown in figures 3C and D, all buckets, whether index or data, have a special tuple displayed as a 4-tuple $< X, Y, Z, E_B >$. This information is provided to orient clients

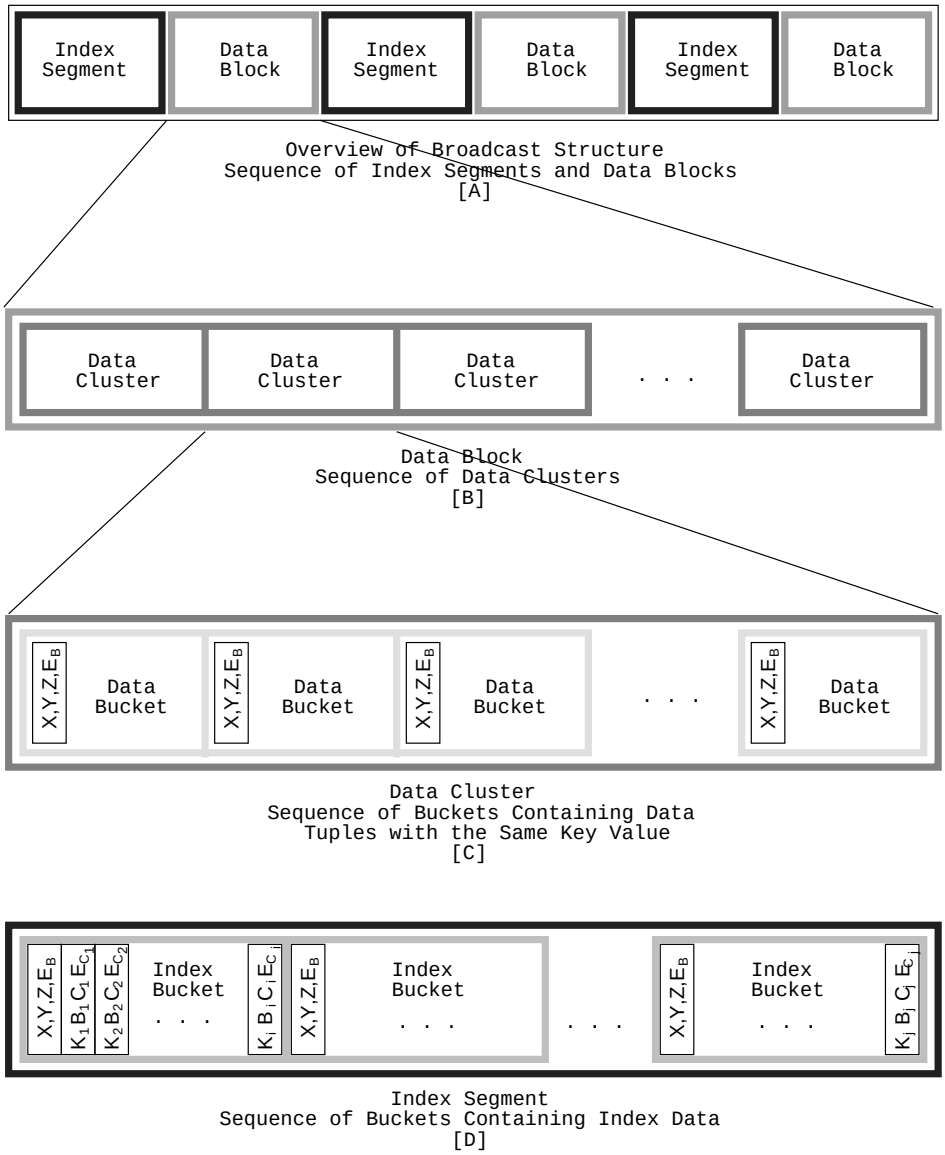

Figure 3: Broadcast Structure

as they initially tune in to a broadcast. The X, Y, Z and E_B terms are defined as follows.

X**:** An offset to the first bucket of the next nearest index segment.

Y**:** An offset to the end of the broadcast, i.e., the start of the next broadcast.

Z**:** Shows the bucket's type (data or index) and contains tuning information for items updated since the previous broadcast. It can hold one of four possible values:

$Z =$ **-2** indicates an index bucket.

$Z =$ **0** indicates a data bucket, and that the bucket is *clean*; i.e., unmodified since the previous

broadcast.

$Z = i$, where i is a positive integer, indicates a data bucket, and that the bucket is *dirty*; i.e., modified since the previous broadcast. Moreover, the actual i value, i.e., the positive integer, is an offset to the the next dirty bucket in the same data cluster.

$Z =$ **-1** indicates a data bucket, and that it is the last dirty bucket of the data cluster.

E_B: The EDT of the data item in the bucket. Obviously, the E_B value of every bucket in the same data cluster is going to be identical and equal to the EDT of the of the cluster key (e.g., in an IBM cluster, all E_B values will be equal to the EDT of the IBM data item.

While the reasons for including X, Y and E_B are obvious, the utility of Z is not. The Z values are used to facilitate efficient retrieval in rapidly changing data scenarios explained earlier in this section. This will be illustrated in section 6.2.

We fix the unit of retrieval as a *bucket*, following the lead of [15], where a bucket is defined as the smallest logical unit of broadcast. All buckets (regardless of data or index) are of equal size. This allows for a direct mapping between time and buckets. In fact, time is measured in terms of buckets in this paper. This means that all the offsets referred to in the preceding description are in terms of buckets.

Finally we add a note regarding the notion of data clusters. A data cluster consists of a sequence of buckets containing data tuples pertaining to a specific key, such as a specific stock id. Moreover, we assume that *selectivity*, i.e., the number of buckets required to broadcast a specific data item, is fixed. This is a reasonable assumption. Consider, for example, stock information expressed as a 3-tuple <*stock_id, market, price*>. Clearly for a particular key value (say IBM), the number of tuples is equal to the number of markets (e.g., New York Stock Exchange) where the stock is traded. This number is fairly constant, making the number of buckets for a particular cluster fairly stable. We also assume that specific buckets within a cluster have identical semantic content from one broadcast to another. For instance, the first bucket within the IBM cluster will always contain the tuples related to the IBM prices in New York and Frankfurt (assuming there are 2 tuples per bucket). Thus while the individual prices may change, the semantic content of each bucket within a cluster is invariant.

6 Server and Client Protocols to Support Adaptive Broadcast Content and Efficient Retrieval

This section proposes adaptive broadcast protocols which seek an optimal balance of access time (quality of service or average query response time) and tuning time (energy consumption). We aim to develop *cooperative* strategies between clients and servers for achieving such a balance.

As mentioned earlier, *periodicity* is an important parameter in designing broadcast strategies. A periodic broadcast signifies that the broadcast "size" (i.e., number of buckets) is fixed. One can ascribe both advantages (e.g., predictability) as well as disadvantages (e.g., loss of flexibility) to periodicity. To study such effects, we describe two sets of protocols below for the periodic and the aperiodic cases. We refer to the periodic protocol as the *constant broadcast size* (CBS) strategy, whereas the aperiodic broadcast protocol is termed a *variable broadcast size* (VBS) strategy.

Finally, note that we support a *mixed mode* (this term is borrowed from [15]) retrieval policy, i.e., when a client arrives in a cell, she first tunes in to the broadcast to see if her `DCI` is already

there. If not, the client explicitly sends a request to the server through the uplink for that item. Thus items may be found readily in the broadcast, or may have to be placed "on demand". This policy has been deemed the most "general" policy in the literature.

6.1 Constant Broadcast Size Strategy

We first present the server protocol, i.e., the strategy used by the server in deciding upon the broadcast content. We then present the client protocol, i.e., how the client retrieves data from the broadcast.

6.1.1 CBS Server Protocol

In this strategy, broadcast size is limited, and the broadcast is periodic. Periodicity mandates an equal size for every broadcast (recall that we consider both size and time in terms of buckets). If there are too few requested items to fill the broadcast period, the broadcast will contain dead air. On the other hand, if there are more requested items than space in the broadcast, the server must prioritize requested items to decide which to include in the broadcast set. We propose that this prioritization mechanism simultaneously satisfy two properties: *popularity consciousness* and *avoidance of chronic starvation*. Popularity consciousness means that items that are requested more often should have a greater chance of being included in the broadcast than less popular items. Avoidance of chronic starvation means that if a client requests a "less popular" item, he should not be chronically starved, i.e., the item should appear in the broadcast at some point during that client's residence in the cell. At a minimum, our protocol attempts (but does not guarantee) to provide access to a requested data item at least once during a client's probable stay in the cell; that is, within RL time of the request.

We propose a system of priority ranking of items based on two factors: a *Popularity Factor* (PF) and an *Ignore Factor* (IF). *PF* is computed as described in section 5. We describe the purpose and definition of *IF* below.

Ignore Factor

Consider a strategy where the inclusion priority of items is based purely on popularity, i.e., items are included in the broadcast based on a *Highest-PF-First* (HPF) strategy. Under such a strategy, assuming a large number of clients in a cell and a data set in which a small subset of items are very popular, clients requesting less popular items may be chronically starved. That is, items with a very low *PF* may be overwhelmed by the large number of requests for very popular items, and will not be broadcast while the requesting client is still in the cell. Recall that in our operating scenario, clients are highly mobile; thus, if a request is ignored for a long time, it is likely that the client will leave the cell without receiving service for that item throughout its entire residence in that cell. From a Quality Of Service standpoint, this is highly undesirable. At the very minimum, our goal is to attempt to service a client's requests at least once during his stay in the cell where the request was made. The purpose of *Ignore Factor* (IF) is to ensure that less-popular but long-neglected items get an opportunity to be included in the broadcast.

We first explain the notion of *IF*. Assume a data item, say X, was last reported in the broadcast ending at time T_0, after which it was dropped. At this point, the Ignore Factor of X, denoted by IF_X, is defined to be 1, the base value for any item's Ignore Factor (the reason for choosing

a minimum *IF* of 1, and not 0, will be explained later). Further assume that the first request for X after time T_0 arrived at time T_{req}. Subsequent to this point, whenever X is not reported in a broadcast, we say that X has been *ignored*. More specifically, the ignore factor of X at any time T_i (assuming X has not been reported until T_i), $T_i > T_{req}$, denoted by $IF_X^{T_i}$, is given by: $IF_X^{T_i} = NumBroadcast(T_i, T_{req})+1$, where $NumBroadcast(T_i, T_{req})$ denotes the number of broadcasts between T_{req} and T_i. Since we are considering a periodic broadcast scenario, this number is easy to compute. In particular, assuming a broadcast period of P_B, $NumBroadcast(T_i, T_{req}) = \left\lfloor \frac{T_i - T_{req}}{P_B} \right\rfloor$. In other words,

$$IF_X^{T_i} = \left\lfloor \frac{T_i - T_{req}}{P_B} \right\rfloor + 1 \tag{1}$$

When talking about the *IF* of a particular data item from hereon, we will often drop the temporal superscript, i.e., T_i, unless the context warrants its explicit inclusion.

Note that equation 1 is only valid under the constraint $(T_i - T_{req}) \leq RL$. To see this, consider a client C, who requested data item X at time T_{req}. Assume no other requests arrive for this data item. Then, at any time T_i following T_{req}, IF_X is given by equation 1 above. However, following the definition of RL, client C would be expected to depart at $T_{dep} = T_{req} + RL$. Thus after T_{dep}, there is no point in reporting X any longer, as its requester may have left. Thus, after T_{dep}, *unless other requests for X have arrived in the meantime*, IF_X should be reduced to 1 (the base value for any item's *IF* is 1; this is explained below. A corollary of this is that the following inequality holds, for any data item X_i,

$$1 \leq IF_{X_i} \leq \left\lfloor \frac{RL}{P_B} \right\rfloor + 1$$

While the above discussion lays the foundation for the fundamental ideas behind the notion of ignore factor, we still have to describe a general procedure for computing ignore factor. To this end, let us consider a data item X, for which a stream of requests arrive. In this case, computation of IF_X assumes greater complexity than discussed before. To see this, consider the case where two requests arrive for X, at times T_{req1} and T_{req2} respectively, where $T_{req2} > T_{req1}$. Assume X is not included in any broadcast until time $T_{dep1} = T_{req1} + RL$. At T_{dep1}, according to the discussion in the two preceding paragraphs, $IF_X = \left\lfloor \frac{RL}{P_B} \right\rfloor + 1$. However, as soon as the clock ticks past T_{dep1}, i.e., after T_{dep1}, the request that came in at T_{req1} is considered void. Thus it should no longer determine IF_X. Rather, after T_{dep1}, IF_X should be computed based on T_{req2}. More specifically, at all times during the interval $[T_{dep1}, T_{dep2}]$ (excluding the initial temporal instant), where $T_{dep2} = T_{req2} + RL$, IF_X is given by $\left\lfloor \frac{current\_time - T_{req2}}{P_B} \right\rfloor + 1$.

From the simple two-request example above, one important fact emerges: in order to maintain an accurate record of an item's *IF*, one must track every request that arrives for that item. With a large number of items and clients, such maintenance could be prohibitively expensive. In particular, the complexity of this process for a specific data item is of order $O(N)$, where N is the number of requests for that item. However, we have devised a mechanism by which the *IF*s of items can be maintained accurately at minimal cost. In particular, as will be clear later, our strategy records *IF* for a data item in constant time.

We first provide the intuition for this process and then describe it in detail. To see the basic idea behind our mechanism, consider figure 4, in which we show a timeline with particular points of interest. The points marked with the darker lines, i.e., B_1, B_2, B_3, B_4, denote broadcast initiation epochs, i.e., instants at which successive broadcasts start. The i^{th} request for data item X arrives at

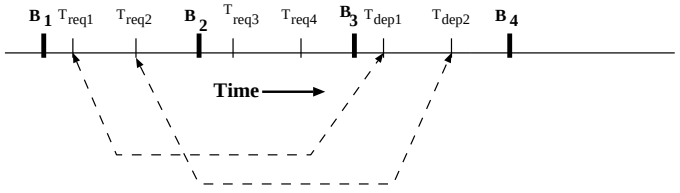

Figure 4: Computation of Ignore Factor

T_{reqi}. In figure 4, we show four requests. In this example, we assume $RL = 2 \times P_B$. Corresponding to the request points, the departure points are shown as T_{depi}. According to the discussion above, we need to keep track of all four requests in order to accurately compute IF_X at any point. However, note that *IF* values will only be used at the time the server decides on the next broadcast set, i.e., immediately preceding the broadcast epochs. Consider, as a specific example, the broadcast epoch B_4. At B_4, in figure 4, the request at T_{req1} has expired. Note that the request at T_{req2} has also expired. Moreover, we can safely say that at B_4, all requests that arrived prior to B_2 would have expired. In general, at all decision points where the next broadcast set has to be decided, i.e., at successive broadcast epochs, all requests that arrived prior to the $\left(\left\lfloor \frac{RL}{P_B} \right\rfloor\right)^{th}$ broadcast preceding the next immediate broadcast are guaranteed to have expired. *Thus,* IF_X *needs to be computed based on the first request that arrived after the* $\left(\left\lfloor \frac{RL}{P_B} \right\rfloor\right)^{th}$ *broadcast preceding the next immediate broadcast.* This is the key intuition. For example, at B_4, IF_X needs to be computed based on T_{req3}. The significance of this intuition is that the system need only keep track of the first request after broadcast epochs and can ignore all successive requests until the next broadcast epoch. Moreover, only the last $\left\lfloor \frac{RL}{P_B} \right\rfloor$ epochs need to be considered. In other words, the system need only maintain a list of $\left\lfloor \frac{RL}{P_B} \right\rfloor$ requests.

This may be achieved by maintaining an ordered list in memory. Elements in this list will contain both the tid and timestamp of the request. The list has $\left\lfloor \frac{RL}{P_B} \right\rfloor + 1$ elements, with tids 1 through $\left\lfloor \frac{RL}{P_B} \right\rfloor + 1$, where the timestamp corresponding to the i^{th} element yields the timestamp of the first request for X after the i^{th} previous broadcast epoch with respect to the next immediate broadcast. The list is updated upon the arrival of the first request for a data item after each broadcast epoch. Such a list can be implemented as a circular queue. Using such a structure, an item can be accurately tracked in constant ($\approx \left\lfloor \frac{RL}{P_B} \right\rfloor$) time.

Priority computation using IF and PF: Having explained *IF* we now turn our attention to computing the inclusion priority for data items. We propose that an item's priority be computed based on the following expression

$$Priority = IF^N \times PF \tag{2}$$

where N is an adaptive scaling factor described below. Equation 2 also explains why we set the base value of *IF* at 1 and not 0. It is easily seen that, at an *IF* of 0, the priority value (i.e., for popular items) is reduced to 0, which guarantees the item's exclusion from the broadcast.

Equation 2 exploits the counteracting trend of *IF* and *PF*. Using the above expression, an item which has been requested by a large number of clients will have a large *PF*. It is thus likely to appear in many broadcasts, and will have a relatively low *IF*. On the other hand, an item with few requests will have a low *PF*. Based on its low *PF*, it is likely to be omitted from broadcasts,

increasing its *IF* and therefore its chances of being included in a future broadcast set.

N is an adaptive exponential weighting factor based on a nearest neighbor approach. Its purpose is to increase the likelihood that items which have been ignored for a long time will appear in the broadcast. *PF* and *IF* differ largely in scale; if N is relatively low, *PF* dominates the priority expression (limited by the number of clients in a cell). If, however, an item has been ignored for a long time, we would like *IF* to dominate. A larger N value will achieve this.

In order to precisely define N, let W_{ij} denote the number of broadcasts client i waited for item j after requesting item j. Let C_j denote the set of clients who are waiting for j. Then, the *average wait count* (AWC) for j is given by,

$$AWC_j = \frac{\Sigma_{i \in C_j} W_{ij}}{|C_j|}$$

Let the *desired wait count* (DWC) denote a quality of service measure defined for the system. DWC denotes, on an average, the maximum number of broadcasts a client can reasonably be expected to wait before receiving service for her desired item. When computing the priority of an item, we compare AWC and DWC. If $AWC > DWC$, N is incremented by unity. If $AWC = DWC$, N remains unchanged. If $AWC < DWC$, N is decremented by unity. N is initialized to a base value for 1 for all data items.

Having explained the underlying concepts, we are now prepared to describe the server protocol for constructing a broadcast. Prior to a broadcast epoch (the time at which a new broadcast is scheduled to begin), i.e., in its broadcast preparation stage, the server prioritizes all items which have been requested, i.e., items with a $PF > 0$, and sorts the items in order of descending priority. It then adds items to the broadcast set until the broadcast is full. For all requested but excluded items, *IF* is adjusted.

6.1.2 CBS Client Protocols

We now describe client protocols designed to smartly retrieve data from the broadcast in cooperation with the server protocol defined above. When a client senses the need for a particular data item, he begins the retrieval process by tuning in to the broadcast at an arbitrary time and reading a bucket. We remind the reader that the data cluster in the broadcast that holds the item of a client's interest is referred to as the *Data Cluster of Interest* (DCI).

The random initial probe in a continuous flow of broadcasts creates a large number of tuning possibilities. We identify seven mutually exclusive initial tuning possibilities. Note that because we assume a B-tree index structure, the client must start reading from the top of the index segment (i.e., the root). If she does not find a pointer to her `DCI` (i.e., `DCI` is not in the current broadcast set), then she requests the item and tunes to the initial index of every succeeding broadcast until either the `DCI` is found in broadcast, or the client departs from the cell. We discuss the cases below. We assume, for simplicity, that any probe into a broadcast always puts the client at the top of a bucket, i.e., we ignore the fact that the client may tune in at any point in a bucket. The error introduced by this, as easily seen, is very small (0.5 bucket on average).

Case 1: Client initially tunes in to an index bucket, DCI exists in current broadcast, and the client has tuned in before DCI in the broadcast

In this case the client tunes in to an index segment. If he has missed the root, he must tune out until the top of the next nearest index segment. This is efficiently achieved by noting the X value

(described in section 3) from the 3-tuple of tuning information provided with every bucket. Upon reading the index, the client finds that his DCI has not yet been broadcast, and tunes out until the beginning of his data cluster of interest. At the beginning of his DCI, he tunes back in and downloads it. Thus the retrieval protocol in this case is given in Protocol 1.

Protocol 1 CASE 1

```
access top of next nearest index segment
read index;
wait until beginning of DCI;
download DCI;
```

Case 2: Client initially tunes in to an index bucket, DCI exists in current broadcast, and the client has tuned in after DCI in the broadcast

As in the previous case, the client tunes in to the top of the next nearest index segment, and becomes aware that her DCI has already passed by comparing the current bucket id with the B value (explained in section 3) in the index tuple corresponding to her data item of interest. She also reads the E value in that index tuple to determine if her DCI is scheduled for removal from the broadcast set following the current broadcast. This is easily achieved by comparing the EDT (i.e., the E value) of the data item with the start time of the next broadcast, which is easily derivable from the Y value of any tuning tuple. If her DCI will appear in the next broadcast, she tunes out until the top of the next broadcast, reads index information and downloads her DCI as outlined in the previous case. If her DCI is scheduled to be dropped, she submits a request for it, and keeps tuning in to the top of every subsequent broadcast until a pointer to it appears in the index, or she leaves the cell. The pseudocode for Case 2, combined with that of Case 3, is shown below.

Case 3: Client initially tunes in to an index bucket, DCI does not exist in current broadcast This case is very similar to the second subcase in case 2. After becoming aware of the non-existence of his DCI in current broadcast, by appropriate access into an index segment, the client submits a request for the data item, and keeps tuning in to the top of every subsequent broadcast until his DCI appears, or he leaves the cell. The protocol is as given in Protocol 2.

Protocol 2 CASE 2 & 3

```
access top of an index segment;
read index;
if DCI missed or excluded from current broadcast then
    wait until beginning of next broadcast;
    if DCI not in index then
        submit request for DCI;
        while DCI not in index
            wait until beginning of next broadcast;
            read index;
    wait until beginning of DCI;
    download DCI;
```

Case 4: Client initially tunes in to a data bucket, DCI exists in current broadcast, and

the client has tuned in before DCI in the broadcast

The procedure is exactly the same as in case 1.

Case 5: Client initially tunes in to a data bucket, DCI exists in current broadcast, and the client has tuned in after DCI in the broadcast

Same as case 2.

Case 6: Client initially tunes in to a data bucket, DCI does not exist in current broadcast

Same as case 3.

Case 7: Client tunes in to a data bucket in the middle of the `DCI` in the current broadcast

This is the most complex of all cases, and also the most important, as the same protocol would be used by clients who are interested in downloading their `DCI`s continually from successive broadcasts. In this case the client tunes in and becomes aware that her `DCI` is currently being broadcast. She immediately begins to download the data. Since she missed a portion of her data cluster of interest in the current broadcast, the remainder must be obtained from the next broadcast. At the end of the data cluster, having already determined whether her `DCI` is scheduled to be dropped from the next broadcast (through methods outlined earlier), she tunes out and waits for the beginning of the next broadcast. If her `DCI` is going to be dropped, she sends a request on the uplink and keeps tuning in to successive broadcasts until her `DCI` is found in the index. The more interesting case occurs when she determines that her `DCI` is not going to be dropped, i.e., it is going to appear in the next broadcast. In this case, as discussed in the "rapidly changing" data description in section 5, she would like to download not only the portion of her `DCI` missed in the previous broadcast, but also the buckets that have been modified between the broadcast epochs. This is achieved as follows: The client downloads the portion of data missed in the earlier broadcast until the point at which she began downloading data in the previous broadcast. She then selectively tunes in and out to retrieve any updates (following the linked list of dirty buckets) to the data downloaded in the previous broadcast until the last dirty bucket in her data cluster of interest. Protocol 3 and Procedure 1 provide details regarding how the selective tuning is achieved.

6.2 Variable Broadcast Size Strategy

Having discussed a periodic broadcasting strategy, we now turn our attention to an aperiodic broadcasting scenario. We call this strategy the *variable broadcast size* (VBS) strategy, for obvious reasons. Note that while the broadcast size varies across broadcasts, at the start of each individual broadcast the size is known; therefore, the start of the subsequent broadcast is known as well. However, the server has no knowledge beyond that, as it does not know what requests may arrive during the current broadcast.

6.2.1 VBS Server Protocol

The server protocol for VBS is much simpler than that for the constant size strategy. All requested items are included, i.e., all items with a PF greater than 0 are added to the broadcast set. The broadcast length changes as items are added and deleted. Items remain in the broadcast for RL units of time from their latest request, and are subsequently dropped from the broadcast set.

Protocol 3 CASE 7

```
access middle of DCI;
download remaining buckets of DCI;
wait until beginning of next broadcast;
read index;
if DCI in index;
    wait until beginning of DCI;
    execute procedure DYNAMIC_READ;
else
    submit request for DCI;
    while DCI not in index
        wait until beginning of next broadcast;
        read index;
    wait until beginning of DCI;
    download DCI;
```

Procedure 1 DYNAMIC_READ

```
if DCI_index.C > 0 then index tuple C value of the DCI*/
    next_dirty ← current_bucket.id + DCI_index.C; /* find 1st dirty bucket in cluster */
else
    next_dirty ← −1;
while not at point in DCI where began reading in previous broadcast
    download bucket;
    if current_bucket.Z = −1 then entire data cluster is clean*/
        next_dirty ← −1;
    else
        if current_bucket.Z > 0 then
            next_dirty ← (next_dirty + current_bucket.Z);
    advance to next bucket;
if next_dirty = −1 then /*the rest of DCI is clean*/
  exit;
wait until next_dirty;
while (true) /*read all remaining dirty buckets in DCI*/
  if current_bucket.Z = −1 then
    {download bucket;
    exit;}
  else
    next_dirty ← (next_dirty + current_bucket.Z)
    download bucket;
    wait until next_dirty;
```

Within the broadcast, items (i.e., DCIs) are ordered based on descending popularity. Since no item is ignored, there is no notion of ignore factor in VBS.

6.2.2 VBS Client Protocols

The client protocols in this strategy are similar to those of the CBS strategy. The main difference between these strategies is in the client's response to finding that his `DCI` is not in the broadcast. Here, if his `DCI` is not in the broadcast, or if he has missed his `DCI` and the item will be dropped when the next broadcast is composed, the client requests the item and exits from the protocol. Since his `DCI` is guaranteed to be in the succeeding broadcast, he begins the retrieval process at the beginning of the next broadcast and finds his `DCI` in that broadcast.

7 An Approximate Analysis for the Constant and Variable Broadcast Size strategies

In this section, we are primarily interested in obtaining expressions for the expected access time (AT) and tuning time (TT) for both the *constant broadcast size* (CBS) and the *variable broadcast size* (VBS) strategies.

Both strategies, as the reader may recall from section 6, are presented as a set of mutually exclusive tuning scenarios. For each scenario, we compute AT and TT. We also compute the probability of encountering each case. Then, using elementary probability theory, expected access time ($E[AT]$) and expected tuning time ($E[TT]$) may be stated as:

$$E[TT] = \sum_{\forall i} P\{Scenario\ i\} TT_i, \qquad E[AT] = \sum_{\forall i} P\{Scenario\ i\} AT_i \tag{3}$$

In order to achieve analytical tractability, we make several assumptions in performing our analysis. Thus our results are approximate. Our assumptions are as follows: (a) clients read from the broadcast in units of buckets; (b) clients always tune in to the start (i.e. the top) of a bucket; (c) if the client tunes in to the middle of an index; segment, it has missed the root of the B-tree, and must wait for the beginning of the next nearest index segment; (d) a cell contains N clients, and each client can request at most one `DCI` during a single broadcast period; and (e) there are two classes of data items. A *hot* item has a higher probability of being requested by the clients than a *cold item*. h percent of `DCI`s are *hot* and $1 - h$ percent are *cold items*.

In addition, certain special assumptions are made for the CBS and VBS strategies. These assumptions are stated preceding each respective analysis section. The next two subsections list our results. Table 1 lists the notations used in our analysis.

Our analysis is probabilistic, i.e., we compute the probabilities of certain events. The events are summarized in Table 2.

7.1 Derivation of Event Probabilities

We start out by deriving probabilities for events that we consider in our analysis, outlined above in Table 2. Note that all of the aforementioned events are valid for both CBS as well as VBS strategies.

Notation	Meaning
d	size of data to broadcast (in items)
S	selectivity of a data item (in buckets)
n	capacity of a bucket (tuples per bucket)
m	number of index segments or data blocks per broadcast
B	size of data block (buckets per block) $= \frac{d}{nm}$
I	size of index segment (in buckets) $= \frac{2d}{Sn}$
T_B	size of the broadcast in CBS $= m(I+B)$
b	number of DCI $= \frac{d}{Sn}$
C_{CBS}	capacity of broadcast in CBS (in DCI)
l	size of an index segment (in buckets) $= \log_{n+1} I$,
L_1	expected distance to DCI from the end of index segment (buckets)
L_2	expected distance to next broadcast from end of index segment (buckets)
p_h	probability that a client requests a *hot* item
p_c	probability that a client requests a *cold* item
$p_{h,in}$	probability that a *hot* item is in broadcast
$p_{c,in}$	probability that a *cold* item is in broadcast
P_{in}	probability that an item is in broadcast
h	fraction of *hot* items in the d
c	fraction of *cold* items in the d
p_0	probability that a client does not request any DCI
k	total number of DCIs in the database

Table 1: Notations used in the Analysis

Event	Meaning
M	DCI is missed completely
$\bar{M}$	DCI is downloaded completely (i.e., not missed at all)
PM	DCI is missed partially
D_i	DCI is in the i^{th} data block
TI_i	tune in to i^{th} index segment first
TD	tune in to a data block first
TI	tune in to an index segment first
PH	client requests a *hot* item
PC	client requests a *cold* item

Table 2: Probabilistic Events

Derivation of $P\{\bar{M}, TI\}$ Given that a client tunes in to an index segment, the probability that the client does not miss `DCI` at all is denoted by $P\{\bar{M} \mid TI\}$.

$$\begin{aligned} P\{\bar{M} \mid TI\} &= \sum_{i=1}^{m} P\{TI_i \cap \{D_{i+1} \cup D_{i+2} \cdots \cup D_m\}\} \\ &= \sum_{i=1}^{m-1} P\{TI_i\} P\{D_{i+1} \cup D_{i+2} \cdots \cup D_m\} \\ &= \sum_{i=1}^{m-1} \frac{1}{m} \frac{m-i}{m} = \frac{m-1}{2m} \end{aligned} \quad (4)$$

The probability that the client tunes in to an index segment is denoted by $P\{TI\}$ (the ratio of the total size of index segments over the total broadcast size). That is,

$$P\{TI\} = \frac{mI}{m(I+B)} = \frac{I}{I+B} = \frac{\frac{2Data}{Sn}}{\frac{2Data}{Sn} + \frac{Data}{nm}} = \frac{2m}{2m+S} \quad (5)$$

$$P\{\bar{M}, TI\} = P\{\bar{M} \mid TI\}P\{TI\} = \frac{m-1}{2m+S} \quad (6)$$

Derivation of $P\{\bar{M}, TD\}$

$$\begin{aligned} P\{\bar{M} \mid TD\} &= 1 - P\{M \mid TD\} - P\{PM \mid TD\} \\ &= 1 - \frac{m-1}{2m} - \frac{B-S}{mB} - \frac{S}{mB} \\ &= \frac{m-1}{2m} \end{aligned}$$

$$P\{\bar{M}, TD\} = P\{\bar{M} \mid TD\}P\{TD\} = \frac{m-1}{2m}\frac{S}{2m+S} \quad (7)$$

Derivation of $P\{M, TI\}$

$$P\{M \mid TI\} = 1 - P\{\bar{M} \mid TI\} = 1 - \frac{m-1}{2m} = \frac{m+1}{2m}$$

$$P\{M, TI\} = P\{M \mid TI\}P\{TI\} = \frac{m+1}{2m}\frac{2m}{2m+S} \quad (8)$$

Derivation of $P\{M, TD\}$ Given that a client tunes in to a data block, the probability that the client misses the `DCI` completely is denoted by $P\{M \mid TD\}$. As in equation (4),

$$P\{M \mid TD\} = \sum_{i=1}^{m} P\{M \mid TD, D_i\}P\{D_i \mid TD\}$$

$$\begin{aligned} P\{M \mid TD, D_i\} = {} & P\{M \mid j > i, TD, D_i\}P\{j > i \mid TD, D_i\} + \\ & P\{M \mid j = i, TD, D_i\}P\{j = i \mid TD, D_i\} + \\ & P\{M \mid j < i, TD, D_i\}P\{j < i \mid TD, D_i\} \end{aligned}$$

where j is the index of the data block to which a client tunes in. Thus,

$$\begin{aligned} P\{j > i \mid TD, D_i\} &= \frac{m-1}{2m} \\ P\{j = i \mid TD, D_i\} &= \frac{1}{m} \\ P\{j < i \mid TD, D_i\} &= \frac{m-1}{2m} \end{aligned}$$

Definitely,

$$\begin{aligned} P\{M \mid j > i, TD, D_i\} &= 1 \\ P\{M \mid j = i, TD, D_i\} &= \frac{B-S}{B} \\ P\{PM \mid j = i, TD, D_i\} &= \frac{S}{B} \end{aligned}$$

$$P\{M \mid TD, D_i\} = \frac{m-1}{2m} + \frac{B-S}{B}\frac{1}{m}$$

$$\begin{aligned} P\{M \mid TD\} &= \sum_{i=1}^{m} P\{M \mid TD, D_i\}P\{D_i\} \\ &= \sum_{i=1}^{m}(\frac{m-1}{2m} + \frac{B-S}{B}\frac{1}{m})\frac{1}{m} \\ &= \frac{m-1}{2m} + \frac{B-S}{mB} \end{aligned}$$

$$P\{M, TD\} = P\{M \mid TD\}P\{TD\} = \frac{mB+B-2S}{2mB}\frac{S}{2m+S} \tag{9}$$

Derivation of $P\{PM, TD\}$

$$\begin{aligned} P\{PM \mid TD, D_i\} &= P\{PM \mid j=i, TD, D_i\}P\{j=i \mid TD, D_i\} \\ &= \tfrac{S}{B}\tfrac{1}{m} = \tfrac{S}{mB} \end{aligned}$$

where j is the index of the data block to which a client tunes in.

$$P\{PM \mid TD\} = \sum_{i=1}^{m} \frac{S}{mB}\frac{1}{m} = \frac{S}{mB}$$

$$P\{PM, TD\} = P\{PM \mid TD\}P\{TD\} = \frac{S}{mB}\frac{S}{2m+S} \tag{10}$$

Table 3 summarizes the probabilities derived above. Note that in the subsequent proofs, we shall repeatedly use the probabilities from table 3.

Events	**Probabilities**
$P\{M, TI\}$	$\frac{m-1}{2m+S}$
$P\{\bar{M}, TD\}$	$\frac{m-1}{2m}\frac{S}{2m+S}$
$P\{M, TI\}$	$\frac{m+1}{2m+S}$
$P\{M, TD\}$	$(\frac{m-1}{2m} + \frac{B-S}{mB})\frac{S}{2m+S}$
$P\{PM\}$	$\frac{S}{mB}\frac{S}{2m+S}$

Table 3: Probabilities of Events

7.2 Analysis of Constant Broadcast Size Strategy

We first present the results of our analysis of the *CBS* strategy. For each *Scenario i*, we present the probability of encountering it ($P\{$*Scenario i*$\}$) and expressions for Tuning Time and Access Time for that scenario, i.e., TT_i and AT_i. Our results are presented as a series of lemmas.

While there are seven cases presented in section 6, there exists a substantial amount of commonality between the cases and they may be aggregated. In the ensuing analysis, we have treated the seven cases through six scenarios, termed Scenario 1, Scenario 2, ..., Scenario 6 respectively. Each of these scenarios map to particular cases described in section 6. Specifically, Scenario 1 (presented in Lemma 3) map to Cases 1 and 4. Scenario 2 (presented in Lemma 4) and Scenario 3 (presented in Lemma 5) map to Cases 2 and 5 respectively, while Scenario 4 (presented in Lemma 6) and Scenario 5 (presented in Lemma 7) map to subcases of Case 7. Scenario 6 (presented in Lemma 8) maps to Cases 3 and 6.

We make two special assumptions in our analysis of the CBS strategy. (a) We assume that items are priority ranked based purely on popularity, i.e. the notion of *Ignore Factor* (IF) is not considered. This was done because it was very difficult to include IF in a mathematically meaningful way in a single analysis. We do include IF in the simulation experiments currently underway. (b) We assume that when a request is made, the requesting client will receive its `DCI` before exiting the cell. Further, for simplicity, we assume that $P\{DCI \in BR_i\}$, i.e. the probability of a requested `DCI` is in the i^{th} broadcast, is independent from $P\{DCI \in BR_{i+1}\}$, thus for a `DCI`, p_{in} remains the same throughout all broadcasts.

Definition 1 *Expected Waiting Time, denoted by $E[T_w]$, is the average amount of time (in number of buckets) that elapses between the submission of a request for a DCI by a client and the start of downloading of that DCI from a subsequent broadcast by the client.*

Lemma 1 *Expected Waiting Time ($E[T_w]$) is expressed as,*

$$E[T_w] = \sum_{j=0}^{\infty} (\frac{1}{2} + j) m(I+B)(p_h(1-p_{h,in})^j p_{h,in} + p_c(1-p_{c,in})^j p_{c,in}) \tag{11}$$

Proof: *Let $T_B = m(I+B)$ be the size of the broadcast in the Constant Broadcast Size strategy. For an arbitrary j,*

$$P\{T_w = \frac{1}{2}T_B + jT_B\} = p_h(1-p_{h,in})^j p_{h,in} + p_c(1-p_{c,in})^j p_{c,in}$$

Therefore,

$$E[T_w] = \sum_{j=0}^{\infty} (\frac{1}{2} + j) m(I+B)(p_h(1-p_{h,in})^j p_{h,in} + p_c(1-p_{c,in})^j p_{c,in})$$

Definition 2 *Pos(i) is a function that returns the ordinal position of an element i in an ordered list A, given that i is an element of A.*

Lemma 2 *Under the Constant Broadcast Size strategy, the probability that a DCI exists in the broadcast that the client initially tunes in to, denoted by P_{in}, is,*

$$P_{in} = \sum_{\{B_h\}} \frac{(\rho_1 + \cdots + \rho_b)!}{\prod_{i=1}^{b} \rho_i!} \prod_{i=1}^{b} p_i^{\rho_i} p_h + \sum_{\{B_c\}} \frac{(\rho_1 + \cdots + \rho_b)!}{\prod_{i=1}^{b} \rho_i!} \prod_{i=1}^{b} p_i^{\rho_i} p_c \tag{12}$$

where $B_h = \{(\rho_1, \rho_2, \cdots, \rho_b) \mid \rho_i \in H, (Pos(\rho_i) \in [1, C_{CBS}])\}$, $B_c = \{(\rho_1, \rho_2, \cdots, \rho_b) \mid \rho_i \in C, (Pos(\rho_i) \in [1, C_{CBS}])\}$, ρ_i is the number of requests to item i. H and C are hot *and* cold *itemsets, respectively.*

Proof:

$$P_{in} = p_{h,in}p_h + p_{c,in}p_c p_{h,in} = \sum_{\{B_h\}} \frac{(\rho_1 + \cdots + \rho_b)!}{\prod_{i=1}^{b} \rho_i!} \prod_{i=1}^{b} p_i^{\rho_i} p_{c,in} + \sum_{\{B_c\}} \frac{(\rho_1 + \cdots + \rho_b)!}{\prod_{i=1}^{b} \rho_i!} \prod_{i=1}^{b} p_i^{\rho_i} \quad (13)$$

All subsequent lemmas in this section deal individual scenarios. Proving these lemmas essentially amounts to deriving the probabilities for the occurrence of the scenarios as well as deriving expressions for AT and TT under these scenarios. However, in the other scenarios, the AT and TT derivation strategies, are very similar to the previous scenario. Therefore, we state but do not explicitly derive AT and TT expressions in the subsequent lemmas.

Lemma 3 *Let* Scenario 1 *denote the scenario where DCI is in the current broadcast and the client didn't miss it. Under* Scenario 1*, the following expressions hold.*

$$P\{Scenario\ 1\} = P_{in}\left(\frac{m-1}{2m+S} + \frac{m-1}{2m}\frac{S}{2m+S}\right) \quad (14)$$

$$TT_1 = 1 + l + S \quad (15)$$

$$E[AT_1|Scenario1] = P_{in}\frac{m-1}{2m+S}\left(\frac{I}{2} + B + I + L_1 + S + \left(\frac{B}{2} + I + L_1 + S\right)\frac{S}{2m}\right) \quad (16)$$

Proof: Lemma 3 is based on the scenario where the `DCI` is in the current broadcast and the client doesn't miss it at all. *Scenario 1* is subdivided into sub-scenarios: *Scenario 1a* where client initially tunes in to an index segment $(\bar{M}, TI)$, and *Scenario 1b* where the client initially tunes in to a data segment $(\bar{M}, TD)$.

$$P\{\text{Scenario 1a}\} = P_{in}P\{\bar{M}, TI\} = P_{in}\frac{m-1}{2m+S}$$

$$P\{\text{Scenario 1b}\} = P\{\bar{M}, TD\} = P_{in}\frac{m-1}{2m}\frac{S}{2m+S}$$

Since Scenario 1a and Scenario 1b are mutually exclusive events,

$$\begin{aligned} P\{\text{Scenario 1}\} &= P\{\text{Scenario 1a}\} + P\{\text{Scenario 1b}\} \\ &= P_{in}\frac{m-1}{2m+S} + P_{in}\frac{m-1}{2m}\frac{S}{2m+S} \\ &= P_{in}\left(\frac{m-1}{2m+S} + \frac{m-1}{2m}\frac{S}{2m+S}\right) \\ TT_1 &= 1 + l + S \end{aligned}$$

The TT and AT are derived as follows,

$$\begin{aligned} TT_1 &= \text{(Client reads the current bucket, gets the address of the next nearest index segment, tunes out) + (Client reads the index tree, tuning in and out along the B-tree, tunes out) + (Client downloads the DCI)} \\ &= 1 + l + S \end{aligned}$$

Under Scenario 1a

$$\begin{aligned} AT_{1a} &= \text{(Access time if a client tunes in to an index segment)} \\ &= \text{(Client tunes in to middle of index segment, waits until next index, reads index, waits until the } \texttt{DCI}\text{, downloads it)} \end{aligned}$$

$$= \frac{I}{2} + B + I + L_1 + S$$

Under Scenario 1b

$$\begin{aligned} AT_{1b} &= \text{(Access time if a client tunes in to a data block)} \\ &\quad \text{(Client tunes in to a data block, waits until the next index, reads} \\ &\quad \text{index, waits until the DCI, downloads it)} \\ &= \frac{B}{2} + I + L_1 + S \end{aligned}$$

Therefore,

$$\begin{aligned} E[AT_1|Scenario1] &= P\{\text{Scenario 1a}\}AT_{1a} + P\{\text{Scenario 1b}\}AT_{1b} \\ &= P_{in}\frac{m-1}{2m+S}(\frac{I}{2} + B + I + L_1 + S) + P_{in}\frac{m-1}{2m}\frac{S}{2m+S}(\frac{B}{2} + I + L_1 + S) \\ &= P_{in}\frac{m-1}{2m+S}(\frac{I}{2} + B + I + L_1 + S + (\frac{B}{2} + I + L_1 + S)\frac{S}{2m}) \end{aligned}$$

All subsequent lemmas will have a similar format to Lemma 3, i.e. they will be concerned with deriving expressions for probabilities of scenarios as well as AT and TT expressions for these scenarios. In the previous lemma we derived expressions for AT and TT. In the subsequent lemmas we omit these derivations.

Lemma 4 *Let* Scenario 2 *denote the scenario where the DCI is in the current broadcast but the client completely missed it. Further, the DCI appears in the next broadcast. Under* Scenario 2*, the following expressions hold.*

$$P\{Scenario\ 2\} = P_{in}^2(\frac{m-1}{2m+S} + (\frac{m-1}{2m} + \frac{B-S}{mB})\frac{S}{2m+S}) \quad (17)$$

$$TT_2 = 1 + 2l + S \quad (18)$$

$$E[AT_2|Scenario2] = P_{in}^2\frac{1}{2m+S}((m-1)(\frac{3I}{2} + B + L_2 + \frac{m(I+B)}{2} + S) + \quad (19)$$

$$(\frac{m-1}{2m} + \frac{B-S}{mB})S(\frac{B}{2} + I + L_2 + \frac{m(I+B)}{2} + S)) \quad (20)$$

Proof: *Scenario 2* is subdivided into two sub-scenarios: *Scenario 2a* where client initially tunes in to an index segment (M, TI),and *Scenario 2b* where client initially tunes in to a data segment (M, TD).

$$\begin{aligned} P\{\text{Scenario 2a}\} &= P_{in}P\{M, TI\}P_{in} = P_{in}^2\frac{m-1}{2m+S} \\ P\{\text{Scenario 2b}\} &= P_{in}P\{M, TD\}P_{in} = P_{in}^2(\frac{m-1}{2m} + \frac{B-S}{mB})\frac{S}{2m+S} \end{aligned}$$

Since Scenario 2a and Scenario 2b are mutually exclusive events,

$$\begin{aligned} P\{\text{Scenario 2}\} &= P\{\text{Scenario 2a}\} + P\{\text{Scenario 2b}\} \\ &= P_{in}^2\frac{m-1}{2m+S} + P_{in}^2(\frac{m-1}{2m} + \frac{B-S}{mB})\frac{S}{2m+S} \\ &= P_{in}^2(\frac{m-1}{2m+S} + (\frac{m-1}{2m} + \frac{B-S}{mB})\frac{S}{2m+S}) \end{aligned}$$

Further,

$$\begin{aligned}
AT_{2a} &= \frac{3I}{2} + B + L_2 + \frac{m(I+B)}{2} + S \\
AT_{2b} &= \frac{B}{2} + I + L_2 + \frac{m(I+B)}{2} + S \\
E[AT_2|Scenario2] &= P\{\text{Scenario 2a}\}AT_{2a} + P\{\text{Scenario 2b}\}AT_{2b} \\
&= P_{in}^2 \frac{m-1}{2m+S}(\frac{3I}{2} + B + L_2 + \frac{m(I+B)}{2} + S) + \\
&\quad P_{in}^2(\frac{m-1}{2m} + \frac{B-S}{mB})\frac{S}{2m+S}(\frac{B}{2} + I + L_2 + \frac{m(I+B)}{2} + S) \\
&= P_{in}^2 \frac{1}{2m+S}((m-1)(\frac{3I}{2} + B + L_2 + \frac{m(I+B)}{2} + S) + \\
&\quad (\frac{m-1}{2m} + \frac{B-S}{mB})S(\frac{B}{2} + I + L_2 + \frac{m(I+B)}{2} + S))
\end{aligned}$$

Lemma 5 *Let* Scenario 3 *denote the scenario where the DCI is in the current broadcast but the client completely missed it. Furthermore, the DCI will not appear in the next broadcast, and thus the client must submit a request to the server. Under* Scenario 3 *the following expressions hold, assuming that the DCI will appear in a broadcast before the client exits the cell.*

$$P\{Scenario\ 3\} = P_{in}(1-P_{in})(\frac{m-1}{2m+S} + (\frac{m-1}{2m} + \frac{B-S}{mB})\frac{S}{2m+S}) \quad (21)$$

$$TT_3 = 1 + \frac{E[W_t]}{m(I+B)}l + S \quad (22)$$

$$AT_3 = m(I+B)(\frac{E[W_t]}{m(I+B)} + 1) + S \quad (23)$$

Proof:

$$P\{\text{Scenario 3}\} = P_{in}P\{M\}(1-P_{in}) = P_{in}(1-P_{in})(\frac{m-1}{2m+S} + (\frac{m-1}{2m} + \frac{B-S}{mB})\frac{S}{2m+S})$$

Lemma 6 *Let* Scenario 4 *denote the scenario where the client partially missed the DCI in the current broadcast, and the DCI reappears in the next broadcast. Under* Scenario 4*, the following expressions hold.*

$$P\{Scenario\ 4\} = P_{in}^2 \frac{S}{mB}\frac{S}{2m+S} \quad (24)$$

$$TT_4 = \frac{S}{2} + l + \frac{S}{2} \quad (25)$$

$$AT_4 = m(I+B) \quad (26)$$

Proof:

$$P\{\text{Scenario 4}\} = P_{in}P\{PM\}P_{in} = P_{in}^2 \frac{S}{mB}\frac{S}{2m+S}$$

Lemma 7 *Let* Scenario 5 *denote the scenario where the client partially missed the DCI in the current broadcast, but the DCI does not appear in the next broadcast (it appears before the client exits the cell). Under* Scenario 5*, the following expressions hold.*

$$P\{Scenario\ 5\} = P_{in}\frac{S}{mB}\frac{S}{2m+S}(1-P_{in}) \quad (27)$$

$$TT_5 = \frac{S}{2} + l + \frac{E[W_t]}{m(I+B)}l + S \quad (28)$$

$$AT_5 = m(I+B)(\frac{E[W_t]}{m(I+B)} + 1) + S \quad (29)$$

Proof:

$$P\{\text{Scenario 5}\} = P_{in}P\{PM\}(1-P_{in}) = P_{in}\frac{S}{mB}\frac{S}{2m+S}(1-P_{in})$$

Lemma 8 *Let* Scenario *6 denote the scenario where the DCI is not in the current broadcast but appears before the client exits the cell. Under* Scenario 6*, the following expressions hold.*

$$P\{Scenario\ 6\} = (1-P_{in}) \quad (30)$$

$$TT_6 = 1 + l + \frac{E[W_t]}{m(I+B)}l + S \quad (31)$$

$$AT_6 = m(I+B)(\frac{E[W_t]}{m(I+B)} + 1) + S \quad (32)$$

7.3 Analysis of Variable Broadcast Size Strategy

In this section, we present our analysis of the *variable broadcast size* strategy in a series of scenarios. As in the CBS analysis presented in the previous section, these scenarios, described through a series of lemmas, are aggregations of the cases presented in section 6. Specifically, Scenario 7 (Lemma 13) refers to Case 1, Scenario 8 (Lemma 14) to Case 4, Scenario 9 (Lemma 15) to Case 2, Scenario 10 (Lemma 16) to Case 5, Scenario 11 (Lemma 17) to Case 7. Case 3 and Case 6 are aggregated in Scenario 12 (Lemma 18). Note that the $E[TT]$ and $E[AT]$ are calculated by multiplying the probability terms in *Scenarios 7* through *11* by $P\{DCI \in BR\}$. Further note that in the abovementioned lemmas we do not derive the AT and TT expressions, as in the CBS case. Recall that we provided a sample derivation of AT and TT in Lemma 3.

We remind the reader at this point that the VBS strategy server protocol will always place requested `DCI`s in the subsequent broadcast. Therefore, if a client's `DCI` is not in the current broadcast, she is guaranteed to find it in the next broadcast.

Lemma 9 *Let L_1 denote the expected distance from end of index to the DCI, given that the DCI is not missed. L_1 is expressed as,*

$$L_1 = E\{Distance\ to\ DCI|\bar{M}\} = \frac{B(m-1)}{4m} + \frac{(I+B)(m-1)(m-2)}{6m} \quad (33)$$

Proof: Expected distance from end of index to the *DCI*, given that the `DCI` has not been missed is,

$$\begin{aligned} L_1 &= E\{\text{Distance to DCI}|\bar{M}\} = \sum_{i=2}^{m}\sum_{j=i}^{m} E\{\text{Distance to DCI}|TI_i, D_j\}P\{TI_i\}P\{D_j\} \\ &= \sum_{i=2}^{m}\sum_{j=i}^{m}(\frac{B}{2} + (j-i)(I+B))\frac{1}{m^2} \\ &= \frac{1}{m^2}\sum_{i=2}^{m}(\frac{(m-i+1)B}{2} + \frac{(m-i+1)(m-i)(I+B)}{2}) \\ &= \frac{B(m-1)}{4m} + \frac{(I+B)(m-1)(m-2)}{6m} \end{aligned}$$

Lemma 10 *Let L_2 denote the expected distance from end of index to the next broadcast, given that the DCI is missed. L_2 is expressed as,*

$$L_2 = E\{\textit{Distance to next broadcast}|M\} = \frac{1}{2m}((m-1)B + (I+B)\frac{2m^2-6m+3}{6}) \tag{34}$$

Proof: The expected distance from end of index to the next broadcast, given that the `DCI` has been missed is,

$$\begin{aligned} L_2 &= E\{\text{Distance to next broadcast}|M\} \\ &= \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} E\{\text{Distance to next broadcast}|TI_j, D_i\}P\{TI_j\}P\{D_i\} \\ &= \frac{1}{m^2}\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}(B + (I+B)(m-j)) \\ &= \frac{1}{m^2}\sum_{i=1}^{m-1}((m-i)B + (I+B)\frac{(m-i)(m-i-1)}{2}) \\ &= \frac{1}{2m}((m-1)B + (I+B)\frac{2m^2-6m+3}{6}) \end{aligned}$$

Lemma 11 *Let $P\{DCI \in BR\}$ denote the probability that the DCI is in the current broadcast under VBS. $P\{DCI \in BR\}$ is given by,*

$$P\{DCI \in BR\} = (1-(1-\tfrac{(1-p_0)p_h}{bh})^N)p_h + (1-(1-\tfrac{(1-p_0)p_c}{b(1-h)})^N)p_c \tag{35}$$

Proof:

$$P\{DCI \in BR\} = p_{h,in}p_h + p_{c,in}p_c = (1-(1-\frac{(1-p_0)p_h}{bh})^N)p_h + (1-(1-\frac{(1-p_0)p_c}{b(1-h)})^N)p_c$$

Lemma 12 *Let BS denote the size of the broadcast under VBS. The expectation of BS is defined as*

$$E\{BL\} = \sum_{j=0}^{d} jS \times P\{BL = jS\} \tag{36}$$

where,

$$\begin{aligned} P\{BL = jS\} &= P\{j \text{ different } DCI \text{ from } N \text{ client}\} \\ &= \sum_{i=0}^{j} \left(\begin{array}{c} bh \\ i \end{array} \right) \left(\begin{array}{c} b(1-h) \\ j-i \end{array} \right) \sum_{A} \left(\frac{N!}{\prod \rho_q!} (\frac{(1-p_0)p_h}{bh})^{l+\sum_{q=1}^{i}\rho_q} (\frac{(1-p_0)p_c}{b(1-h)})^{j-i+\sum_{q=i+1}^{j}\rho_q} \right) \\ \text{and, } A &= (\rho_1, \cdots, \rho_j), \forall \rho_q \geq 0, q \in [1,j], \sum_{q=1}^{j} \rho_q = N - j, \rho_i = \text{number of requests to item } i \end{aligned}$$

Lemma 13 *Let* Scenario 7 *denote the scenario where the client tunes in to an index bucket but does not miss the DCI in the current broadcast. Under* Scenario 7*, the following expressions hold.*

$$P\{Scenario\ 7\} = \frac{m-1}{2m+S} \tag{37}$$

$$TT_7 = 1 + l + S, \tag{38}$$

$$AT_7 = \frac{I}{2} + B + I + L_1 + S \tag{39}$$

Proof:

$$P\{\text{Scenario 7}\} = P\{\bar{M}, TI = P\{\bar{M} \mid TI\}P\{TI\} = \frac{m-1}{2m+S}$$

Lemma 14 *Let* Scenario 8 *denote the scenario where the client tunes in to a data bucket before the data block containing the DCI in the current broadcast. Under* Scenario 8*, the following expressions hold.*

$$P\{Scenario\ 8\} = \frac{m-1}{2m}\frac{S}{2m+S} \tag{40}$$

$$TT_8 = 1 + l + S, \tag{41}$$

$$AT_8 = \frac{B}{2} + I + L_1 + S \tag{42}$$

Proof:

$$P\{\text{Scenario 8}\} = P\{\bar{M}, TD\} = P\{\bar{M} \mid TD\} = \frac{m-1}{2m}\frac{S}{2m+S}P\{TD\}$$

Lemma 15 *Let* Scenario 9 *denote the scenario where the client tunes in to an index bucket but missed the DCI in the current broadcast. Under* Scenario 9 *the following expressions hold.*

$$P\{Scenario\ 9\} = \frac{m+1}{2m+S} \tag{43}$$

$$TT_9 = 1 + 2l + S \tag{44}$$

$$AT_9 = \frac{I}{2} + B + I + L_2 + \frac{m(I+B)}{2} + S \tag{45}$$

Proof:

$$P\{\text{Scenario 9}\} \quad = \quad P\{M, TI\} = P\{M \mid TI\}P\{TI\} = \frac{m+1}{2m+S}$$

Lemma 16 *Let* Scenario 10 *denote the scenario where the client tunes in to a data bucket, but missed the DCI in the current broadcast. Under* Scenario 10*, the following expressions hold.*

$$P\{Scenario\ 10\} = \frac{mB + B - 2S}{2mB}\frac{S}{2m+S} \tag{46}$$

$$TT_{10} = 1 + 2l + S \tag{47}$$

$$AT_{10} = \frac{B}{2} + I + L_2 + \frac{m(I+B)}{2} + S \tag{48}$$

Proof:

$$P\{\text{Scenario 10}\} \quad = \quad P\{M, TD\} = P\{M \mid TD\}P\{TD\} = \frac{mB + B - 2S}{2mB}\frac{S}{2m+S}$$

Lemma 17 *Let* Scenario 11 *denote the scenario where the client tunes in to a data bucket in the middle of the DCI in the current broadcast. Under* Scenario 11*, the following expressions hold.*

$$P\{Scenario\ 11\} = \frac{S}{mB}\frac{S}{2m+S} \tag{49}$$

$$TT_{11} = \frac{S}{2} + l + \frac{S}{2} \tag{50}$$

$$AT_{11} = m(I+B) + \frac{S}{4} \tag{51}$$

Proof:

$$P\{\text{Scenario 11}\} \quad = \quad P\{PM, TD\} = P\{PM \mid TD\}P\{TD\} = \frac{S}{mB}\frac{S}{2m+S}$$

Lemma 18 *Let* Scenario 12 *denote the scenario where the client tunes in to either a data bucket or an index bucket. Given that DCI does not exist in the current broadcast, under* Scenario 12 *the following expressions hold.*

$$P\{Scenario\ 12\} = 1 - ((1 - (1 - \frac{(1-p_0)p_h}{bh})^N)p_h + (1 - (1 - \frac{(1-p_0)p_c}{b(1-h)})^N)p_c) \tag{52}$$

$$TT_{12} = 1 + 2l + S \tag{53}$$

$$AT_{12} = m(I+B) + S \tag{54}$$

Proof:

$$P\{\text{Scenario 12}\} \quad = \quad 1 - P\{DCI \in BR\} \text{ and substitute the result of Lemma 11}$$

8 Discussion

Based on our analyses of both the *constant broadcast size* and the *variable broadcast size* strategies, we plotted the tuning time (TT) and access time (AT) curves for various parameters. In this section, we illustrate our strategies in a small scale example, primarily due to the computational costs involved. Although the database used in the examples consists of 200 buckets, we believe it is large enough to capture the essence of the strategies. Our parameters are: $b = 10$, $S = 20$, $n = 1$, $p_h = 0.8$, $h = 0.2$, $c = 0.8$, $m = 4$ for VBS, and $C_{CBS} = 4$ for CBS, initially.

8.1 Discussion on AT and TT curves

Here, in figures 5A and B, we illustrate the changes in the tuning and access times of the two strategies that we proposed. In both figures, we vary the number of clients in the cell, keeping other parameters constant.

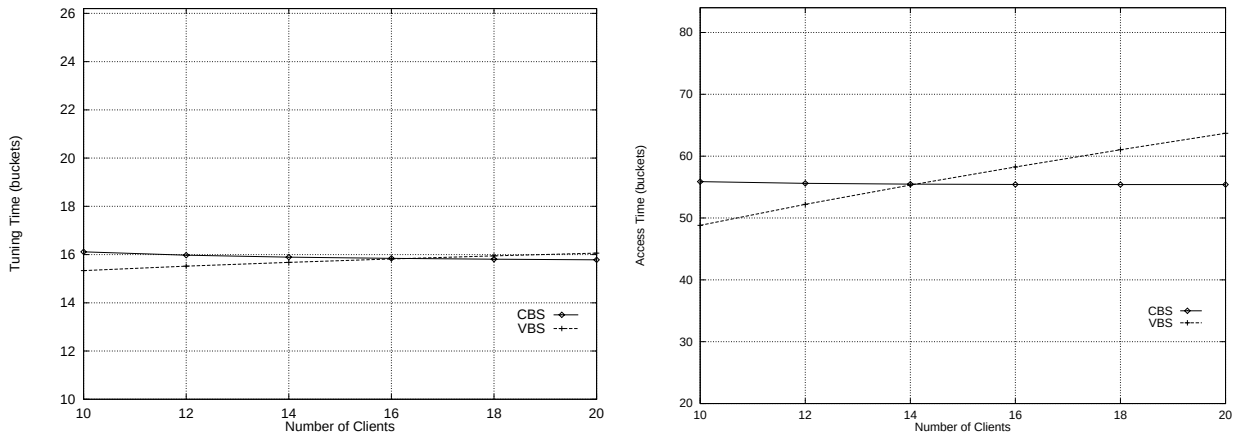

Figure 5: Experimental Results: [A] TT for VBS and CBS; [B] AT for VBS and CBS

Tuning Time (TT) curves: We first turn our attention to the *variable broadcast size* (VBS) strategy. VBS strategy includes every requested item in the broadcast. As number of clients increases, the set of requested items grows as well. As a result, the size of the index segments in VBS goes up. Clearly, TT is proportional to the size of the index. Thus in VBS, TT increases as the number of clients increases. This explains the upward trend of the TT curve for VBS in figure 5A.

Another interesting feature of the TT curve for VBS is that the slope decreases along the curve. It is easily seen that the slope of this curve is proportional to the rate of growth of the requested item set, i.e., the faster the growth of the demand set, the faster the growth of the tuning time. When the cell is sparsely populated, each additional user has a high probability of asking for new items. However, under heavier population loads, new arrivals tend to ask for items already in the request set (e.g. *hot* items) which slows down the growth of this curve. We call such requests *redundant requests*.

We now turn our attention to the second curve in figure 5A; the curve for the *constant broadcast size* strategy (*CBS*). In the *CBS* strategy, index size is limited because the broadcast size is limited. As the number of clients increases, the number of redundant requests, particularly for *hot* items, increases. Therefore, the probability that a requested item is in the broadcast (p_b) increases, lowering TT for larger client sets. For example, for $N = 10$, p_b is 0.830155 whereas for $N = 12$, it increases to 0.840821. Here, in the case where $N = 12$, clients have a higher probability of downloading the `DCI` in the current broadcast than in the case where $N = 10$. Clients who download the `DCI` in the current broadcast need not submit a request and tune to the index of successive broadcasts, thus saving tuning time. As the number of clients increases, the number of non-redundant requests also increases, which slows the decrease in TT as illustrated in figure 5A.

Having discussed the individual TT curves, we now focus on explaining the relative performance of the two strategies. With the current set of parameters, *VBS* performs slightly better than *CBS* for low to moderate loads in the system. However, as the load, i.e., the number of clients, increases, *CBS* performs better. Intuitively, this makes sense. When the number of clients grows very large, *VBS* will include all their requests, which will include many *cold items* as well as hot items. This causes indices to grow to a point where TT becomes very large. By virtue of limiting the size of the index, *CBS* does not suffer from this problem.

Access Time (*AT*) curves: In the *variable broadcast size* strategy, AT responds to an increasing number of clients in the same way as TT. It increases as the number of clients increases, but the increase slows as redundancy in the request set increases. This can be seen readily in figure 5B.

Initially in the *constant broadcast size* strategy, the server does not receive enough requests to fill the broadcast, causing dead air in the broadcast. Clients are forced to submit requests for their `DCI`s and wait for the next broadcast. This contributes to the relatively high initial AT. As the broadcast starts to fill up, the effect of the increasing number of clients on AT decreases. *Hot* items are likely to be requested by many clients, and are therefore likely to be in the current broadcast when a client tunes in. figure 5B shows the curve for the *CBS*.

The two curves in figure 5B behave differently at different load levels. The *VBS* curve performs better than the *VBS* when the system load is low. For moderate to high loads in the system, *VBS* AT increases sharply while *CBS* AT remains fairly constant. In this scenario, *CBS* provides a lower access times for clients than *VBS*.

Overall, it can be deduced that when the system load is low, *VBS* outperforms *CBS* for both TT and AT. For moderate loads, there is a trade-off between the two strategies: *VBS* performs better for TT and *CBS* performs better in terms of AT. For high loads in the system, *CBS* dominates *VBS*, providing lower TT and AT.

8.2 Changes in *TT* and *AT* for *VBS* as the number of indices in broadcast varies

The number of index segments in the broadcast (m) is the most important parameter for the *VBS* because the broadcast size is not limited. Increasing m provides a larger number of opportunities for clients to read the index, which tends to lower tuning times, but increases the overall size of the broadcast, which increases access times. Therefore, we plotted the TT and AT curves for comparing the effect of various m values. Figures 6A and B illustrate the changes in TT and AT curves, respectively, as m changes.

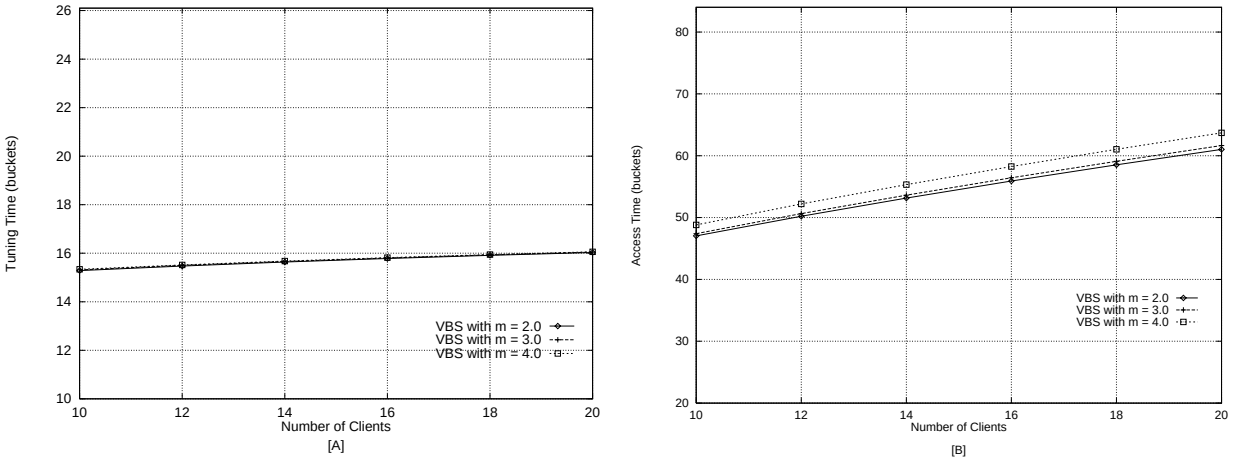

Figure 6: Experimental Results: [A] TT for VBS with various m values; [B] AT for VBS with various m values

***TT* curves:** Figure 6A shows that the change m does not have a significant effect on the TT. Since the data in the broadcast remains constant at a given client size, the index size remains constant as well. Slight differences in the curves are mainly due to round-up errors, and TT curves converge to a single curve.

***AT* curves:** At a given number of clients, even if the m value is different, the amount of data in the broadcast is the same. However, for different values of m, the broadcast overall size changes since we change the number of index segments. Therefore, if m is increased, the size of the broadcast grows, thus increasing the AT.

8.3 Changes in *TT* and *AT* for *CBS* as broadcast capacity varies

The crucial parameter in the *Constant Broadcast Size* strategy is the *capacity of the broadcast* denoted by C_{CBS}. Limiting the capacity of the broadcast and favoring more popular items for inclusion reduces the access and tuning times for *hot* items, at the expense of *colder* items. The basic premise of the strategy is that with a "good" value of C_{CBS}, both access and tuning times may be reduced. In figures 7A and B, we illustrate the effects of changing the broadcast capacity (C_{CBS}) on TT and AT for the *constant broadcast size* strategy. Note that in the curves the parameter *Capacity* represents C_{CBS}.

***TT* curves:** Here, we turn our attention to figure 7A, and observe the effects of changing the broadcast capacity on TT.

For $C_{CBS} = 2$, p_b increases as the number of clients in the cell increases. However, a broadcast capacity of 2 `DCIs` is extremely small. In this scenario, *hot* items are continually included in the broadcast, causing the exclusion of *cold* items. Clients requesting *cold* items are forced to tune in to each successive broadcast until they exit the cell or the request becomes invalid. This results in a considerable increase in TT.

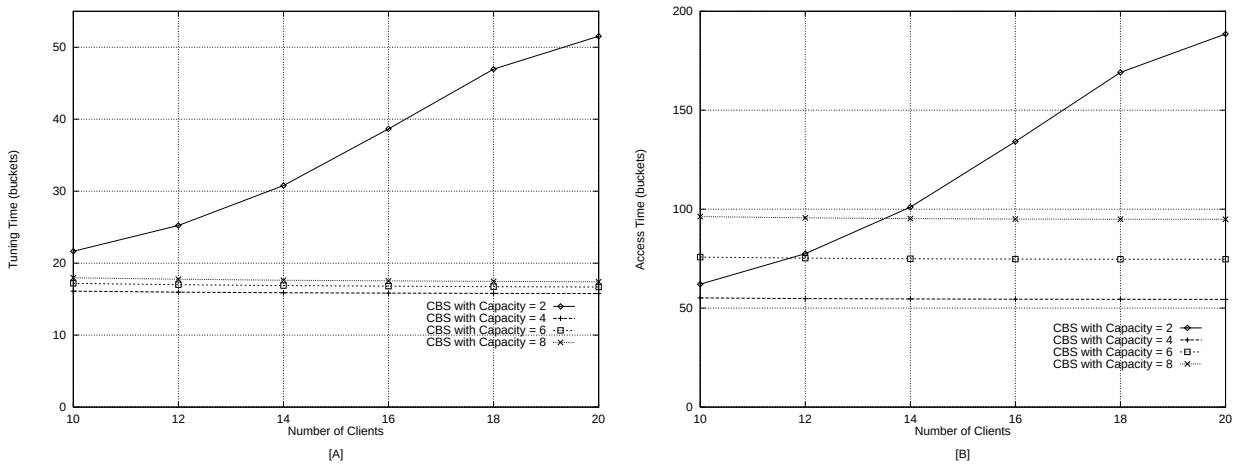

Figure 7: Experimental Results: [A] TT for CBS with various C_{CBS} values; [B] AT for CBS with various C_{CBS} values

For $C_{CBS} = 4$, p_b is higher than in the case where $C_{CBS} = 2$. Both *hot* and *cold* items have a higher chance to be included in the broadcast as the capacity of the broadcast grows. Therefore, clients are less starved, and the Expected Waiting Time is lower than in the case where $C_{CBS} = 2$. The number of redundant requests, particularly for *hot*items, increases as the number of clients increases, causing a slight decrease in the TT curve.

The curve for $C_{CBS} = 8$ is similar to the curve for $C_{CBS} = 4$. Its location above the $C_{CBS} = 4$ curve may seem counterintuitive at first; we would expect the tuning time to decrease even more by increasing the size of the broadcast. However, this is an excellent example of unnecessarily high broadcast size. The server does not receive enough requests to fill the broadcast capacity, and broadcasts dead air.

***AT* curves:** Expected Waiting Time is the most significant part of the AT because clients who have missed their `DCI`s, or whose `DCI`s are not included in the current broadcast, must wait, at a minimum, until the next broadcast to retrieve their DCIs. This waiting time is not considered part of TT. The curves for AT in figure 7B are similar to those shown for TT in figure 7A.

It can be concluded that, in order for *CBS* to perform at its best, the broadcast size parameter should be adjusted to the load in the system. This result emphasizes the importance of adaptive capability of broadcasting strategies.

9 Conclusion

In this paper, we looked at the problem of data retrieval by mobile units from wireless broadcasts. While the problem if data organization in broadcasts has been looked at before, the specific content of the broadcast has been unexplored. Specifically, in the presence of mobile clients with differing demands, the determination of the broadcast content is an important problem. This paper presented a number of protocols to achieve such a determination. In conjunction, we also stipulated a number of client retrieval algorithms to aid efficient retrieval from broadcasts.

We have used a general architecture of a mobile platform with a number of mobile units, *fixed hosts* and *base stations*. Mobile units (*clients*) move around freely in a *cell* served by base stations (*servers*). Clients and servers communicate through wireless channels. We assumed that the wireless channel consists of both an uplink for moving data from client to server and a downlink for moving data from server to client.

Since in wireless networks, the active life of a mobile unit's power source (battery) is an important constraint on applications, we have considered, in addition to the conventional *response time* metric, a metric called *tuning time*. This latter metric captures the actual duration of time clients must be active, thereby providing a measure of energy expenditure.

The problem addressed in this paper may be captured by the following question: *given that users are highly mobile in their mobility domain, what are good strategies that the server can use to decide on what to broadcast?* We also looked at the question of retrieval strategies: *given that good broadcast strategies are found, what are good retrieval algorithms by which users can retrieve/download data from broadcast, with a minimum of energy expenditure?*

Most of the earlier work in this area is concerned with *access methods*, i.e., how to organize the broadcast data. Given the composition of a broadcast, there has been lot of nice work in organizing data and index on air. Our concern is to organize the broadcast data as well as determine and dynamically alter its content based on the demand pattern of the clients in the cell.

In this paper, we presented adaptive protocols for broadcast content determination and information retrieval. Periodicity has been regarded as an important parameter in *broadcast-and-retrieve* environments. We thus considered both periodic (*constant broadcast size* (CBS) strategy) and aperiodic (*variable broadcast* (VBS) size) broadcast strategies.

In the *constant broadcast size* strategy, we fixed the length of a broadcast and added or dropped items from the broadcast based on their popularity. We also introduced a metric called *Ignore Factor* to weigh items on how long requests for them have been ignored so that even if they are not popular enough to be included in the broadcast, the clients requesting it are not starved.

In the *variable broadcast size* strategy, the broadcast size changes according to the number of items requested during the previous broadcast period. Clients are guaranteed to receive items that they requested. Since this strategy can potentially include all the items in the database, we introduced the concept *Residence Latency* which will drop items from the broadcast based on the expected time that a client requesting an item will stay in the cell.

We have performed an approximate yet thorough performance analysis by analytical methods. Our analysis of these strategies suggests that the aperiodic (*VBS*) strategy outperforms the periodic (*CBS*) strategy at low system loads. For moderate loads, VBS provides lower tuning times, while *CBS* provides lower access times. At high loads, *CBS* outperforms *VBS* overall, providing lower tuning and access times.

Also, we have seen that the number of index segments in the broadcast is the most important parameter for the *VBS*. This parameter has the greatest effect on the access time since broadcast size in this strategy is not limited.

We have also varied broadcast capacity in the constant broadcast size strategy, and observed that limiting the capacity of the broadcast and favoring more popular items for inclusion reduces the access and tuning times for *hot* items, at the expense of *colder* items.

References

[1] V. C. Leung A. D. Malyan, L. J. Ng and R. W. Donaldson. Network architecture and signaling for wireless personal communications. *IEEE Journal on Selected Areas in Communications*, pages 830–841, August 1993.

[2] R. Alonso and H. Korth. Database issues in nomadic computing. In *Proceedings of the 1993 ACM-SIGMOD*, 1993.

[3] B. R. Badrinath and T. Imielinski. Replication and mobility. In *Second Workshop on the Management of Replicated Data*, pages 9–12, November 1992.

[4] D. Barbara and T. Imielinski. Sleepers and workaholics: Caching strategies in mobile environments. In R. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD*, pages 1–12, May 1994.

[5] D. Cox D. Lam, J. Jannink and J. Wisdom. Modeling location management in personal communication services. Technical report, Stanford University, October 1995.

[6] A. Datta. Research Issues in Databases for Active Rapidly Changing data Systems (ARCS). *ACM SIGMOD RECORD*, 23(3):8–13, September 1994.

[7] M. H. Dunham and A. Helal. Mobile computing and databases: Anything new? In *SIGMOD Record*. Association for Computing Machinery, December 1995.

[8] H. G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computers*, 27(4), April 1994.

[9] Y. Huang, P. Sistla, and O. Wolfson. Data replication for mobile computers. *SIGMOD Record*, 23(2):13–24, 1994.

[10] T. Imielinski and B. R. Badrinath. Querying in highly mobile distributed environments. In *Proceedings of the 18th International Conference on Very Large Databases*, pages 41–52, August 1992.

[11] T. Imielinski and B. R. Badrinath. Mobile wireless computing. *Communications of the ACM*, pages 19–28, October 1994.

[12] N. H. Vaidya P. Krishna and D. K. Pradhan. Location management in distributed mobile environments. In *Proceedings of the Conference on Parallel and Distrbuted Information Systems*, pages 81–88, September 1994.

[13] M. Satyanarayanan. Accessing information on demand at any location. mobile information access. *IEEE Personal Communications*, 3(1):26–33, 1996.

[14] N. Shivakumar and J. Wisdom. User profile replication for faster location lookup in mobile environments. Technical report, Stanford University, 1994.

[15] S. Vishwanath T. Imielinski and B. R. Badrinath. Data on air: Organization and access. submitted for publication, available at http://athos.rutgers.edu/ badri/dataman/bcast.html.

[16] S. Viswanathan T. Imielinski and B. R. Badrinath. Indexing on air. In *Proceedings of the 1994 ACM SIGMOD*, 1994.

[17] S. Viswanathan T. Imielinski and B. R. Badrinath. Power efficient filtering of data on air. In *Proceedings of the 4th International Conference on Data Management*, March 1994.

[18] U. Madhow V. Anantharam, M. L. Hong and V. K. Wei. Optimization of a database hierarchy for mobility tracking in a personal communications network. *Performance Evaluation*, pages 287–300, May 1994.

[19] Nitin H. Vaidya and Sohail Hameed. Data broadcast in asymmetric environments. In *First International Workshop on Satellite-based Information Services (WOSBIS)*, nov 1996.

[20] Nitin H. Vaidya and Sohail Hameed. Scheduling data broadcast in asymmetric communication environments. Technical Report 96-022, Computer Science Department, Texas A&M University, College Station, November 1996.

[21] O. Wolfson and S. Jajodia. Distributed algorithms for dynamic replication of data. In *Proceedings of the Symposium on Principles of Database Systems*, pages 149–163, 1992.