Cost-Aware Wireless Data Broadcasting

C. K. Liaskos, *Member, IEEE,* S. G. Petridou, *Member, IEEE,* G. I. Papadimitriou, *Senior Member, IEEE*

***Abstract*—Research on push systems has naturally focused on improving the client serving time. However, in many cases the determinant factor for choosing between push and pull logic is the low central infrastructure cost. Adaptive push systems take this condition to the next step, requiring the central calculation of optimal broadcast schedules every few seconds or minutes, performed by relatively inexpensive server hardware. Aiming at introducing cost-aware wireless broadcasting systems, the Fast Optimization - Memory Conserving (FOMC) method is presented. The goal of the proposed method is to reduce the required computational power and memory of the central server, to the point that it can be implemented with mainstream hardware, and be incorporated to minute-scale adaptive systems. In order to demonstrate the importance of taking cost into account, FOMC is compared with the classical and influential Broadcast Disks method. While FOMC succeeds in requiring reasonable computational power and minimal memory, the Broadcast Disks method is rendered merely unrealizable in practice. Finally, in a first effort to minimize the scanning time for optimal broadcasting parameters, their relation with the total number broadcasted data items is being studied through a specially adapted multivariate illustration technique, and it is observed to be non-linear.**

***Index Terms*—Analysis, data broadcasting, cost, wireless push system.**

I. Introduction

RECENT years have witnessed the widespreading use of wireless push-based systems. Simple in architecture and implementation, lightweight and energy efficient - especially from the client's point of view and both hardware and software wise - the push-based approach has been adopted for use in a variety of information dissemination applications and has been incorporated in almost every single mobile telecommunications device [1–5].

Popular uses include airport and hospital informative systems, instant messaging services, and multimedia on-demand over the internet or cellular networks [6, 7]. Among push-based applications utilizing broadcasting are XM Traffic & Weather [8] and Microsoft's Smart Personal Objects Technology (SPOT) initiative [9]. Consequently, the on-growing interest of the telecommunications industry has spurred the research on the resource efficiency and performance optimization of these systems.

In contrast with the pull architecture, where the clients obtain information by posing specific queries to a server and receiving corresponding replies, push-based systems function by constantly transmitting information according to a common program that best fits the needs of the clients. In classic push schemes this schedule is fixed and static. In modern, adaptive push systems though, the schedule is periodically renewed and the clients contribute to its production through a simplified voting scheme.

C.K. Liaskos, S.G. Petridou and G.I. Papadimitriou are with the Department of Informatics, Aristotle University of Thessaloniki, 54124, Thessaloniki, Greece (e-mails: {cliaskos, spetrido, gp}@csd.auth.gr)

The application scopes of the push and pull schemes are not mutually exclusive. Essentially, any information transaction can be performed more efficiently by the classic pull scheme, which dominates the modern application pool. However, the push scheme has a very unique advantage: minimal realization cost with respect to scalability. Trivial equipment is required to broadcast data, and far more lightweight are the receiving client devices. Should the number of clients increase by any factor, none of them would suffer quality degradation or denial of service, nor would there be any need to upgrade the broadcast server. This would obviously not be the case for the corresponding pull aspect, where empowering the server to handle millions of queries would be imperative.

Evidently, the cost of a broadcast server must remain logically minimal for the push scheme adoption to make sense. Should the cost become exquisite, a shift to the most efficient (serving time-wise) pull logic is inevitable. The main burden of a broadcast server is the definition of the optimal broadcast schedule. In the classic, static push systems where the scheduled needed to be calculated just once, the computational burden of this task was negligible. In the case of adaptive push systems though, the server must periodically produce an updated version of the schedule by utilizing the client feedback provided through the voting mechanism. A cost related problem obviously arises when e.g. the formation of the schedule on the lightweight push server takes more time than the renewal period.

Interestingly, while many outstanding scheduling algorithms have been proposed, none has been tested-to the best of the authors's knowledge-for compliance with the aforementioned cost-related criteria.

A. Related Work

Research on the field of broadcast systems can be roughly split into two main categories. The mathematical foundation and analysis attempts of the optimal broadcast problem in general and algorithmic approaches that tend to provide simplified, algorithmically applicable solutions of the general problem and/or examine other advanced aspects. In the definition of the data broadcast problem, each to-be-broadcasted piece of information is assigned a generic "broadcast weight". Then, the problem is to find a schedule over an infinite-time horizon so that the mean client waiting time as well as the mean broadcast weight are minimized. In this context, the problem is proved to be NP-hard [10–12]. [10] discusses the generalized maintenance problem (a known NP-hard problem), and proves the broadcasting of equally-sized items to be a subcase of it. A lower bound for the client's mean waiting time is also

provided. [13] proves the existence of a possible solution for teletext systems, and also defines a lower bound for the client's mean waiting time in this case.

Another research branch ignores broadcast weights. In this case, uniform and non-uniform item sizes are discussed. [14] and [15] present the lower bound for the client mean waiting time in the case of both uniform and nonuniform sizes. A scheduling algorithm is also defined, which even today achieves the minimum client waiting times. The lower bound in the nonuniform case is not tight though, and this case is furtherly analyzed in [12]. [16] introduces a "reservation" system which favors client waiting time in the case of big sized items.

Retaining the no-broadcast weights consideration, several algorithmic approaches have been proposed to simplify the data broadcasting problem, the most influential of all being the Broadcast Disks model [17]. A great deal of work has been done based on this model, studying data prefetching, caching [18, 19] and indexing [20, 21], hybrid data broadcasting [22, 23], as well as scheduling strategies [24, 25] and noise interference [17].

Another branch studies data broadcasting from a higher level, transaction aspect (e.g. queries that must be answered in a limited amount of time [26], among other limitations) and was introduced in [27]. This approach has since been extended with fail-over tactics [28] and techniques [29, 30] that increase the concurrency of transactions. [31] and [32] consider the scenario of processing queries using information simultaneously received from multiple channels, while [33] present two algorithms for supporting location-based queries in wireless data broadcasting systems.

Finally, other recent studies deal with the case of correlated client queries, e.g. complex queries requesting multiple single data items [34–36]. [37] deals with the case of requesting successive single items, while [38] and [39], examine the case of random access order of single items. [40] and [41] present algorithms that achieve the lower bounds of average access time in the case of broadcasting a pair of files.

The authors have so far contributed in both analytical and algorithmic approaches. In [42] and [43] an effort to combine clustering techniques with the broadcast disks model was made, with satisfactory results. However, this approach lacked the means of defining the optimal parameters required to construct the broadcast schedule. Thus, in [44] an analytical approach was presented that provided the tools of defining the optimal scheduling parameters in advance, as well as estimating the mean client waiting time in advance with very good accuracy, without relying on time consuming simulations. The analysis was furtherly refined in [45]. Moreover, research on adaptive push systems has been carried out in [7, 43, 46–48].

B. Contribution

Purpose of the present work is to point out the potential difficulty in realizability of cost-unaware broadcast scheduling schemes, in the case of adaptive wireless push systems. The authors aim at motivating cost evaluation research on this field, which is traditionally overlooked.

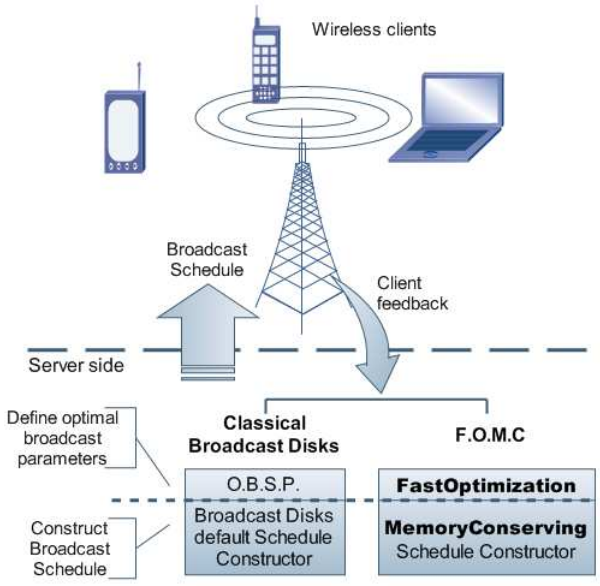

Fig. 1. Network layout and operation overview.

As a most influential scheduling algorithm, the Broadcast Disks method is examined from a computational and memory requirements aspect (the main parameters influencing overall cost), and is found to require resources that render it only conditionally realizable.

Additionally, a new, cost-efficient version of the method, the FastOptimization-MemoryConserving (FOMC) algorithm is presented, which performs acceptably even on mainstream computer hardware.

In adaptive push systems the scheduling procedure comprises of two parts, as shown in Fig. 1:

- Optimization of broadcast parameters based on the clients' feedback (e.g. the optimal broadcast frequency of each data item based on its popularity).
- Construction of the broadcast schedule (i.e. proper data serialization).

The optimization procedure of both compared schemes relies on ranging the broadcast parameters in predefined value sets and projecting the mean client serving time for all possible parameter combinations. (Before [44, 45] optimization for the Broadcast Disks relied solely on time and resource consuming simulations for each combination). A way of limiting required computational resources would thus be the study of the relation connecting the optimal broadcast parameters, in order to minimize the value sets' sizes. Thus, as a minor contribution toward this direction, the linearity of these relations is examined.

C. Overview of Contents

In Section II the network architecture and operation as well as the Broadcast Disks Method and its mathematical analysis are described in more detail. In Section III the aforementioned FastOptimization-MemoryConserving schedule constructor algorithm is presented. Section IV refers to the comparison of the two schemes. Section V provides a brief overview of the covered client cases and presents the specially adapted

TABLE I
BASIC SYMBOLS' NOTATION

Symbol	Description
page	Alias for data item
$dbsize$	Total number of available pages in the server's database, $l = 1, \ldots, dbsize$
$range$	Portion of $dbsize$ pages accessed by a client
k	Number of $Regions$
$Regions = \{r_1, \ldots, r_k\}$	Equally sized groups of pages derived from the subset $range$
$regionsize$	Number of pages contained in a single region r_j, $j = 1, \ldots, k$
NoD	The number of disks
W_i	The i^{th} disk's workload
U_i	The i^{th} disk's angular velocity
p_i	The workload balancing parameter of the i^{th} disk
$\overline{D}$	The client's mean waiting time
L	The size of the broadcast schedule

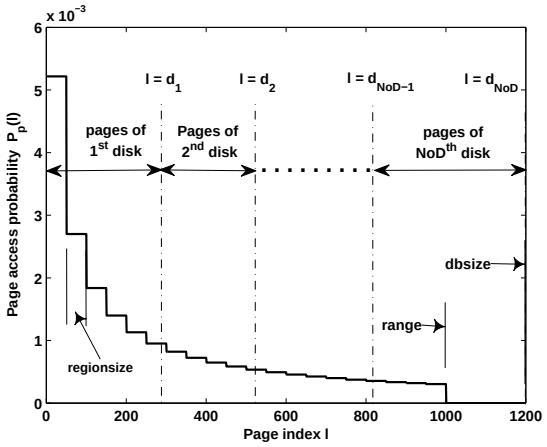

Fig. 2. The client's probability distribution function

multivariate illustration technique used to depict the relation between optimal scheduling parameters. Finally, conclusions are given in Section VI.

II. SYSTEM BACKGROUND

The proposed FOMC scheme is a cost-aware version of the classical Broadcast Disks method. The FASTOPTIMIZATION algorithm is a cost-efficient version of OBSP [45], while MEMORYCONSERVING is the cost-aware analogue of the standard schedule constructor algorithm of the Broadcast Disks method (Fig. 1). Thus, to facilitate the presentation of the FOMC scheme, an overview of the OBSP and the Broadcast Disks method will be given.

A. Network Architecture and Client Probabilistic Model

The network layout consists of a central server and wireless clients in a typical star topology (Fig. 1). According to the most commonly used model [17] of the system, a database is connected to the server and contains $dbsize$ equally sized items, also known as pages, which will be denoted as $l = 1, \ldots, dbsize$. The server is properly equipped, hardware and protocol wise, in order to act as a central node of a cellular network [7, 47], while any number of adequately equipped portable devices can act as clients.

As in [17] though, the approach of a single probabilistically equivalent client is adopted. Subsequently, this client accesses only a random portion of the total $dbsize$ available pages, which is denoted as $range$. Thus it holds that $range \leq dbsize$. As shown in Table I, these pages are organized in k equally sized groups called $Regions$, where $k = \lceil \frac{range}{regionsize} \rceil$, while $regionsize$ denotes the number of pages contained in a single region r_j, $j = 1, \ldots, k$. All $regionsize$ pages in r_j share the same probability of being requested by a client. The $Regions$ themselves are considered to follow the zipf probability distribution [49]. Thus the pages that comprise the $Regions$ follow the p.d.f. of (1):

$$P_p(l) = \frac{1}{regionsize \cdot \sum_{j=1}^{k} \frac{1}{j^\theta} \cdot \lceil \frac{l}{regionsize} \rceil^\theta} \qquad (1)$$

where θ is the zipf p.d.f. parameter (and as such $\theta \neq 1$) and l denotes the pages' index, $l = 1, \ldots, dbsize$.

Thus the client's p.d.f. P_p has the form of Fig. 2. The server gradually obtains an approximate form of P_p by means of an appropriate feedback mechanism as in [43]. In the context of this paper, every simulation that took place was executed under the assumption that enough time has elapsed for the server to have an accurate approximation of P_p as depicted in Fig. 2. The mathematical analysis though is totally p.d.f. agnostic. It must be stated that the only assumption is that the pages' access probability $P_p(l), l = 1, \ldots, dbsize$, are known [43].

B. The Broadcast Disks Method: A Mathematical Analysis' Outline

Once the server has acquired sufficient knowledge of client's P_p the Broadcast Disks method intervenes and acts in three phases:

1) The pages are grouped according to their access probability $P_p(l)$ in a fixed number of groups called disks. This scheme can be abstractedly represented as an array of physical disks spinning around a common axis.
2) Then, each of these disks is set to spin with a angular velocity proportional to the aggregate demand of its contained pages.
3) Finally, an imaginary set of stationary heads retrieves pages from the disks and forwards them to the broadcasting system in the same order that they have been read.

Thus the periodicity and proportionality criteria of the broadcast schedule mentioned in [17] are met.

It becomes evident that the Broadcast Disks method requires a way of defining the number and sizes of disks of step 1 and the disk angular velocities of step 2. To this end, in [45] an optimal such procedure (OBSP) was presented, the performance of which was superior to the compared alternatives. Since, in the context of this paper, this procedure will be altered to require less computational power and memory, it is imperative that a brief overview be given.

In [44] the notion of a disk's workload W_i was introduced as the aggregate client waiting time for the portion of the client

queries that refer to pages belonging to the i^{th} disk. More specifically, W_i was defined as:

$$W_i = N_i \cdot D_q \tag{2}$$

where N_i stands for the number of client queries that correspond to pages from the i^{th} disk out of a total number of N queries and D_q denotes the estimate waiting time for a single client query of this kind. Furthermore, it has been proven [44] that (2) can be rewritten as follows:

$$W_i = \left[N \cdot \sum_{l=d_{i-1}}^{d_i} P_p(l) \right] \cdot \frac{L}{2U_i} \tag{3}$$

where d_{i-1}, d_i are the page indices that define the i^{th} disk and P_p is the p.d.f. of the client queries as depicted in Fig. 2. L stands for the size of the broadcast schedule's period measured in number of pages broadcasted and U_i indicates the angular velocity of the i^{th} disk, according to Table I.

In addition, by setting $G(x) = \sum_{l=1}^{x} P_p(l)$, (3) is transformed as follows:

$$W_i = \frac{NL}{2} \cdot \frac{1}{U_i} \Big[G(d_i) - G(d_{i-1}) \Big], \quad i = 1, \ldots, NoD \tag{4}$$

where NoD represents the total number of disks.

In the context of the same work [44], the uniform workload distribution was examined, according to which:

$$W_1 = W_2 = \ldots = W_i = \ldots = W_{NoD}$$

It has also been proven that non-uniform distributions may achieve better client response times [45]. Such arbitrary distributions can still be expressed in equation form as follows:

$$p_1 \cdot W_1 = p_2 \cdot W_2 = \ldots = p_i \cdot W_i = \ldots = p_{NoD} \cdot W_{NoD} \tag{5}$$

where p_i expresses the i^{th} disk's workload balancing parameter and $p_i \in (0, \infty)$. Equation (5) through (4) becomes:

$$\frac{p_1}{U_1} \cdot \Big[G(d_1) - 0 \Big] = \frac{p_2}{U_2} \cdot \Big[G(d_2) - G(d_1) \Big] = \ldots$$
$$\ldots = \frac{p_{NoD}}{U_{NoD}} \cdot \Big[1 - G(d_{NoD-1}) \Big] \tag{6}$$

since $G(d_0) = \sum_{l=1}^{0} P_p(l) = 0$ and $G(d_{NoD}) = \sum_{l=1}^{dbsize} P_p(l) = 1$. Equation (6) can be expressed in matrix form $A \cdot X = B$ as presented in (7).

Thus, $G(d_i), i = 1, \ldots, NoD$, and therefore - since $G(x)$ is 1-1 as a strictly rising function - the d_i points themselves can be generally expressed as:

$$\begin{cases} d_i = \mathcal{F}_i(NoD, \{p_j, U_j, j{=}1, \ldots, NoD\}), \ i{=}1, \ldots, NoD \\ d_{NoD} = dbsize \end{cases} \tag{8}$$

where $\mathcal{F}_i$ denotes the relation between the NoD, p_j, U_j parameters and d_i resulting from the solution of the linear system (7).

The parameters p_j and U_j are usually assumed to be terms of uniparametrical numerical series as follows:

$$p_j = \Pi(j, \phi)$$
$$U_j = \Upsilon(j, \Delta)$$

where ϕ, Δ are arbitrarily chosen parameters.

Thus (8) can be rewritten as:

$$\begin{cases} d_i = \mathcal{F}_i(NoD, \Delta, \phi), \quad i = 1, \ldots, NoD \\ d_{NoD} = dbsize \end{cases} \tag{9}$$

Finally, an estimation of the mean delay time was given in [44] as:

$$\overline{D} = \frac{\sum_{i=1}^{NoD} W_i}{N} \overset{(4)}{=} \frac{L}{2} \sum_{i=1}^{NoD} \frac{G(d_i) - G(d_{i-1})}{U_i} =$$
$$\overset{(9)}{=} \mathcal{F}(NoD, \Delta, \phi) \tag{10}$$

Thus, the OBSP optimization scheme is established as follows:

1) Define valid value sets $S_{NoD}, S_\Delta, S_\phi$ for the NoD, Δ and ϕ parameters.
2) For every combination $S_{NoD} \times S_\Delta \times S_\phi$:
   i. Solve the linear system (7) and calculate the d_i points in order to define the corresponding disk sizes.
   ii. Proceed to the calculation of the corresponding estimated mean delay time through (10).
   iii. Keep the combination $S_{NoD} \times S_\Delta \times S_\phi$ and disks sizes that achieve the minimum $\overline{D}$.

III. The proposed FOMC scheme

As the name indicates, the goal of FOMC is to reduce the required computational power and memory. As shown in Fig. 1, FOMC comprises of the FastOptimization broadcast parameters'optimizer and the MemoryConserving schedule constructor.

A. The FastOptimization algorithm

For OBSP, optimization speed is subjected to the following tasks:

1) Calculation of the elements of the parameter matrix (7) according to the chosen $S_{NoD} \times S_\Delta \times S_\phi$ combination.
2) Solution of the linear system (7).
3) Reversal of the $G(d_i)$ values and obtain the d_i points.
4) Detection of the minimum estimated delay time $\overline{D}$ through (10).

For the above $1, 3$ and 4 tasks, which are trivial and standard, computationally optimal corresponding procedures already exist. Due to the peculiar form of the linear system (7) though, task 2, which makes up for the bulk of the computational power required by the aforementioned optimization procedure, can be significantly accelerated.

The best solution so far (incorporated in OBSP) was the utilization of the Thomas algorithm [50] designed for solving any tridiagonial linear $n \times n$ system in $O(n)$ operations instead of $O(n^3)$ required by Gaussian elimination. The method consists of three steps:

$$\begin{pmatrix} \frac{p_1}{U_1}+\frac{p_2}{U_2} & -\frac{p_2}{U_2} & 0 & 0 & \dots & 0 & 0 & 0 \\ -\frac{p_2}{U_2} & \frac{p_2}{U_2}+\frac{p_3}{U_3} & -\frac{p_3}{U_3} & 0 & \dots & 0 & 0 & 0 \\ 0 & -\frac{p_3}{U_3} & \frac{p_3}{U_3}+\frac{p_4}{U_4} & -\frac{p_4}{U_4} & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & -\frac{p_{NoD-1}}{U_{NoD-1}} & \frac{p_{NoD-1}}{U_{NoD-1}}+\frac{p_{NoD}}{U_{NoD}} \end{pmatrix} \cdot \begin{pmatrix} G(d_1) \\ G(d_2) \\ G(d_3) \\ \vdots \\ G(d_{NoD-1}) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \frac{p_{NoD}}{UNoD} \end{pmatrix} \quad (7)$$

Step 1. Calculate the a_i, b_i, c_i, d_i parameters and form the tridiagonial system in the following manner:

$$\begin{pmatrix} b_1 & c_1 & 0 & \dots & 0 & 0 \\ a_2 & b_2 & c_2 & \dots & 0 & 0 \\ 0 & a_3 & b_3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_n & b_n \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{pmatrix}$$

Step 2. Modify the coefficients as follows:

$$c'_i = \begin{cases} \frac{c_i}{b_i}, & i = 1 \\ \frac{c_i}{b_i - c'_{i-1} \cdot a_i}, & i = 2, 3, \dots, n-1 \end{cases}$$

$$d'_i = \begin{cases} \frac{d_i}{b_i}, & i = 1 \\ \frac{d_i - d'_{i-1} \cdot a_i}{b_i - c'_{i-1} \cdot a_i}, & i = 2, 3, \dots, n \end{cases}$$

Step 3. Obtain the solution by back substitution:

$$x_n = d'_n$$

$$x_i = d'_i - c'_i \cdot x_{i+1}, \quad i = n-1, n-2, \dots, 1$$

In the case of system (7):

- Step 1 requires $(n+1) + n + n - 1 = 3n$ operations, since the matrix is symmetric and $b_i = |c_{i-1}| + |a_i|,\ i = 2, \dots, n$.
- Step 2 requires $1 + 3(n-1) = 3n - 2$ operations, since $d_i = d'_i = 0,\ i = 1, \dots, n-1$.
- Step 3 requires $2(n-1) = 2n - 2$ operations.

Thus, the Thomas algorithm requires a total of $T_1 = 8n - 4$ operations.

The proposed FASTOPTIMIZATION algorithm handles the linear system solution task as follows: given (5) we define A as:

$$A = p_1 \cdot W_1 = p_2 \cdot W_2 = \dots = p_i \cdot W_i = \dots = p_{NoD} \cdot W_{NoD} \quad (11)$$

which expresses the value of the $p_i \cdot W_i,\ i = 1, \dots, NoD$ parameters. Thus:

$$A = p_i \cdot W_i \quad (12)$$

The presence of the i index in (12) may be misleading inferring dependence of A from i. However, as shown in (11) A is constant and independent from it.

Because of (4) and (12), A can be expressed as:

$$A = \frac{LN}{2} \cdot \frac{p_i}{U_i} \cdot \Big[G(d_i) - G(d_{i-1}) \Big] \quad (13)$$

Let $\widetilde{A} = \frac{A}{\frac{LN}{2}} = \frac{p_i}{U_i} \cdot \Big[G(d_i) - G(d_{i-1}) \Big]$. It holds that:

$$G(d_i) = \frac{U_i}{p_i} \cdot \widetilde{A} + G(d_{i-1}), \quad i = 1, \dots, NoD \quad (14)$$

which can be rewritten as:

$$G(d_i) = \frac{U_i}{p_i}\widetilde{A} + \frac{U_{i-1}}{p_{i-1}}\widetilde{A} + \dots + \frac{U_1}{p_1}\widetilde{A} = \widetilde{A}\sum_{j=1}^{i}\frac{U_j}{p_j}$$

Setting $i = NoD$, the above equation becomes:

$$G(d_{NoD}) = \widetilde{A}\sum_{j=1}^{NoD}\frac{U_j}{p_j}$$

and since $G(d_{NoD}) = 1$:

$$\widetilde{A} = \frac{1}{\sum_{j=1}^{NoD}\frac{U_j}{p_j}} \quad (15)$$

Finally, though (14):

$$G(d_{i-1}) = G(d_i) - \frac{U_j}{p_j}\widetilde{A}, \quad i = NoD, \dots, 2 \quad (16)$$

and thus the solution is obtained by back substitution.

Equation (15) requires $2NoD$ operations, while (16) requires $2(NoD - 1)$, since the $\frac{U_i}{p_i}$ parameters have been calculated in (15). Thus, the FASTOPTIMIZATION algorithm requires a total of $T_2 = 4NoD - 2$ operations. The size of the parameter matrix n and the number of disks NoD are connected through the relation: $n = NoD - 1$ and thus, $T_2 = 4n + 2$.

Based on the above, OBSP requires $T_1 = 8n - 4$ operations, while the FASTOPTIMIZATION algorithm requires $T_1 = 4n + 2$ operations. (The computational complexity of the remaining tasks of each algorithm is trivial). It is obvious that $T_1 > T_2$ for every $n > 1$, since $n \in N^+$. Furthermore, for $n \gg 1$, $\frac{T_2}{T_1} \simeq \frac{4n}{8n} = 50\%$. This means that the FASTOPTIMIZATION algorithm requires just the half of the computational operations compared to the OBSP algorithm. Moreover, the simplicity of the new algorithm's implementation means it easier to achieve its full theoretical potential (due to implementation issues $\frac{T_2}{T_1}$ usually ranges from 10% to 20%).

B. The MEMORYCONSERVING *algorithm*

The standard broadcast schedule constructor algorithm of the Broadcast Disks method [17] states that each disk is to be split in smaller chunks, which then are to be broadcasted in a round-robbin manner as shown in Fig. 3. The default broadcast schedule constructor algorithm can be described as follows:

1) Calculate the max_chunks as the Least Common Multiple of the disks' velocities U_i, where $i = 1, \dots, NoD$
2) Split each disk into $num\_chunks_i = \frac{max\_chunks}{U_i}$ chunks
3) Broadcast the $C_{i,j}$ chunks in a round robin manner

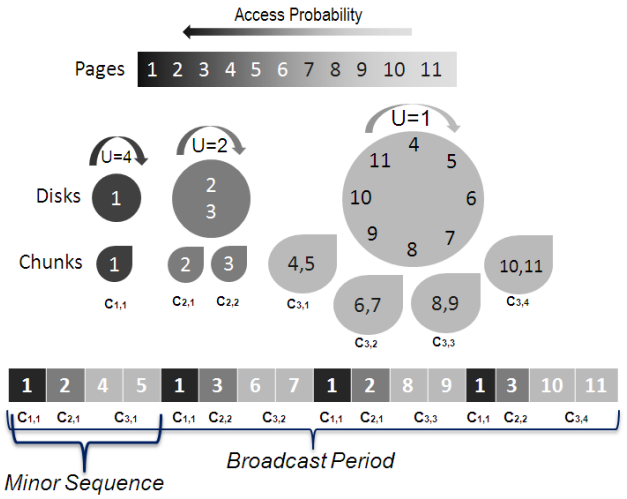

Fig. 3. The broadcast schedule construction procedure.

The above broadcast schedule constructor algorithm defines the broadcasting of pages' chunks. Yet, for a realistic implementation, it is advantageous to know which single page is to be broadcasted at any given moment. The proposed approach:

1) facilitates the creation of system's simulators as it obviously offers reduced complexity.
2) it promotes the efficiency of client-side traffic prediction scheme.

A relatively simple way to implement a procedure that accepts the current time slot index as input and produces the page to be broadcasted as output is the following:

1) use the default broadcast schedule constructor algorithm to create a whole period of broadcast program and store it in a cache memory section
2) at any given time slot T, broadcast the page with index $l_B = mod(T, L)$, where L indicates the size of the broadcast schedule and as such:

$$L = \sum_{i=1}^{NoD} \lceil \frac{d_i - d_{i-1}}{num\_chunks_i} \rceil \cdot num\_chunks_i \cdot U_i \quad (17)$$

where $d_0 = 0$.

This approach, though simple in conception, has two performance-degrading disadvantages:

1) L increases exponentially as the number of disks and their corresponding velocities increase [17], and therefore so does the size of the required cache memory size.
2) Constructing and storing an enormous broadcast schedule to cache memory adds a considerable overhead to the system's performance.

A more memory-conserving approach would calculate the index of the page to be broadcasted, e.g. l_B, directly from the current time slot, e.g. T, without storing any parts of the broadcast schedule to memory. The proposed MEMORYCONSERVING algorithm is demonstrated by Algorithm 1.

More specifically, in lines $1-3$, the Algorithm 1 calculates the constants $max\_chunks, num\_chunks_i, M$, which are described in Table II, and L which denotes the size of the broadcast schedule. Then, in line 4, the page index that represents the current page l_c is calculated. As denoted by its formula, the value range of l_c is $[1, L]$, since the notion of l_c is the page index inside the broadcast program, as depicted in Fig. 4. Then, given the values of l_c and M the algorithm proceeds to select the minor sequence ms as well as the index of page inside the minor sequence l_{ms}. Obviously, the value range of l_{ms} is $[1, M]$. In lines $7-12$ the current $disk$ is calculated and the value of l_{ms} is updated to

TABLE II
MEMORYCONSERVING ALGORITHM NOTATION

Symbol	Description
DS_i	Disk size: the number of pages contained in i^{th} disk, $i = 1, \ldots, NoD$
max_chunks	The Least Common Multiple of the disks' velocities U_i, $i = 1, \ldots, NoD$
num_chunks_i	The number of chunks the i^{th} disk is split into, $i = 1, \ldots, NoD$
$C_{i,j}$	One of the j chunks, $j = 1, \ldots, num\_chunks_i$, derived from disk i, $i = 1, \ldots, NoD$
$\|C_{i,j}\|$	The number of pages contained in $C_{i,j}$ chunk, $\|C_{i,j}\| = \lceil \frac{DS_i \cdot U_i}{max\_chunks} \rceil$
M	The length of the Minor Sequence as depicted in Fig. 3, $M = \sum_{i=1}^{NoD} \|C_{i,1}\|$, since all chunks of the same disk are equally sized i.e. contain the same number of pages

Algorithm 1 The MEMORYCONSERVING algorithm.

Input: The current timeslot T, the disks' sizes DS_i and the velocities U_i of the NoD disks, an arbitrarily value *ALT*.
Output: The page to be broadcasted, l_B.

1: Fill the $|C_{i,1}| = \lceil \frac{DS_i \cdot U_i}{max\_chunks} \rceil$ and the $num\_chunks_i = \frac{max\_chunks}{U_i}, i = 1, \ldots, NoD$ arrays
2: $M = \sum_{i=1}^{NoD} |C_{i,1}|$
3: $L = M \cdot max\_chunks$
/ Calculate the current page index l_c */*
4: $l_c = \mod(T-1, L) + 1$
/ Calculate the minor sequence index ms */*
5: $ms = \lceil \frac{l_c}{M} \rceil$
/ Calculate the page index l_{ms} in minor sequence */*
6: $l_{ms} = \mod(l_c - 1, M) + 1$
/ Calculate the disk index */*
7: $disk = 0$
8: **while** $l_{ms} > 0$ **do**
9: $disk = disk + 1$
10: $l_{ms} = l_{ms} - |C_{disk,1}|$
11: **end while**
12: $l_{ms} = l_{ms} + |C_{disk,:}|$
/ Calculate the chunk index */*
13: $chunk = \mod(ms - 1, num\_chunks_{disk}) + 1$
/ Calculate the range limits of l_B */*
14: $low\_limit = \sum_{i=1}^{disk-1} DS_i$
15: $up\_limit = \sum_{i=1}^{disk} DS_i$
/ Calculate the page l_B to be broadcasted */*
16: $l_B = low\_limit + (chunk - 1)|C_{disk,1}| + l_{ms}$
17: **if** $(l_B > up\_limit)$ **then**
18: $l_B = ALT$
19: **end if**

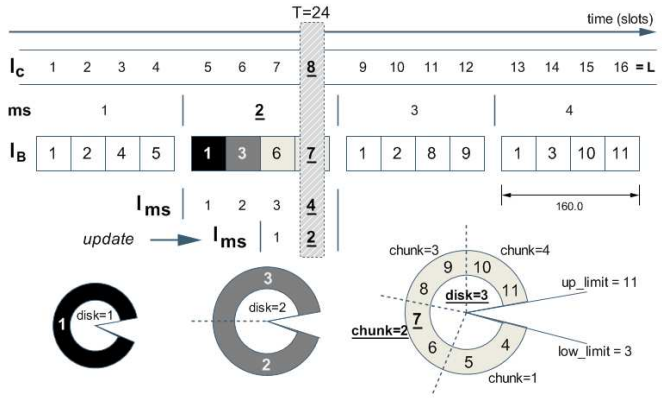

Fig. 4. An illustration of the MEMORYCONSERVING algorithm.

TABLE III
CPU TIME ASSESSMENT SPECIFICATIONS.

Parameter	Value
Test CPU model	x86 Family 6 model 15 Stepping 11 GenuineIntel
Number of cores	4× 2672MHz
Test RAM	4GB DDR2 800 MHz
Operating System	Windows XP Professional
Version	5.1.2600 SP3 Build 2600
Programming language	Single-threaded MATLAB M-code
Environment	MATLAB R2009b
CPU Usage during execution	Steady 100% on one dedicated core*
Application Priority	13 ("High")

*: *Measured by means of Windows Task Manager*

show the sub-index of the page inside the $C_{disk,chunk}$, where $chunk \in [1, num\_chunks_{disk}]$ is calculated in the next line. The parameters low_limit and up_limit of lines 14 and 15 define the database index range of l_B. Given the values of the above parameters, in line 16 of Algorithm 1, the page to be broadcasted, l_B, is calculated.

It is possible, however, for l_B to be out of bounds. As mentioned in [17], this is due to the fact that dividing a disk into an integer number of chunks is usually not possible without the insertion of an appropriate number of dummy pages. In this case, an arbitrary page with index e.g. *ALT* is broadcasted (lines $17-19$). This can be selected as e.g. one of the most popular pages.

As a minor notice, the function $y = mod(x-1, K)+1, x \in N_*^+$ of steps 4, 6 and 13 is used to cycle through elements $[1..K]$, given an incremental slot number x.

It must be stated that constants in lines $1-3$ of Algorithm 1 need only be calculated once, and can even be skipped and replaced with inline calculations where required. Thus it is obvious that the amount of memory required by Algorithm 1 is minimal.

IV. COST EVALUATION

In order to obtain a realistic measure of the cost-efficiency of the presented algorithms, they were implemented in MATLAB alongside their corresponding rivals. Standard, mainstream computer hardware and software was used, presented in Table III. The MATLAB profiler was employed to measure the CPU time spent [51, 52] by each algorithm, when assigned the same task, as well as the required memory. Notice that in terms of client waiting time all algorithms perform the same [45]. The new, cost-related results, along with more detailed descriptions are given in the following subsections.

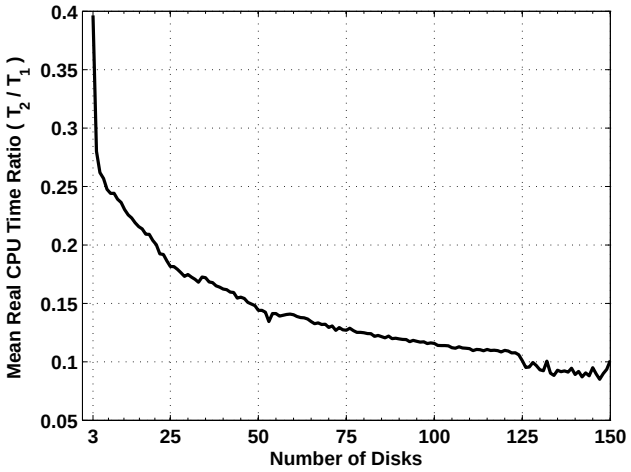

Fig. 5. OBSP (T_1) vs the FASTOPTIMIZATION algorithm (T_2).

A. Evaluating the FASTOPTIMIZATION algorithm

Both OBSP and FASTOPTIMIZATION algorithm were implemented, receiving the arrays $(p_i, U_i, i = 1, \ldots, NoD)$ as input, and producing the corresponding $G(d_i), i = 1, \ldots, NoD$ as output. The NoD parameter was set to vary in the range $[3, \ldots, 150]$. For each single value of NoD, the CPU time spent was recorded, in T_1 and T_2 for OBSP and the FASTOPTIMIZATION algorithm respectively. The procedure was repeated 100 times in order to obtain a more reliable estimate of the aforementioned times. Then, their mean values were calculated and their ratio $\frac{\overline{T_2}}{\overline{T_1}}$ was recorded and illustrated in Fig. 5.

Even though it is obvious that the FASTOPTIMIZATION algorithm outperforms OBSP in every single case, the ratio $\frac{\overline{T_2}}{\overline{T_1}}$ is much lower than its theoretical values (up to 10% rather than 50%). This is due to the fact that, when calculating an algorithm's complexity, as in Section III-A, the memory allocation, memory-to-CPU-registers and CPU-registers-to-memory transfer commands are not taken into account, as they are subjected to the CPU architecture. Yet these operations also take up a considerable portion of the total CPU time. In addition, OBSP demands the allocation and manipulation of four separate variable arrays, namely a_i, b_i, c_i, x_i. On the other hand, the FASTOPTIMIZATION algorithm requires only one array, the x_i, just to hold the results. Thus the memory allocation and access commands are bound to be much less in its case. Finally, the CPU time of a function is also implementation-dependent. Better implementations of OBSP may be possible than the one used in the context of this paper. Yet this brings to light another advantage of the new

TABLE IV
COMPARISON OF CPU AND MEMORY REQUIREMENTS.

θ	**Optimal NoD**	**CPU time (*sec*)**	**Items scheduled in one pass**
DEFAULT BROADCAST DISKS SCHEDULE CONSTRUCTOR			
0.3	8	0.52	30240
0.4	12	67.54	3963960
0.5	15	3499.93	205405200
0.6	47	$> 24h$	$1.3968 \cdot 10^{10}$
1.1	49	$> 24h$	$1.1651 \cdot 10^{20}$
1.5	68	$> 24h$	$1.34 \cdot 10^{28}$
1.8	80	$> 24h$	$5.5596 \cdot 10^{19}$
MEMORYCONSERVING			
$[0.3, 1.8]$	$[8, 80]$	$7.5 \cdot 10^{-6}$	1
OBSP			
$[0.3, 1.8]$	$[8, 80]$	1603.3	n/a
FASTOPTIMIZATION			
$[0.3, 1.8]$	$[8, 80]$	179.9	n/a
$dbsize = 5000$, $range = 5000$, $regionsize = 1$.			

algorithm: its simplicity and easy of implementation make it easy to achieve its full theoretical potential, in contrast with the OBSP which requires programmatic optimization as well.

Notice that, as the OBSP definition of Section II-B dictates, the tridiagonal system of (7) must be solved $S(NoD) \cdot S(\Delta) \cdot S(\phi)$ times for a single optimization task, where $S(.)$ represents the number of elements contained in the parameter set of a variable. Thus, the advantage of FASTOPTIMIZATION in speed is magnified correspondingly. For common value sets $\Delta = 1 \ldots 100$, $\phi = 0 : 0.1 : 2$ and $NoD = 3 \ldots 150$ [45]:

- OBSP takes 1603.3 seconds to complete
- FASTOPTIMIZATION takes 179.9 seconds to complete

as shown in Table IV.

An overhead of 0.5 hours means that OBSP cannot be used in push systems that must adapt to the client needs by the minute. OBSP could still be employed through the use of parallel programming techniques or specialized hardware, but since FASTOPTIMIZATION produces the same results without such needs, its superiority is obvious.

B. Evaluating the MEMORYCONSERVING *algorithm*

In a similar fashion with the evaluation of FASTOPTIMIZATION vs. OBSP, the MEMORYCONSERVING algorithm was implemented in the environment of Table III, alongside its predecessor, the default schedule constructor of the Broadcast Disks method. For several client cases, the CPU time and memory required by each algorithm was recorded. The results of the Broadcast Disks algorithm are displayed in Table IV.

It is readily obvious that the default schedule constructor is not appropriate for adaptive push systems. Hours to days of CPU time mean that by the moment the schedule is produced, the clients preferences will have changed to the degree that the schedule would offer no performance advantage any more. Boosting performance with specialized hardware could perhaps make the algorithm applicable, though at a precariously increased implementation cost. Also notice the fact that this algorithm cannot take advantage of parallel computing schemes due to its nature.

The reason for this performance are the excessively long broadcast schedules produced. The algorithm employs the least common multiple of disk velocities and disk padding to define chunks to be broadcasted. When the optimal NoD increases, the least common multiple of speeds becomes excessively large, and the disk chunks extremely small. The produced schedule is then even billions of data items long. This also means that the required memory to hold such a schedule would be terabytes large or more.

MEMORYCONSERVING on the other hand requires no schedule storage as it is able to calculate the page to be broadcasted on the fly. The CPU time required for this task is just $7.5 \cdot 10^{-6}$ seconds on the test machine. MEMORYCONSERVING is also able to take advantage of parallel computing schemes, by assigning the calculation of ranges of time moments to different processors or threads.

V. FLUCTUATIONS OF THE OPTIMAL BROADCAST PARAMETERS

The performance related observations of the previous sections had a direct impact on the system's simulation time as well, enabling the study of numerous client cases in a relatively small amount of time. For the first time every single one of the $S = \{dbsize, range, regionsize, \theta\}$ set of system defining parameters varied in a predefined range. Purpose was the study of the linearity or non-linearity of the behavior of the optimal values of the $P = \{NoD, \Delta, \phi\}$ performance defining parameters. This revealed another problem: a way of 5-dimensional representation (the four parameters of the S set and one from the P set) had to be improvised. The result is depicted in Fig. 6 and 7.

Each such diagram refers to a different $\{\theta, \frac{range}{dbsize}\}$ combination and illustrates the behavior of one of the parameters in the P set. In each diagram, every concentric circle corresponds to a different $dbsize$ value. Every radial line represents a different value of the $\frac{regionsize}{range}$ ratio. Each intersection of concentric circle-radial line represents the mean value $\overline{p}$ of the parameter in discussion over all the different values of the $\frac{regionsize}{range}$ ratio. By moving from this point along the radial line in a centrifugal fashion, values greater than $\overline{p}$ are represented. Likewise, by moving in a centripetal fashion values smaller than $\overline{p}$ are represented. The point alongside a radial line, which is equidistant from the current circle and the next, represents the maximum value of the parameter in discussion, over all different values of the $\frac{regionsize}{range}$ ratio. Correspondingly, its symmetrical point represents the minimum value of the parameter. Lines interconnect all points, thus forming the pentagons shown in Fig. 6 and 7.

Subsequently, concentric and resemblant pentagons like the ones presented in Fig. 6 yield linearity in the behavior of the parameter in discussion. In contrast, abnormalities like the ones presented in Fig. 7 yield non-linear behavior. These two diagrams, though indicative - as space restrictions hindered the presentation of the totality of the results - present the general conclusion flawlessly: for small values of the $\frac{regionsize}{range}$ ratio,

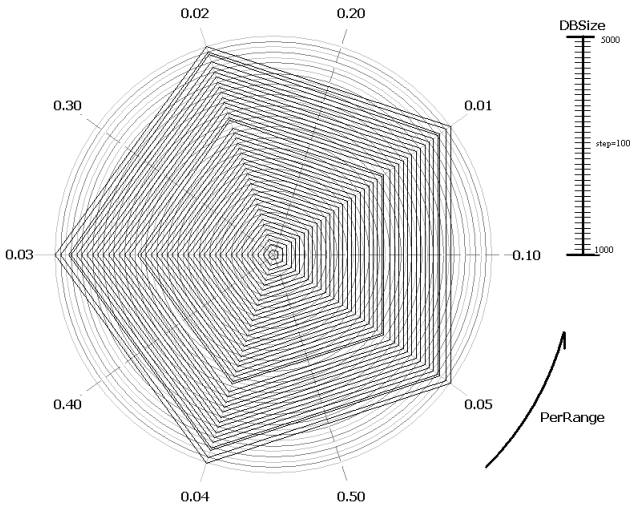

Fig. 6. Illustration of the behavior of the NoD parameter, for $\frac{range}{dbsize} = 0.3$ and $\theta = 0.9$ while $dbsize \in [1000, 5000]$ and $\frac{regionsize}{range} \in [0.01, 0.05]$

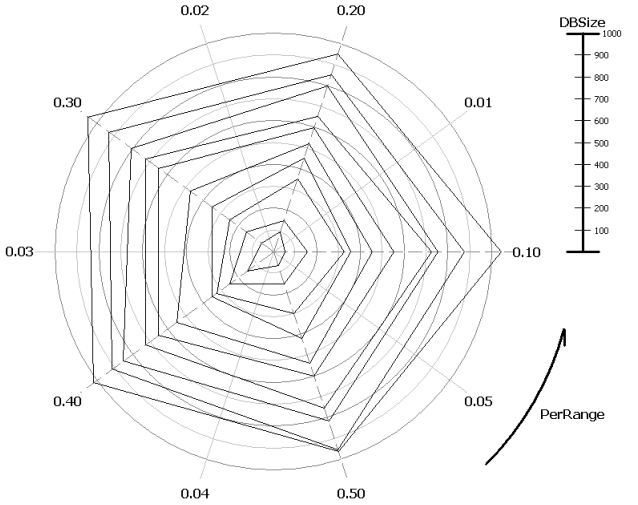

Fig. 7. Illustration of the behavior of the Δ parameter, for $\frac{range}{dbsize} = 0.3$ and $\theta = 0.9$ while $dbsize \in [100, 1000]$ and $\frac{regionsize}{range} \in [0.1, 0.5]$

linearity characterizes the behavior of all parameters of the P set. For higher values though, the client's p.d.f. greatly changes in form, and thus non-linearity makes its appearance.

VI. Conclusion

Up to now, cost criteria when designing performance related algorithms for broadcast systems have been overlooked, even though the primary advantage of these systems is their low implementation cost. In this paper, the novel Fast Optimization - Memory Conserving (FOMC) scheme was introduced and evaluated. The proposed FOMC establishes and handles two main cost criteria, namely the computational and memory requirements of a broadcast server, by employing two algorithms: the FastOptimization and the MemoryConserving algorithm, respectively.

Being a cost-aware version of the classical Broadcast Disks method, FOMC was compared cost-wise with its predecessor. It was shown that while FOMC is readily applicable in the realm of adaptive push systems, the cost-unaware Broadcast Disks does not meet the lightweight standards of these systems.

The cost-related observations were readily exploited in order to run thousands of whole system simulations, ranging every single one of the system's configuration parameters, in a small amount of time. Linearity between the achieved client mean waiting time and the parameters' values was observed when the clients access only a small portion (i.e $< 5\%$) of the available data items.

References

[1] S. Kang, S. Choi, S. Choi, G. Lee, J. Lew, and J. Lee, "Scheduling data broadcast based on multi-frequency in mobile interactive broadcasting," *IEEE Trans. Broadcasting*, vol. 53, no. 1, pp. 405–411, 2007.

[2] T. Yoshihisa, M. Tsukamoto, and S. Nishio, "A scheduling scheme for continuous media data broadcasting with a single channel," *IEEE Trans. Broadcasting*, vol. 52, no. 1, pp. 1–10, 2006.

[3] T. Yoshihisa, M. Tsukamoto, and S. Nishio, "A scheduling protocol for continuous media data broadcasting with large-scale data segmentation," *IEEE Trans. Broadcasting*, vol. 53, no. 4, pp. 780–788, 2007.

[4] L-S.Juhn and L-M.Tseng, "Adaptive fast data broadcasting scheme for video-on-demand service," *IEEE Trans. Broadcasting*, vol. 44, no. 2, pp. 182–185, 1998.

[5] L-S.Juhn and L-M.Tseng, "Fast data broadcasting and receiving scheme for popular video service," *IEEE Trans. Broadcasting*, vol. 44, no. 1, pp. 100–105, 1998.

[6] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Continuous flow wireless data broadcasting for high-speed environments," *IEEE Trans. Broadcasting*, vol. 55, no. 2, pp. 260–269, 2009.

[7] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Exploiting locality of demand to improve the performance of wireless data broadcasting," *IEEE Trans. Vehicular Technology*, vol. 55, no. 4, pp. 1347 – 1361, 2006.

[8] XM, "Xm traffic, weather," http://www.xmradio.com.

[9] DirectBand Network, "Microsoft Smart Personal Objects Technology (SPOT)," http://www.microsoft.com/resources/spot/.

[10] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieberaruch, "Minimizing service and operation costs of periodic scheduling," in *Proc. SODA'98*, USA.

[11] V. Gondhalekar, R. Jain, and J. Werth, "Scheduling on airdisks: Efficient access to personalized informationservices via periodic wireless data broadcast," in *Proc. IEEE Int. Conf. on Communications, ICC'97*, Canada.

[12] C. Kenyon and N. Schabanel, "The data broadcast problem with non-uniform transmission times," in *Proc. SODA'99*, USA.

[13] M. Ammar and J. Wong, "On the optimality of cyclic transmission in teletext systems," *IEEE Trans. Communications*, vol. 35, no. 11, pp. 1159–1170, 1987.

[14] S. Hameed and N. Vaidya, "Efficient algorithms for scheduling data broadcast," *ACM/Baltzer Wireless Networks*, vol. 5, no. 3, pp. 183–193, 1999.

[15] N. Vaidya and S. Hameed, "Scheduling data broadcast in asymmetric communication environments," *Wireless Networks*, vol. 5, no. 3, pp. 171–182, 1999.

[16] N. Schabanel, "The data broadcast problem with preemption," in *Proc. STACS'00)*, pp. 181–192.

[17] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks: data management for asymmetric communication environments," in *Proc. SIGMOD'95*, USA.

[18] S. Acharya, M. Franklin, and S. Zdonik, "Prefetching from broadcast disks," in *Proc. ICDE'96*, pp. 276–285.

[19] K. Wu, P. Yu, and M. Chen, "Energy-efficient caching for bandwidth-limited wireless mobile computing," in *Proc. ICDE'96*, pp. 335–343.

[20] M. Chen, P. Yu, and K. Wu, "Optimizing index allocation for sequential data broadcasting in wireless mobile computing," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 1, pp. 161–173, 2003.

[21] J. X. et al., "An error-resilient and tunable distributed indexingscheme for wireless data broadcast," *IEEE Trans. Knowledge and Data Eng.*, vol. 18, no. 3, pp. 392–404, 2006.

[22] C. Hu and M. Chen, "Adaptive multi-channel data dissemination: Support of dynamic traffic awareness and push-pull time balance," *IEEE Trans. Vehicular Technology*, vol. 54, no. 2, pp. 192–203, 2005.

[23] J. Hu, K. Yeung, G. Fend, and K. Leung, "A novel push-and-pull hybrid data broadcast scheme for wireless information networks," in *Proc. ICC'00)*, pp. 1778–1782.

[24] Y. Chang, C. Yang, , and J. Shen, "A binary-number-basedapproach to data broadcasting inwireless information systems," in *Proc. IEEE Intl Conf. Wireless Networks*, 2005.

[25] W. Peng and M. Chen, "Efficient channel allocation tree generation for data broadcasting in a mobile computing environment," *Wireless Networks*, vol. 9, no. 2, pp. 117–129, 2003.

[26] H. Leung, J. Yuen, K. Lam, and E. Chan, "Enhanced multiversiondata broadcast scheme for time-constrained mobile computingsystems," in *Proc. DEXA'02*.

[27] T. B. et al., "The datacycle architecture," *Comm. ACM*, vol. 35, no. 12, pp. 71–81, 1992.

[28] J. S. et al., "Efficient concurrency control forbroadcast environment," in *Proc. ACM SIGMOD'99*.

[29] E. Pitoura and R. Chrusanthis, "Scalable processing of read-only transactions in broadcast push," in *Proc. ICDCS'99*.

[30] E. Pitoura and R. Chrusanthis, "Multiversion data broadcast," *IEEE Trans. Computers*, vol. 51, no. 10, pp. 1224–1230, 2002.

[31] X. Zhang, W.-C. Lee, P. Mitra, and B. Zheng, "Processing transitive nearest-neighbor queries in multi-channel access environments," in *Proc. EDBT'08*, pp. 452–463.

[32] C.-H. Hsu, G. Lee, and A. Chen, "An efficient algorithm for near optimal data allocation on multiple broadcast channels," *Distrib. Parallel Databases*, vol. 18, no. 3, pp. 207–222, 2005.

[33] W.-C. Lee and B. Zheng, "Dsi: A fully distributed spatial index for location-based wireless broadcast services," in *Proc. IEEE ICDCS'05*, pp. 349–358.

[34] G. Lee, S. Lo, and A. Chen, "Data allocation on wireless-broadcast channels for efficient query processing," *IEEE Trans.Computers*, vol. 51, no. 10, pp. 1237–1252, 2002.

[35] A. Si and H. Leong, "Query optimization for broadcast database," *Data and Knowledge Eng.*, vol. 29, no. 3, pp. 351–380, 1999.

[36] G. Lee and S.-C. Lo, "Broadcast data allocation for efficient access of multiple data items in mobile environments," *Mob. Netw. Appl.*, vol. 8, no. 4, pp. 365–375, 2003.

[37] J. Huang and M. Chen, "Broadcasting dependent data fororddered queries without replication in a multi-channel mobile environment," in *Proc. ICDE'03)*.

[38] Y. Chung and M. Kim, "Efficient data placement for wireless broadcast," *Distributed and Parallel Database*, vol. 9, no. 2, pp. 133–150, 2001.

[39] J. Huang and M. Chen, "Broadcast program generation forunordered queries with data replication," in *Proc. SAC'03*.

[40] F. Martinwz, U. Gonzalez, and I. Stojmenovic, "A parallelhill climbing algorithm for pushing dependent data in clients-providers-servers systems," in *Proc. ISCC'02*, 2002.

[41] A. Bar-Noy, J. Naor, , and B. Schiber, "Pushing dependent datain clients-providers-servers systems," in *Proc. ACM MobiCom'00*, pp. 222–230.

[42] C.K.Liaskos, S.G.Petridou, G.I.Papadimitriou, P.Nicopolitidis, M.S.Obaidat, and A. S. Pomportsis, "A novel clustering-driven approach to wireless data broadcasting," in *15th Annual Symposium of the IEEE/CVT*, Belgium.

[43] C. Liaskos, S. Petridou, G. Papadimitriou, P. Nicopolitidis, M. Obaidat, and A. Pomportsis, "Clustering-driven wireless data broadcasting," *IEEE Wireless Communications Magazine*, to appear.

[44] C. Liaskos, S. Petridou, G. Papadimitriou, P. Nicopolitidis, and A. Pomportsis, "An analytical approach to the design of wireless broadcast disks systems," in *Proc. ISADS'09*, Greece.

[45] C. Liaskos, S. Petridou, and G. Papadimitriou, "On the analytical performance optimization of wireless data broadcasting," *IEEE Trans. Vehicular Technology*, to appear, 2009.

[46] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Using learning automata for adaptive push-based data broadcasting in asymmetric wireless environments," *IEEE Trans. Vehicular Technology*, vol. 51, no. 6, pp. 1652–1660, 2002.

[47] P. Nicopolitidis, G. Papadimitriou, and A. Pomportsis, "Multiple-antenna data broadcasting for environments with locality of demand," *IEEE Trans. Vehicular Tech-*

nology, vol. 56, no. 5, pp. 2807 – 2816, 2007.

[48] P. Nicopolitidis, G. Papadimitriou, M. Obaidat, and A. Pomportsis, "Performance optimization of an adaptive wireless push system in environments with locality of demand," *Computer Communications, Elsevier*, vol. 29, no. 13-14, pp. 2542–2549, 2006.

[49] L. Pietronero, E. Tosatti, V. Tosatti, and A. Vespignani, "Explaining the uneven distribution of numbers in nature: The laws of benford and zipf," *Physica A*, pp. 297–304, 2001.

[50] S. Conte and C. DeBoor, *Elementary Numerical Analysis: An Algorithmic Approach*. McGraw-Hill Higher Education, 1980.

[51] E. Alexandre, A. Pena, and M. Sobreira, "Low-complexity bit-allocation algorithm for mpeg aac audio coders," *IEEE Signal Processing Letters*, vol. 12, no. 12, pp. 824–826, 2005.

[52] R. Sukkar, S. Herman, A. Setlur, and C. Mitchell, "Reducing computational complexity and response latency through the detection of contentless frames," in *Proc. ICASSP'00*, Belgium, pp. 3751 – 3754.

Christos K. Liaskos ($M'08$) received the Diploma degree in electrical and computer engineering in 2004 and the M.S. degree in medical informatics in 2008 from the Aristotle University of Thessaloniki, Macedonia, Greece. He is currently working towards the Ph.D. degree in communication networks. His research interests include wireless networks, neural networks and fractal compression algorithms.

Sophia G. Petridou ($M'08$) received the Diploma and Ph.D. degrees in Computer Science from the Aristotle University of Thessaloniki, Greece in 2000 and 2008 respectively. Her research interests include clustering, optical and wireless networks.

Georgios I. Papadimitriou ($M'89$, $SM'02$) received the Diploma and Ph.D. degrees in Computer Engineering from the University of Patras, Greece in 1989 and 1994 respectively. In 1997 he joined the faculty of the Department of Informatics, Aristotle University of Thessaloniki, Greece, where he is currently serving as an Associate Professor. His main research interests include optical networks and wireless networks. Prof. Papadimitriou is Associate Editor of five IEEE Journals. He is co-author of three books published by Wiley. He is author or coauthor of 75 journal and 85 conference papers. He is a Senior Member of the IEEE.