Broadcast Levels: Efficient and Lightweight Schedule Construction for Push-Based Data Broadcasting Systems

Stathis Mavridopoulos, Petros Nicopolitidis, *Senior Member, IEEE*,
AQ1 Georgios Papadimitriou, *Senior Member, IEEE*, and Panagiotis Sarigiannidis, *Member, IEEE*

***Abstract*—In wireless push-based systems traffic is asymmetric meaning that the server has much more bandwidth available, as compared to the clients. Since it is often not possible or desirable for the clients to send any requests, the server should broadcast the data periodically. The performance of this system relies heavily on the proper scheduling of the broadcast data. The well known algorithm proposed by Vaidya and Hameed battles this challenge and achieves state-of-the-art performance. However, this performance comes with a computational complexity linear to the number of data items that are broadcast. Furthermore, Vaidya's and Hameed's proposed solution for this time complexity degenerates performance. This paper proposes a scheduling algorithm for wireless push-based data broadcasting systems, which performs near optimal and has low computational complexity. Performance evaluation results are presented to demonstrate its efficient performance despite its low computational demands.**

***Index Terms*—Broadcast, push-based system, wireless, low computational complexity, adaptive, multiple channels.**

I. Introduction

WIRELESS data broadcasting is a field of significant interest as technologies like mobile computing and wireless networks continuously develop. Data broadcasting has emerged as an efficient mean for information disseminating over asymmetric wireless environments. In these environments, asymmetry arises from the fact that the downstream capacity to clients from servers is much greater than the upstream communication capacity. Such asymmetric communications environments, characterize a great variety of applications, such as satellite or cellular-based wireless networks. Examples of data broadcasting applications include information retrieval applications, like traffic information systems, public safety applications and satellite communications. In such applications, client requests for data items are usually overlapping. As a result, data broadcasting stands to be an efficient solution, since the broadcast of a single information item is likely to satisfy a (possibly large) number of client requests [1]–[7].

Push-based and pull-based systems are the two main approaches used in wireless data broadcasting. In pull-based systems, clients explicitly request ("pull") data from the server in order to provide them to locally running applications. In the case of push-based systems the system is described by a relatively larger downstream communication capacity compared to the upstream communication capacity. In the push-based architecture, data is pushed continuously and repeatedly from the Broadcast Server (BS) out to the clients. Depending on the architecture of a such system there could be minimal or no feedback from the clients to the server, thus the broadcast schedule must be carefully planned and constructed in order to satisfy the demands of the clients.

A critical factor of the performance of a wireless push-based system is the proper construction of the broadcast schedule of the information items (or data items). The broadcast schedule specifies when each data item is to be transmitted. Clients demand different data items with different demand probabilities, so a broadcast schedule that exploits this fact and broadcasts popular data items more often can achieve better performance. One of the most critical performance metric in data broadcasting systems is the average client mean access time, or average delay. It is defined as the amount of time the client has to wait from the moment it demands a data item until the item is received from the broadcast.

This paper investigates scheduling algorithms for push-based systems. We propose a new method and compare it with the method proposed by Vaidya and Hameed [1]. Simulation results reveal similar performance compared to the method of [1], which however is achieved with a very low computational complexity, whereas the method of [1] has a complexity linear to the number of items broadcast by the Broadcast Server. In our study we consider a database for the Broadcast Server that is divided into data items. There is a different demand probability for each data item from the clients, which is known a priori to a server, as also happens in [1]. The server uses this knowledge to construct a structure called *levels*, and uses the proposed algorithm to create its broadcast schedule. The proposed algorithm is dynamic and could be modified to construct broadcast schedules in an adaptive

Manuscript received October 6, 2014; revised April 2, 2015 and April 20, 2015; accepted May 23, 2015. *(Corresponding author: Petros Nicopolitidis.)*

S. Mavridopoulos, P. Nicopolitidis, G. Papadimitriou are with the Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece (e-mail: petros@csd.auth.gr).

P. Sarigiannidis is with the Department of Informatics and Telecommunication Engineering, University of Western Macedonia, Kozani, Greece.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TBC.2015.2444334

system with no a priori knowledge of the data items demand probabilities.

The rest of the paper is organized as follows. Section II overviews the related research on push-based data broadcasting. Section III presents our proposed scheduling algorithm and the level structure. Section IV contains an evaluation of the performance of our scheme. An analytical study of the time complexity of our method is described in Section V. Finally, a summary is presented in Section VI.

II. Related Research

A. Broadcast Disks

One of the first and well-known concept for solving the problem of constructing the broadcast schedules for asymmetric communication environments is Broadcast Disks [2], [4], [5]. This method takes advantage of research results showing that in order to minimize mean access time for clients, and thus improve performance, broadcast schedules must be periodic and the variance of spacing between consecutive instances of the same item must be reduced.

In order to accomplish that, the Broadcast Disks program construction uses a structure called Broadcast Disk, which is an abstract container of data items. Each Disk "rotates" at a different speed. The broadcast schedule is constructed by assigning data items to disks of varying sizes and speeds and then multiplexing the disks on the broadcast channel. By storing items that are requested more often, to faster disks and items demanded less to slower disks, performance is increased. It can be seen from Fig. 1, which shows an example of producing a broadcast program via the Broadcast Disks approach. In this example, items with a low id are demanded with a higher demand probability compared to items with a higher id and are thus marked as "hot". It can be seen that such items with higher probability of being demanded, appear more often in the final broadcast schedule than "cold" items.

The reason that Broadcast Disks performs better than a simple broadcast schedule, which broadcasts items based on their popularity, can be understood through the so called Bus Stop Paradox (or else called Inspection paradox). According to this paradox, if the inter-arrival rate of items at the broadcast channel is fixed, e.g., items are broadcast every 30 msec, then the expected delay for a client demanding an item at a random time will be 15 msec, which is the average time between successive broadcasts of the same item. On the contrary, if successive instances of the same item are broadcast according to a Poisson process, on average every 30 msec, then the client will have to wait for 30 msec because Poisson distribution is unaware of the past arrivals and thus the time until the next item broadcast is independent of how long it has been since the broadcast of the same item.

However, Broadcast Disks is not an optimal technique. First, it cannot be used in real time systems. In order to achieve good performance an external entity should carefully select the system parameters, mainly the number of disks used, their relative speeds and the assignment of items to each disk, in order to fine-tune the schedule. But even if one suggested an automated method for parameter selection in order to create the broadcast schedule, the Broadcast Disks method has the disadvantage of introducing "null pages" in the broadcast (Fig. 1). Those null pages are actually spots of inactivity in the broadcast schedule, idle periods when there is nothing broadcast. Null pages arise from the nature of producing the schedule in Broadcast Disks and in some cases they can cause dramatic negative effects in performance [2]–[4].

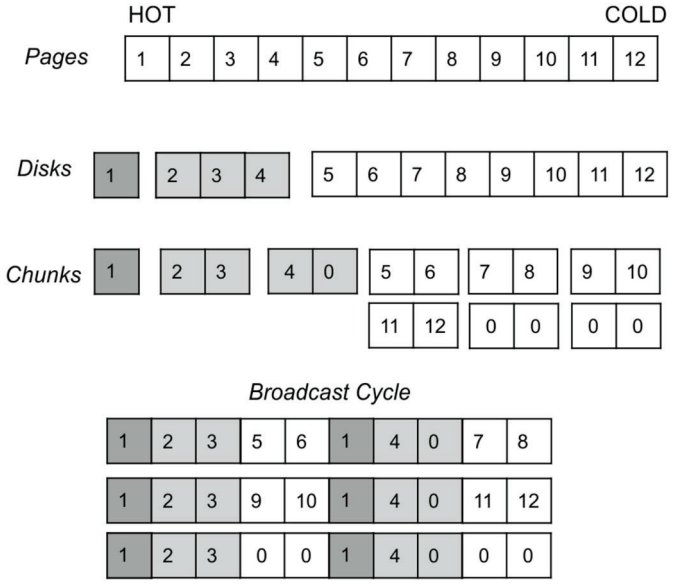

Fig. 1. Broadcast program production in Broadcast Disks. Different disks are in different colors. Null pages (marked 0) can be seen in the generated broadcast schedules.

B. Vaidya and Hameed's Proposed Scheduling Method

Another popular algorithm for constructing schedules in push-based systems was proposed in [1]. This method is based on the so-called "square root rule" which gives a bound to the achievable lowest mean average delay for the clients. Results from [1] imply that, for optimal performance, instances of an item i should be equally spaced with spacing s_i, such that

$$\frac{s_i^2 p_i}{l_i} = constant, \qquad 1 \le i \le N \tag{1}$$

where p_i is the demand probability, l_i is the length for the data item i and N is the total number of data items that are subject to broadcasting. The optimal overall mean access time, name $t_{optimal}$, can be calculated assuming that instances of each data item are equally spaced and is obtained as

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

Keeping in mind that the assumption instances of each data item to be equally spaced, cannot always be realized, $t_{optimal}$ represents a lower bound on the achievable overall mean access time. This lower bound is in general not achievable.

The resultant equation has been used in the scheduling algorithm in [1] to decide which item to broadcast at a given time.

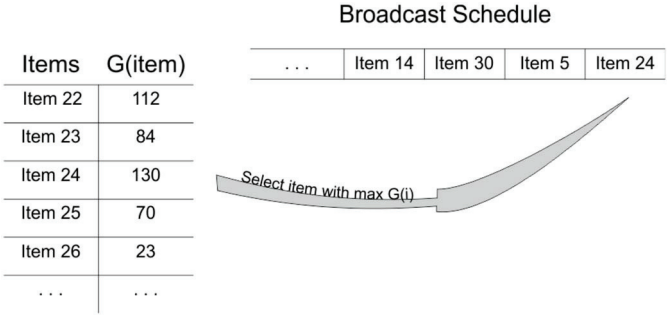

Fig. 2. Performing an item selection via the method in [1].

Vaidya and Hameed [1] calculate the following function, in order to choose which data item to broadcast

$$G(i) = \frac{(Q - r(i))^2 p_i}{l_i}, \qquad 1 \leq i \leq N \tag{3}$$

where $Q - r(j)$ is the spacing between the current time and the time at which item j was previously transmitted. The scheduling algorithm works by calculating the G function for all data items and then selecting to broadcast the item with the maximum value of G. An example is shown in Fig. 2. When there is more than one item with maximum value, it chooses any one of them arbitrarily. This randomness in the algorithm will generate a small deviation in the average spacing of consecutive broadcasts of a data item; nevertheless even small deviations drives the system away from optimal performance.

Vaidya and Hameed [1] generates a schedule with average delay near to the theoretical lowest possible. It is also fully automated; it doesn't need any fine tuning or outside aid in order to generate the broadcast schedule. However this method suffers from some drawbacks too. The greatest drawback is its large computational complexity. Since the algorithm needs to calculate the maximum value of G for N data items in every step of the generation of the broadcast schedule, the average complexity to broadcast just one data item is $\mathcal{O}(\mathcal{N})$. This decreases its usability, though improvements have been suggested in [1] that minimize computational complexity with trade-off in accuracy of the method, with consequent a decrease in performance. Moreover, there is randomness inserted in the generation of the broadcast. Due to this fact, this method cannot create deterministic schedules completely, as the above-mentioned randomness can change the schedule at each period. This causes the increase of the lowest achievable mean access time, which is essentially a decrease in performance and can be explained via the Bus Stop Paradox mentioned earlier in this Section.

C. Research Related to Computational Complexity Problems

Pull-based systems are naturally more complex than push-based systems. Usually the real world applications that choose the push-based broadcasting architecture, do so in order to achieve minimal cost. However, the state of the art algorithm proposed in [1] has a forbidding $\mathcal{O}(\mathcal{N})$ time complexity. The method proposed in [5] attacks this problem of high computational cost, while achieves near optimal performance. However, the purpose of [5] is to decrease complexity in the case of large data items populations, as in such cases the method in [1] suffers from a high time complexity. Our method is an alternative to [1] and performs similar to [1] in terms of mean access time, while having lower computational demands, also in environments with small data item populations, in which cases [5] has no complexity benefit over [1].

Polatoglou *et al.* [8] investigates that problem of the computational complexity for the estimation of the data item demand probabilities in an adaptive system [6], [7]. Their system uses the bucketing scheme from [1].

Another interesting topic of push-based systems is the minimum energy broadcasting problem in multihop networks, which is investigated in [9] and [10]. Centralized optimal and near optimal algorithms are proposed with the former having $\mathcal{O}(N^2)$ complexity and the latter $\mathcal{O}(\mathcal{N})$. Ataei *et al.* [9] also proposed a distributed algorithm of negligible complexity. References [11]–[17] propose and investigate algorithms that exploit locality of client demands, while simultaneously conserve user equipment energy. The main interest of [11]–[13] is the trade-offs between energy conservation and access latency performance. Nicopolitidis [18] investigates performance fairness in wireless data broadcasting, when multiple applications with different demand pattern receive content from a Broadcast Server. An adaptive wireless push system is proposed, which is able to fairly allocate performance to the multiple applications. Those methods are not comparable to our proposed scheme since they investigate problems related to the optimal transmission ranges for the wireless nodes, while we propose a new broadcast schedule generation algorithm for the Base Station.

III. Proposed Scheduling Algorithm

In our proposed method we combine knowledge gained from both [1] and [2], while carefully trying to avoid the drawbacks of each method. Specifically, we use the facts that a) the broadcast schedule should be kept periodically and b) the spacing between inter-arrivals of a data item should be constant. To accomplish those goals we propose a new scheduling algorithm, that uses a specific structure, called Broadcast Levels, which will be described in this Section.

A. Preliminaries

To construct Broadcast Levels we use the square root rule from [1] and the equations that derive as a consequence (Equation 1 and 2). In order to achieve a low overall mean access time, as described in Equation 2, to the extent possible, instances of each data item must be equally spaced and Equation 1 must hold true.

Considering that s_1 and s_N are the spacing of the least demanded and the most demanded data items respectively, then

$$s_1 = max(s_i), \qquad s_N = min(s_i) \tag{4}$$

and as result of Equation 1

$$\frac{s_1^2 p_1}{l_1} = \frac{s_i^2 p_i}{l_i} \Leftrightarrow x_i = \frac{s_1}{s_i} = \sqrt{\frac{p_i l_1}{p_1 l_i}}, \qquad 1 \leq i \leq M \tag{5}$$

In Equation 5, we call the relation $\frac{s_1}{s_i}$ as x_i for simplicity. The value of x_i can be physically interpreted as the number of times we have to transmit $item_i$ for every occurrence of the least demanded data item ($item_1$) in a period of the broadcast schedule.

If we broadcast all the data items periodically, then each consequent iteration of data item 1 denotes the end of a major cycle of the broadcast, while each broadcast of data item N denotes the end of a minor cycle. Thus a minor cycle of the broadcast occurs every time the most important data item is broadcast and a major cycle occurs when the least important data item is broadcast, since all the other data items have already been added in the broadcast schedule at least once. With the equal-spacing assumption, the quantity $\frac{l_i}{s_i}$ is the fraction of the bandwidth allocated to item i. Therefore, $\sum_{i=1}^{n} \frac{l_i}{s_i} = 1$.

The lengths of the major and minor cycles of the broadcast schedule are calculated as in Equation 6 and 7 respectively.

$$\sum_{i=1}^{N} x_i l_i = s_1 \sum_{i=1}^{N} \frac{l_i}{s_i} = s_1 \tag{6}$$

$$\frac{1}{x_N} \sum_{i=1}^{N} x_i l_i = s_N \tag{7}$$

B. Broadcast Scheduling Algorithm

Our proposed method exploits Equation 7. We group data items, depending on their x_i value, in structures called levels. These data item containers are similar to the Broadcast Disks in [2] and buckets in [1]. Then we traverse the generated structure in order to construct a minor cycle each time. This structure offers near to the optimal performance.

In order to place the data items in levels, we must first calculate the x_i value for each data item, using Equation 5. Then we group data items with similar values of x_i to create the levels. Each level needs two variables in order to be used in our proposed method. The *Index* pointer variable is an essential part of the level structure, since we need a variable to remember which was the item of the level we broadcast last. We always traverse each level from its beginning to its end, so when the pointer reaches the end of the level we should put it back to the beginning. Finally, *Remainder* is a variable that stores the remainder of the division used in the algorithm. Algorithm 1, shown below, describes in pseudocode the algorithm for scheduling levels generation.

When grouping data items to create levels, we face a trade-off between performance and computational time complexity. We could use any number of levels for specific data items population and we could choose loose or tight intervals in assignment of x_i values to levels. An implementation could even theoretically use uneven intervals in order to minimize the number of levels used. Furthermore, there is no actual degradation in the performance of the broadcast schedule in the case of a broadcast program with empty levels. On the contrary, the performance of a broadcast schedule would benefit to a point for a larger number of levels, because that way we increase the accuracy of our method. However, the time complexity induced by the proposed method depends on the number of levels used. In the worst case scenario, where the broadcast scheduler has to traverse the whole level structure before selecting a data item for broadcast, the time cost of our method would be $\mathcal{O}(\mathcal{N})$. On the contrary, in a more suitable situation, where all levels select data items for broadcast during each traversal, our method could perform as good as $\mathcal{O}(1)$.

In a dynamic method to create levels, we could introduce an accuracy parameter which dictates the maximum difference on x_i's between items of the same level. In our performance evaluation we used such a method, where *accuracy* is a variable that describes the levels structure. It denotes the percentage of how much larger the x_i of all items in a specific level can be compared to the x_i of least demanded in that level. For example, if we order the data items based on their x_i value from the least to the most demanded, then the first level will contain the least demanded item x_1 and all data items with $x_1 \leq x_i \leq x_1 * (accuracy + 1)$. For the next items we create a

Algorithm 1 Generate Scheduling Levels Structure Algorithm

Require: data items i in ascending order
1: **for all** items i **do**
2: $x_i = \sqrt{\frac{p_i l_1}{p_1 l_i}}$
3: **end for**
4: list $items_0 \leftarrow []$ ▷ List of the items level j contains
5: $remainder_0 \leftarrow 0$ ▷ Remainder parameter of level j
6: $index_0 \leftarrow 0$ ▷ Index parameter of level j
7: $X_0 \leftarrow min(x_i)$ ▷ X parameter of level j
8: $j \leftarrow 0$
9: **for all** items i **do**
10: **if** $x_i > X_j * (accuracy + 1)$ **then**
11: $j \leftarrow j + 1$
12: initialize new level j as in lines 4 to 6
13: $X_j \leftarrow x_i$
14: **end if**
15: add i to $items_j$
16: **end for**

Algorithm 2 Broadcast Scheduling Algorithm

Require: Broadcast levels structure
1: $j \leftarrow 0$
2: **while** broadcasting **do**
3: $M = \frac{X_j * length(items_j)}{X_{max}} + remainder_j$
4: **while** $M > 0$ **do**
5: **if** $length(items_j[index_j]) < M$ **then**
6: broadcast $items_j[index_j]$
7: $M \leftarrow M - length(items_j[index_j]$
8: $index_j \leftarrow (index_j + 1) \mod length(items_j)$
9: **else**
10: $remainder_j \leftarrow M$
11: $M \leftarrow 0$ ▷ Move to the next level
12: **end if**
13: **end while**
14: $j \leftarrow (j + 1) \mod$ number_of_levels
15: **end while**

new level and repeat this process until all the data items are processed.

Depending on the value of the *accuracy* parameter our method will produce 1 to N levels. For bigger values of *accuracy* we assign more data items to each level since we select data items from a wider set of x_i's. Respectively, smaller values of *accuracy* lead to more and smaller levels.

There is a value for *accuracy* when we create a level for each data item. This value must be small, so there are no items with close x_i values to populate the same level. From that point onward, even if we use smaller *accuracy* values the generated broadcast schedule will remain the same. Hence, the performance and the computational cost remain constant. Each level is characterized by the parameter X_i, which is the average of the x_i's of all the items that level contains.

After the construction of the levels structure we generate the broadcast schedule by broadcasting data items from each level. Algorithm 2, shown in the previous page, describes in pseudocode the algorithm for generating the schedule. Our algorithm traverses the levels in a round-robin manner, starting every time from the fastest and ending to the slowest level. If one follows the steps taken above, then the levels should already be in a sorted order based on their X_i parameter. In fact, the order of level traversing doesn't affect performance, should the same route is being used in every traversal of the structure. Each traverse from top to bottom creates a minor cycle. The broadcast completes one period when we have broadcast all data items at least once.

Every time we visit a level i, we choose to broadcast M items from there based on the calculation:

$$M = \frac{X_i \cdot level_i}{X_{max}} + remainder_i \tag{8}$$

where $level_i$ is the sum of the lengths of the data items in that level and X_{max} is the X_i parameter of the most important level. The quotient of this division M is the total length of the data items we should broadcast at that specific time from level i. We cyclically traverse the level broadcasting data items until we select items with lengths summing to M or less. In the case we cannot completely satisfy M, we compensate by using the difference of M minus the sum of lengths in the next traversal of that level by storing that value in the $remainder_i$ parameter.

By using this simple setup we ensure that we broadcast data items accordingly to the square root rule while keeping the spacing between two consecutive broadcasts of a data item fixed. To this end, results very near the theoretical best are achieved.

C. Computational Complexity Benefits

The proposed algorithm is less computationally complex than algorithm in [1]. In [1] the time complexity to broadcast just one data item is $\mathcal{O}(\mathcal{N})$. Vaidya and Hameed [1] also describe a scheduling algorithm with bucketing, but even then complexity remains $\mathcal{O}(l)$, where l is the number of buckets used.

In the proposed scheme we have to create the scheduling levels structure first, an $\mathcal{O}(\mathcal{N})$ task only run the first time. Then, we need to traverse the levels. Going from one level to another is a task that has a constant computational cost. Each time we traverse a specific level we can broadcast multiple or no data items. The only operations needed to be done are to calculate once Equation (3) and to assign values to two variables, the *Remainder* and *Index* of that level. All of those operations are of constant time. Thus if we broadcast a data item from each level every time we traverse it, then time complexity is $\mathcal{O}(1)$. In an extremely unfavorable environment, our method could reach time complexity of $\mathcal{O}(\mathcal{N})$. However, in a realistic environment, our method behaves exceptionally well.

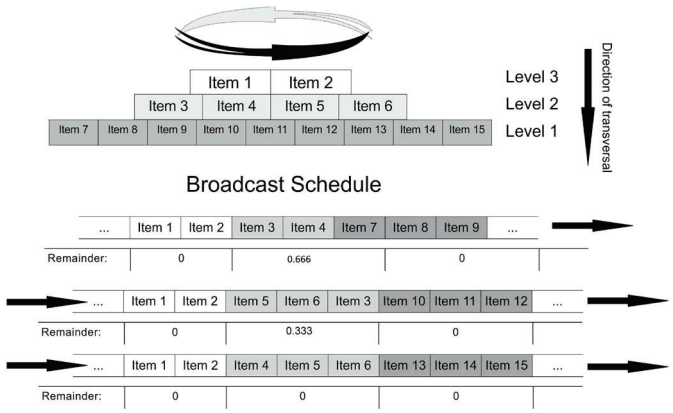

Fig. 3. Example of the construction of the broadcast using the proposed method. Levels structure and a period of the broadcast schedule can be seen.

An extreme example where the difference in time complexity of the two methods can be understood is the case of a uniform distribution for the data items demand probabilities. The time complexity to broadcast one date item in [1] is $\mathcal{O}(\mathcal{N})$, as in any other scenario. However, in our proposed method only one level will be generated and all data items will be broadcasted in each traversal, thus our method has a constant ($\mathcal{O}(1)$) time complexity cost. In Section V, we further investigate our method's time complexity behavior.

D. Construction of the Broadcast Schedule Example

In order to give more insights to the proposed method Fig. 3 presents an example of constructing a period of the broadcast schedule when the number of items are 15. In this simplistic example the values of x_i for items 1 and 2 are 3, for items 3-6 x_i is 2 and 1 for items 7-15. All data items have length of one unit.

By using Equation 8 we plan the broadcasting schedule. In every traversal of level 3 we broadcast it whole. In the case of level 1 we cyclically transmit one third of the items, and use the *Index* parameter to keep track of the last item sent. However, in level 2 we must use the *remainder* parameter in order to have a just broadcast. That way, by traversing the levels we generate the broadcast schedule.

E. Effect of Transmission Errors

The algorithms presented in Section III do not take into account transmission errors. In the discussion so far, we assumed that each data item broadcast by the server is received

without errors by each client. However, wireless communications are characterized by the presence of errors. In this Section, we modify our proposed method to create broadcast schedules in the presence of transmission errors.

In an environment subject to failures, errors may be introduced during the transmission from the server to the client. In order to detect and correct those errors, data is transmitted using an error detection and correction encoding (ECC). Such ECC techniques are able to detect errors and reconstruct the original data in many cases. However, ECC cannot correct large number of errors in the data. In an asymmetric environment, the message received with such uncorrectable errors will be discarded and the client will have to wait for the next broadcast of the data item.

Assuming that $E(l)$ is the probability an item with length l will be discarded due to uncorrectable errors and l_i is the length of data item i after encoding with an error control code, Equation 5 should be written as [1]:

$$\frac{s_1^2 p_1}{l_1} \cdot \left(\frac{1+E(l_1)}{1-E(l_1)}\right)^2 = \frac{s_i^2 p_i}{l_i} \cdot \left(\frac{1+E(l_i)}{1-E(l_i)}\right)^2 \qquad (9)$$

and thus the x_i for each data item should be calculated as follows:

$$x_i = \frac{s_1}{s_i} = \sqrt{\frac{p_i l_1}{p_1 l_i}} \cdot \frac{\frac{1+E(l_i)}{1-E(l_i)}}{\frac{1+E(l_1)}{1-E(l_1)}}, \qquad 1 \le i \le M \qquad (10)$$

Our proposed method can be modified in order to generate a broadcast schedule in an erroneous environment by only adjusting the calculation of the x_i values in line 2 of Algorithm 1 accordingly to Equation 10.

F. Dynamic Data Item Demand Probabilities

In the discussion so far we assumed that the data items' demand probabilities are static and known a priori. However, the most realistic scenario for a broadcasting server is that there is no such knowledge a priori and furthermore the demand probabilities are not static in nature.

By using Learning Automata [20] methods we modify our proposed method in order to generate a broadcast schedule in a dynamic environment. Polatoglou *et al.* [8] propose such a system, with low complexity for the estimation process for the items demand probabilities. However, the overall mean access time in the proposed method in [8] depends on the bucketing scheme of [1]. The performance of the bucketing scheme in [1] is worse than that of the proposed algorithm in [1] and while our proposed method's performance is comparable to the latter, Polatoglou *et al.* [8] cannot be directly compared to our method.

In the dynamic version of our algorithm the broadcast server is equipped with a Learning Automaton like the one in [8] that contains the server's estimate of the actual demand probabilities for each data item i among the set of the items the server broadcasts.

In the beginning, we assume that all data items have equal demand probability estimates. Thus, before the execution of Algorithm 1, we initialize all the data items demand probabilities to $p_i = 1/N$, where N is the number of data items. Then we use Algorithm 1 to generate an initial broadcast levels structure. However, instead of using the x_i as in the proposed scheme, we use their normalized values by dividing all x_i with the $max(x_i)$. Thus, those normalized x_i values are in the interval (0, 1]. Those initial values are also stored in the list *known_x*, which will be used in the Learning Automaton.

In order to choose which data items to broadcast, Algorithm 2 is used as in the proposed scheme. After the transmission of item i, the broadcast server awaits for an acknowledging pulse from every client that was waiting item i. The aggregate received pulse power is used at the server to update the values of $known\_x_i$ [18]. As $b(k)$ we represent the environment response and is calculated by the sum of the received feedback pulses after the server's kth transmission. After normalization the value of $b(k)$ lies in the interval [0, 1]. After each transmission k the server updates the value of transmitted item i, $known\_x_i$ as:

$$known\_x_i = known\_x_i + L \cdot b(k) \qquad (11)$$

where L is a parameter in interval $(0, 1)$ that describes the characteristics of the Automaton. Values for L near 0 make the Automaton more stable, but slow in its convergence, while values near 1 make the Automaton less stable and fast.

After a number of server transmission (for example 2000 transmissions), we update the Broadcast Levels structure. First, we normalize the $known\_x_i$ values by dividing all x_i with the $max(x_i)$. Then, we modify the Broadcast Level structure by moving data item between levels, and creating new levels if needed as is described in Algorithm 3.

This adaptive mechanism is evaluated in the results section and its performance is shown to be close to [1] using an SL_{r-i} Learning Automaton [20]. An SL_{r-i} Learning Automaton has the drawback that it needs to update the demand probability estimate for all data items for every transmission, so it has $\mathcal{O}(N)$ time complexity. The proposed method's performance is almost identical. Furthermore, we need to pay the high time complexity of Algorithm 3 only once every few thousand transmissions.

G. Multiple Broadcast Channels

Our proposed method and the algorithms presented so far assume that the server is broadcasting data items over a single channel and all the clients are tuned to this channel. However, it is not uncommon to use multiple channels in wireless communication systems. Many different setups of multiple channel broadcasting systems can be conceived depending on the desired system demands and characteristics. One possible scenario though is that some clients listen to only one channel. In the case that channel does not broadcast the whole data items' population a situation could arise called "starvation", effectively meaning infinite access time for some requests.

In order to avoid a starvation our broadcast schedule algorithm has to ensure that all data items are broadcast at least once from every channel. In this Section we propose an algorithm that will generate that multichannel broadcast schedule for n channels.

Our proposed algorithms 1 and 2 produce a deterministic broadcast schedule with a major cycle calculated in Equation 6

Algorithm 3 Learning Automaton Update Function

Require: Broadcast Levels structure, normalized $known\_x_i$

1: **for all** items i in Broadcast Levels structure **do**
2: **if** item i stays in its initial level **then**
3: do nothing
4: continue with the next item
5: **end if**
6: $j \leftarrow$ the X value of the level item i resides
7: **while** choosing level **do**
8: **if** $known\_x_i > X_j * (accuracy + 1)$ **then**
9: $j \leftarrow j + 1$ ▷ Move up a level
10: if level j doesn't exist create it with $X_j = X_j * (accuracy + 1)$
11: **else if** $known\_x_i < X_j$ **then**
12: $j \leftarrow j - 1$ ▷ Move down a level
13: if level j doesn't exist create it wit $X_j = X_j/(accuracy + 1)$
14: **else** ▷ Place item i here
15: remove item from its initial level
16: add item to level j just before the *Index* pointer of level j
17: **end if**
18: **end while**
19: **end for**

Algorithm 4 Multichannel Broadcast Scheduling Algorithm

Require: Broadcast Levels structure, n number of channels, M major cycle length

1: $M \leftarrow 0$ ▷ Major cycle variable
2: **for all** levels $level$ in Broadcast Levels structure **do**
3: $M \leftarrow M + length(level) \cdot X_{level}$
4: **end for**
5: $channel\_distance \leftarrow M/n$
6: **while** $n > 0$ **do**
7: Simulate Broadcasting from the Levels structure until items with $channel\_distance$ total length are transmitted
8: Copy the Broadcast Levels structure for channel n
9: $n \leftarrow n - 1$
10: **end while**
11: In each channel start generation from its saved level structure

and a minor cycle calculated in Equation 7. Our goal is to satisfy the demand for constant spacing between inter-arrivals of a data item. To achieve this goal we n equally spaced points in the major cycle, where n is the number of channels to use, and each channel starts to generate the broadcast schedule from that point. Algorithm 4 describes this process.

In lines 2-4 of Algorithm 4, we present an alternative method of calculating the major cycle length of the broadcast schedule. The Learning Automaton scheme we proposed in Section III-F can be modified for multiple channel broadcasting. In that scheme, Algorithm 4 should be used when we update the demand probabilities estimates and consequently change the broadcast level structure.

IV. Performance Evaluation

In order to assess the performance of our approach a simulation environment was implemented to test our proposed algorithm versus the method in [1]. We didn't run any tests versus the method in [2] since [1] is known to be the most efficient approach, as it produces near-optimal mean access time.

In our simulation, clients are not equipped with any cache memory (as in [1]). For the first three experiments we consider that the data access pattern of the clients is static, meaning that it always remains the same. In the last experiment, the server has no a priori knowledge of the data item demand probabilities and estimates those probabilities using a Learning Automaton. All experiments have been conducted considering that data items have lengths that follow a uniform distribution, with minimum length of 1 and maximum of 10. The data item access probabilities are known to the Broadcast Server and they do not change. Comparison of the proposed approach towards that of [1] is made in terms of overall client mean access time, which, as explained earlier, is the average time a client waits for a requested data item to arrive from the broadcast channel. In all cases, multiple experiments were conducted and the confidence interval of the results is 95% for standard deviation of less than 0.1%.

The simulation environment investigates a network where the Broadcast Server decides which data item to schedule based on its broadcasting algorithm. In all experiments we conducted the simulation ended when the server satisfied 5.000.000 data item requests. There was always one server and 100 clients. Clients demand data items with access patterns defined by the Zipf distribution, used in other relevant papers as well [1]–[10], [14]–[16], [18].

The Zipf distribution is governed by a parameter θ: $\theta > 0$, known as the access skew coefficient, and defines the demand probability for item i as follows [19]:

$$p_i = \frac{1}{H_{(N,\theta)} \cdot i^{\theta}}, \quad i \geq 1 \tag{12}$$

Where:

$$H_{(N,\theta)} = \sum_{i=1}^{N} \frac{1}{i^{\theta}} \tag{13}$$

is the Nth generated harmonic number of rank θ.

For $\theta \rightarrow 0$ (the *theta* parameter of the distribution), the produced distribution verges on the uniform distribution, thus clients tend to demand a larger portion of the data items subject to broadcasting. While $\theta \rightarrow \infty$ the distribution is increasingly skewed, thus the clients tend to demand only a small subset of the data items.

We compared our proposed method against that of [1] in terms of average delay experienced by the clients. Comparison is made for various values of θ, N and *accuracy* parameters.

For varying θ, conducted experiments examine cases where the number N of items the Broadcast Server has to broadcast is 100, 500 and 1.000, in Figs. 4–6 respectively. In Fig. 8, comparison is made for varying *accuracy* with $\theta = 1$ and

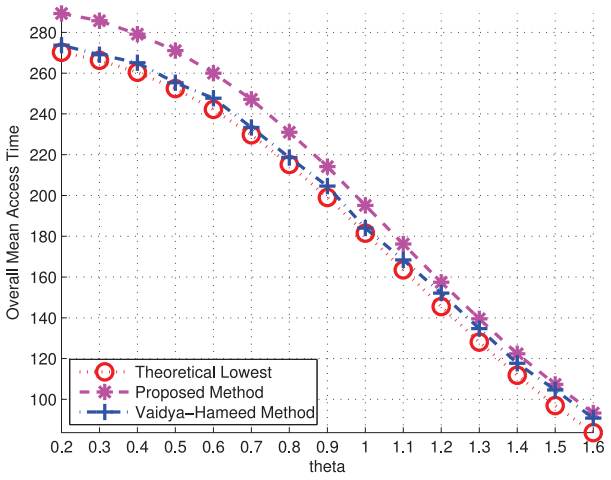

Fig. 4. Performance for varying theta parameter, for a database size of 100 data items in the Broadcast Server. The value of *accuracy* is set to 0.05.

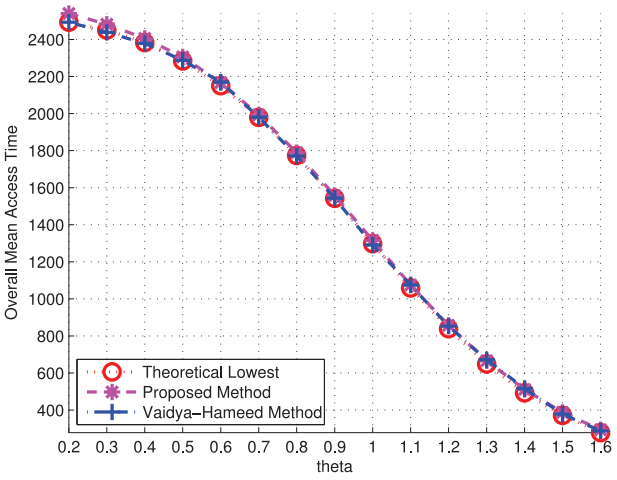

Fig. 6. Performance for varying theta parameter, for a database size of 1000 data items in the Broadcast Server. The value of *accuracy* is set to 0.05.

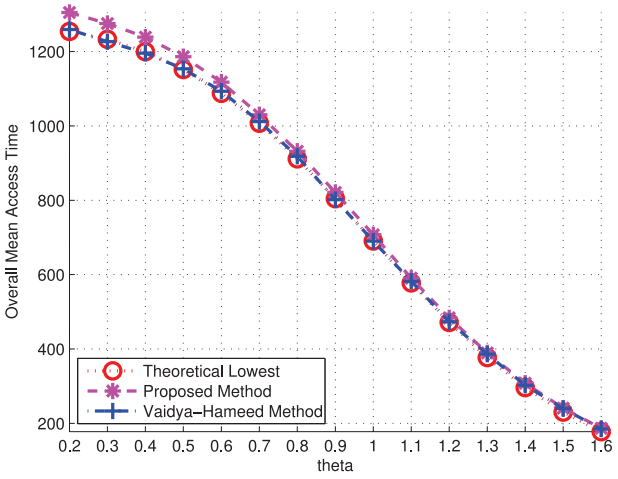

Fig. 5. Performance for varying theta parameter, for a database size of 500 data items in the Broadcast Server. The value of *accuracy* is set to 0.05.

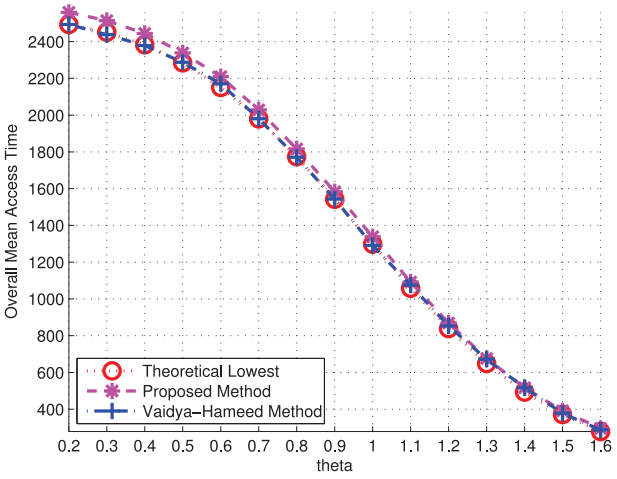

Fig. 7. Performance for varying theta parameter, for a database size of 1000 data items in the Broadcast Server. The value of *accuracy* is set to 0.5.

$N = 500$. The main conclusions that can be drawn from these Figures are discussed below.

A. Comparison for Varying Demand Skewness (θ Parameter)

In this experiment group, for values of θ parameter ranging from 0.2 to 1.6, we evaluate the network under examination for a population of 100, 500 and 1.000 data items and the results are shown in Figs. 4–6 respectively. Values for θ near 0 mean a practically uniformly distributed probability distribution function (pdf) for the demand of clients for data items, whereas values of θ near 1.6 mean a greatly skewed pdf for the item demands. For each plot we used the same data items lengths, in order to keep consistency in our results. The value of the *accuracy* for all experiments is set to 0.05.

All three experiments produce similar plots which demonstrate the performance of the proposed method compared to [1], irrespective of the number of data items that are subject to broadcasting. The average delay for the broadcast schedule generated from our method is less than 2% worse than the average delay that Vaidya-Hameed's method produces in the same scenarios. The results in Fig. 4 may seem like an exception, however the performance difference falsely shows bigger only because of the scale of the axes, which is different across Figs. 4–6.

B. Comparison for Varying Accuracy Parameter

In order to investigate the impact of *accuracy* parameter on performance, we rerun the experiment presented in Fig. 6, only this time the value of the parameter *accuracy* is increased to 0.5. The results are shown in Fig. 7. The performance this time is only slightly worse than before; with an *accuracy* of 0.5 the proposed method is less than 2% worse than when using an *accuracy* of 0.05. However, there is a great benefit in the average number of level hops needed for a broadcast with this decreased accuracy. At *theta* $= 1.6$ and with *accuracy* $= 0.05$ our results showed that our algorithm traversed 3.5 levels at average before a data item is selected for broadcast, while with *accuracy* $= 0.5$ our algorithm selected almost 1 item per level hop.

The average number of level hops is described as the average level traversals for a level to successfully broadcast any

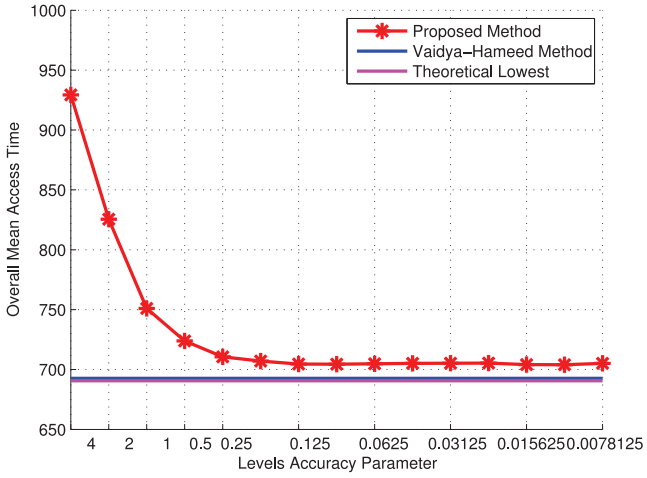

Fig. 8. Performance for varying levels accuracy parameter, for a database size of 500 and theta parameter equal to 1.

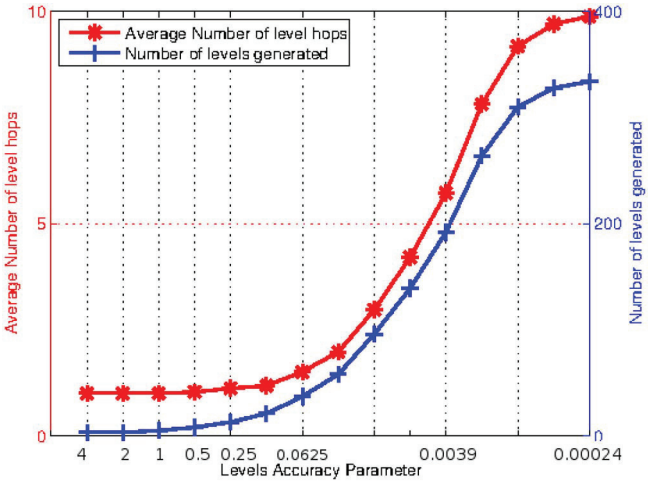

Fig. 9. Number of generated levels and average number of level hops before broadcast, for a database size of 500 and theta parameter equal to 1.

number of data items. This metric is similar to the average complexity of the proposed algorithm. Thus, an average number of level hops equal to 3.5 can be described as $\Theta(3.5)$ time complexity for our method.

In the next experiment we evaluate the network performance for a fixed value of $\theta = 1$ and $N = 500$. We selected 15 values for *accuracy* parameter starting from 4 and ending to 0.000244 by reducing by half in each step.

To assess the impact of *accuracy* on time complexity, in Fig. 8 we demonstrate performance results for choosing small values for *accuracy* and in Fig. 9 the number of levels generated and the average number of level hops between broadcasts. As can be seen in Fig. 8, for values of *accuracy* parameter lower than or equal to 0.125, the benefit in performance is non-existent for our approach. At the same time, Fig. 9 shows that the number of generated levels and the average number of level hops increases rapidly from that point onward. Nevertheless, even for the extreme case of a very small value of *accuracy* $= 0.0078125$, the average number of levels hops is equal to approximately 10. Thus, our method has an $\Theta(10)$ time complexity in that extreme case. Of course, it can be seen

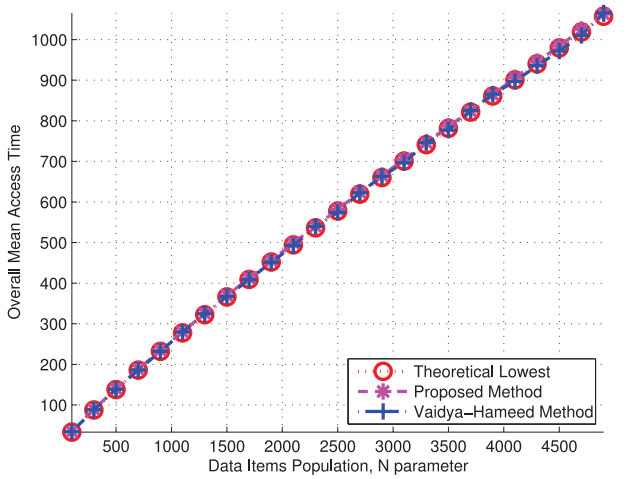

Fig. 10. Performance for varying number of data items (N) parameter, for theta parameter equal to 1.

from Fig. 7, that for useful larger values of *accuracy* that produce meaningful benefits in terms of mean access time (e.g., larger than 0.25 as can be seen from Fig. 8), the time complexity of our approach is even smaller and approaches $\Theta(1)$. This constitutes a significant benefit towards [1], which is required to calculate N times Equation 3 in order to select an item for broadcast.

C. Comparison for Varying the Number of Data Items Broadcast by the Broadcast Server

In this experiment we assess the system performance for a fixed value of $\theta = 1$ and for a varying number of data items in the Broadcast Server. The *accuracy* parameter is set to 0.05. The results are shown in Fig. 10. The linear relationship between average delay and number of data items is obvious in Fig. 10, which again demonstrates the almost identical performance in terms of mean access time of the proposed approach compared to that of [1].

Overall, from Figs. 4–8, it can be observed that the performance of [1] and our method are both very close to the lowest possible achievable average delay. In general the average delay produced by [1] is within 2% of the theoretical lowest, while that of our method is good within 4%. The great benefit of our proposed method is the low complexity versus the $\mathcal{O}(\mathcal{N})$ for method [1]. That compromise between the degradation of performance and the low complexity of our method could be justified in many real world applications.

Finally, a comparison of our approach in terms of that of Vaidya and Hameed [1] when the latter utilizes the bucketing scheme should be made. By using l buckets the computational complexity in [1] reduces from $\mathcal{O}(\mathcal{N})$ to $\mathcal{O}(l)$. Vaidya and Hameed [1] demonstrate performance results of their method versus its bucketing version, where the latter under-performs greatly while parameter θ increases. This is not the case for our method, which demonstrates similar performance for different values of the θ parameter, even with high values for *accuracy* parameter. Simultaneously, our method achieves low computational complexity; even for very low values for the *accuracy* parameter worst time complexity was $\Theta(10)$ and for better selected values of *accuracy* our

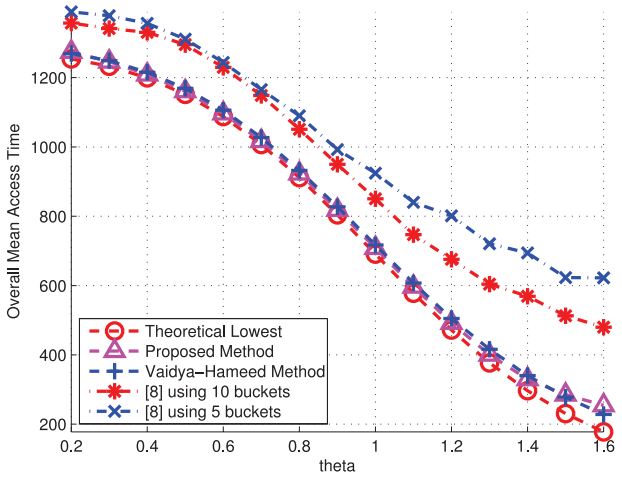

Fig. 11. Performance for varying theta parameter, for a database size of 500. The data item demand probabilities are unknown to the server. The value of *accuracy* is set to 0.05.

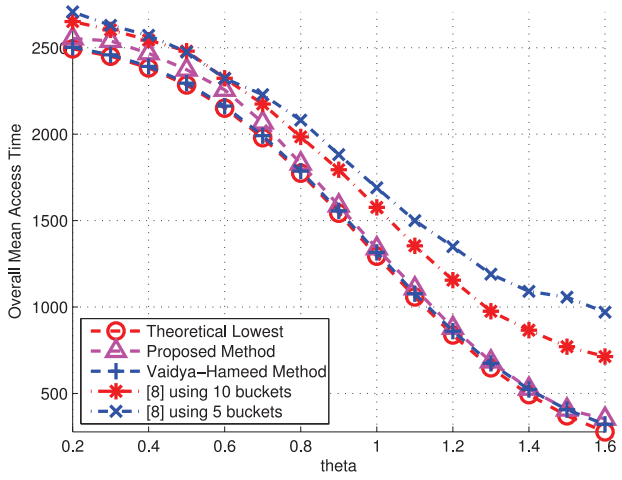

Fig. 12. Performance for varying theta parameter, for a database size of 1000. The data item demand probabilities are unknown to the server. The value of *accuracy* is set to 0.5.

method has time complexity of $\Theta(1)$, much better than the time complexity linear to the number of buckets the bucketing version in [1] achieves.

D. Comparison for Varying Demand Skewness Using Learning Automaton

In this group of experiments we assess the system performance for varying values of θ parameter ranging from 0.2 to 1.6. The data items demand probabilities are not known to the server a priori. Instead, a Learning Automaton is used in all algorithms under investigation. Specifically, we compare the proposed method to both the method in [1] and [8]. In the case of the former a SL_{r-i} Automaton was used. This Automaton is described by high computational cost since it needs to update the estimation of the demand probabilities for each data item, every time the server selects a data item for broadcast. In [8] an algorithm is described with low computational which exploits the bucketing scheme proposed in [1]. However, the performance of that method is bound by the limitations of using the bucketing scheme.

Those methods are compared to our proposed scheme when using the Learning Automaton described in Algorithm 3 with the updating function running every 5000 broadcasts. In Fig. 11 the number of data items is 500 and in Fig. 12 the number of data items is 1000. In the former experiment the *accuracy* parameter for the proposed method is set to 0.05 and in the latter to 0.5. Those values of the *accuracy* parameter lead to a Broadcast Levels structure with an computational complexity less than $\mathcal{O}(5)$, in correspondence to the method in [8] which is described by $\mathcal{O}(5)$ in the case of using 5 buckets and $\mathcal{O}(10)$ when using 10 buckets.

Both the proposed method and the methods in [1] and [8] were allowed to "warm-up" for 1.500.000 data item broadcasts.

As can be seen in Figs. 11 and 12, the proposed scheme and the method in [1] performed almost identically and managed to converge to the true data item demand probabilities. The performance of the method in [8] depends on the number of buckets used.

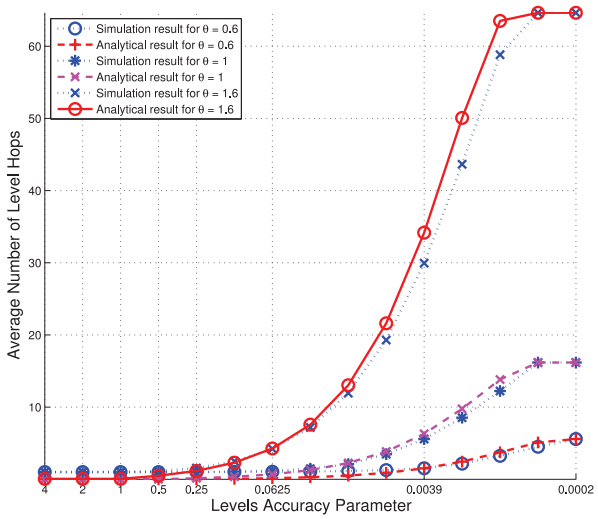

Fig. 13. Broadcast Levels Complexity Calculation through Analysis and Simulation for a population of 1.000 data items. AQ4

V. Study on Broadcast Levels Schedule Construction Time Complexity

The main advantage of the proposed method versus the state of the art algorithm proposed in [1] is the low computational complexity. In this section we present a study on Broadcast Levels average time complexity for data item population following the Zipf distribution as described in Equation 12.

A. Preliminaries

In zipf distribution p_1 is the $max(p_i), \forall i \in [1, N]$ and p_N is the $min(p_i), \forall i \in [1, N]$. Each of the x_i used in the broadcast schedule construction algorithm is given by Equation 5. Since we are interested in the average computational complexity of our method, we calculate the expected values for x_i using the expected data items lengths (l_i parameter).

$$E(x_i) = \frac{s_N}{s_i} = \sqrt{\frac{p_i \cdot E(l_N)}{p_N \cdot E(l_i)}}, \qquad 1 \le i \le M \qquad (14)$$

Since $E(l_N)$ equals to $E(l_i)$:

$$x_i = \sqrt{\frac{p_i}{p_N}} = \sqrt{\frac{\frac{1}{H_{N,\theta} \cdot i^\theta}}{\frac{1}{H_{N,\theta} \cdot N^\theta}}} = \sqrt{\frac{N^\theta}{i^\theta}} \tag{15}$$

Hence from Equation 12 and 15 we calculate minimum and maximum x_i:

$$x_{max} = x_1 = \sqrt{N^\theta}, x_{min} = x_N = 1 \tag{16}$$

The lengths of the major and minor cycles of the broadcast schedule are calculated as in Equation 17 and Equation 18 respectively.

$$s_N = \sum_{i=1}^{N} x_i = s_1 \sum_{i=1}^{N} \frac{1}{s_i} \tag{17}$$

$$s_1 = \frac{1}{x_1} \sum_{i=1}^{N} x_i = \frac{\sqrt{\frac{N^\theta}{1^\theta}} + \sqrt{\frac{N^\theta}{2^\theta}} + \cdots + \sqrt{\frac{N^\theta}{N^\theta}}}{\sqrt{N^\theta}} = \tag{18}$$

$$= \sum \sqrt{\frac{1}{i^\theta}} = H_{N,\frac{\theta}{2}} \tag{19}$$

B. Number of Levels Constructed

The number of levels generated depends on the *accuracy* parameter and the characteristics of the data items demand probabilities distribution.

System parameter $accuracy \in [1, \infty)$, notated here as l, dictates for all levels that

$$l = \frac{x_k}{x_{k+m_k}} = \sqrt{\frac{(k+m_k)^\theta}{k^\theta}} \tag{20}$$

$$m_k = \left(l^{2/\theta} - 1\right) \cdot k \tag{21}$$

where x_k is the most, x_{k+m} is the least popular data item and m_k is the number of data items in the level that starts from item k. If we set $(l^{2/\theta} - 1)$ as a, then Equation 21 can be rewritten as $m_k = a \cdot k$.

The first data item of level i function, notated as $lev(i)$, is a recursive one with $lev(1) = N$ and:

$$lev(i+1) = lev(i) - m_{lev(i)} = lev(i) \cdot (1-a) \tag{22}$$

and the last data item in level i is calculated as

$$lev\_last(i) = N \cdot (1-a)^i \tag{23}$$

Equation 23 can calculate levels only while $lev\_last(i-1) - lev\_last(i) \geq 1$. Solving for the equality and i variable yields:

$$i = \log_{1-a} \frac{1}{Na} + 1 \tag{24}$$

As a result the total number of levels is i as calculated in Equation 23 plus the remaining data items that create a level each and is always less than N, the population of data items.

$$Number\_of\_Levels = i + N \cdot (1-a)^i \tag{25}$$

C. Average Time Complexity

Every time a minor cycle of the broadcast schedule is completed, all levels have been traversed exactly once. The average number of level traversals per broadcast can describe the time complexity of the proposed algorithm.

The average complexity of Broadcast Levels is a function of the total number of levels divided by the number of items broadcast in a minor cycle:

$$T(N) = \frac{Number\_of\_Levels}{H_{N,\frac{\theta}{2}}} \tag{26}$$

This analytical function for time complexity is a good estimate that agrees with the results from the experiments conducted in Section IV. This can be seen in Fig. 11 where we compare the average number of level hops per broadcast as calculated from both the Equation 26 and the simulation for different θ and *accuracy* values.

VI. Conclusion

The main interest in push-based systems is that they can provide a cheap telecommunication alternative to pull-based architectures. However, the state of the art algorithm for the construction of the broadcast schedule is characterised by its high computational complexity of $\mathcal{O}(\mathcal{N})$. Such complexity is prohibitively high since push-based systems commonly contain a large population of data items. The bucketing scheme in [1] battles this constrain, though degrades the system's performance.

This paper proposes a new scheduling algorithm for wireless push systems, which performs near optimal and has a low computational complexity. The trade-off between performance and time complexity can be controlled through *accuracy* , the single parameter of the system.

Extensive simulation results are presented that reveal our methods excellent performance despite its low computational demands.

References

[1] N. H. Vaidya and S. Hameed, "Scheduling data broadcast in asymmetric communication environments," *ACM/Baltzer Wireless Netw.*, vol. 5, no. 3, pp. 171–182, 1999.

[2] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination-based data delivery using broadcast disks," *IEEE Pers. Commun.*, vol. 2, no. 6, pp. 50–60, Dec. 1995.

[3] E. Tiakas, S. Ougiaroglou, and P. Nicopolitidis, "Efficient algorithms for constructing broadcast programs in asymmetric communication environments," *Telecommun. Syst.*, vol. 21, no. 3, pp. 185–209, 2009.

[4] M. H. Ammar and J. W. Wong, "On the optimality of cyclic transmission in teletext systems," *IEEE Trans. Commun.*, vol. 35, no. 1, pp. 68–73, Jan. 1987.

[5] C. Liaskos, S. Petridou, and G. Papadimitriou, "Towards realizable, low cost broadcast systems for dynamic environments," *IEEE Trans. Netw.*, vol. 19, no. 2, pp. 383–392, Apr. 2011.

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

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

[8] M. Polatoglou, P. Nicopolitidis, and G. I. Papadimitriou, "On low-complexity adaptive wireless push-based data broadcasting," *Int. J. Commun. Syst.*, vol. 27, no. 1, pp. 194–200, 2014.

[9] M. R. Ataei, A. H. Banihashemi, and T. Kunz, "Low-complexity energy-efficient broadcasting in one-dimensional wireless networks," *IEEE Trans. Veh. Technol.*, vol. 61, no. 7, pp. 3276–3282, Sep. 2012.
[10] J. Widmer, C. Fragouli, and J. Y. Le Boudec, "Low-complexity energy-efficient broadcasting in wireless ad-hoc networks using network coding," in *Proc. 1st Workshop Netw. Coding Theory Appl. (NetCod)*, Riva del Garda, Italy, 2005, pp. 1–6.
[11] H. Y. Shin, "Evolved broadcast scheduling mechanism supporting energy conserving e-MBMS transmission," *Wireless Netw.*, vol. 19, no. 7, pp. 1725–1738, 2013.
[12] H. Y. Shin, M. H. Yen, C. C. Tseng, and H. H. Lie, "Fast data access and energy-efficient protocol for wireless data broadcast," *Wireless Commun. Mobile Comput.*, vol. 12, no. 16, pp. 1429–1441, 2012.
[13] H. Y. Shin, F. M. Hsu, K. H. Tsai, and M. H. Yen, "Access popularity based wireless broadcasting mechanism," in *Proc. Int. Conf. Mach. Learn. Cybern. (ICMLC)*, vol. 1. Lanzhou, China, 2014, pp. 65–70.
[14] C. K. Liaskos, S. G. Petridou, and G. I. Papadimitriou, "Cost-aware wireless data broadcasting," *IEEE Trans. Broadcast.*, vol. 56, no. 1, pp. 66–76, Mar. 2010.
[15] C. K. Liaskos, A. Xeros, G. I. Papadimitriou, M. Lestas, and A. Pitsillides, "Broadcast scheduling with multiple concurrent costs," *IEEE Trans. Broadcast.*, vol. 58, no. 2, pp. 178–186, Jun. 2012.
[16] C. K. Liaskos, A. Xeros, G. I. Papadimitriou, M. Lestas, and A. Pitsillides, "Balancing wireless data broadcasting and information hovering for efficient information dissemination," *IEEE Trans. Broadcast.*, vol. 58, no. 1, pp. 66–76, Mar. 2012.
[17] J. Liu, Y. Zhang, and J. Song, "Energy saving multicast mechanism for scalable video service using opportunistic scheduling," *IEEE Trans. Broadcast.*, vol. 60, no. 3, pp. 464–473, Sep. 2014.
[18] P. Nicopolitidis, "Performance fairness across multiple applications in wireless push systems," *Int. J. Commun. Syst.*, vol. 28, no. 1, pp. 161–166, 2014.
[19] L. Devroye, *Non-Uniform Random Variate Generation*. New York, NY, USA: Springer, 1986, pp. 550–551.
[20] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1989, pp. 209–210.

Stathis Mavridopoulos received the B.Sc. degree in computer science from the Department of Informatics, Aristotle University of Thessaloniki, in 2013, where he is currently pursuing the M.Sc. and Ph.D. degrees. His current research interests are in the areas of wireless networks, broadcast systems, and nanoscale communications.

Petros Nicopolitidis received the B.S. and Ph.D. degrees in computer science from the Department of Informatics, Aristotle University of Thessaloniki, Greece, in 1998 and 2002, respectively. From 2004 to 2009, he was a Lecturer with the Department of Informatics, Aristotle University of Thessaloniki, where he currently serves as an Assistant Professor. His research interests are in the areas of wireless networks and mobile communications. He has published over 70 papers in international refereed journals and conferences. He has co-authored the book entitled *Wireless Networks* (Wiley, 2003). Since 2007, he serves as an Associate Editor for the *International Journal of Communication Systems* and the *Security and Communication Networks Journal*, both published by Wiley.

Georgios Papadimitriou received the Diploma and Ph.D. degrees in computer engineering and informatics from the University of Patras in 1989 and 1994, respectively. He is a Professor with the Department of Informatics, Aristotle University, Greece. In 1997, he joined the faculty of the Department of Informatics, Aristotle University, where he currently serves as a Full Professor. His major research interests are wireless networks, optical networks, learning algorithms, and application of learning algorithms in communication networks. He has published 110 papers in peer-reviewed journals (42 in the IEEE Journals) and 120 papers in international conferences. He is the author of three books published by Wiley and an editor of a book published by Kluwer/Springer. He served as an Associate Editor for the IEEE NETWORK, the IEEE TRANSACTIONS ON BROADCASTING, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, the IEEE SENSORS JOURNAL, and the *IEEE Communications Magazine*. He served as the Chair/TPC Chair for four international conferences, a TPC Member for 60 international conferences, and a Reviewer for 36 scientific journals. He has participated in 19 research projects, some of which as a Team Leader or a Coordinator. He serves as an Evaluator for the international and national research and development programs.

Panagiotis Sarigiannidis (S'05–M'07) received the Diploma and Ph.D. degrees in computer science from the Aristotle University of Thessaloniki, Greece, in 2001 and 2007, respectively. He is currently a Lecturer with the University of Western Macedonia, Greece. His research interests include wireless networks, mobile computing, optical networks, and optical switching.