# To Broadcast Push or Not and What?

J. Beaver*, P.K. Chrysanthis*, K. Pruhs*
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
{beaver, panos, kirk}@cs.pitt.edu

V. Liberatore†
Division of Computer Science
Case Western Reserve University
Cleveland, Ohio 44106-7071
{vincenzo.liberatore}@case.edu

## Abstract

*A major problem in mobile web applications as well as the wireless Internet is the scalable delivery of data. The most popular solution for this problem is a hybrid system that uses broadcast push to scalably deliver the most popular data, and reserves broadcast pull for delivery of less popular data. Such a hybrid scheme introduces a variety of data management problems at the broadcast server. In this paper, we examine three of these problems: the push popularity problem, the document classification problem, and the bandwidth division problem. The push popularity problem is to estimate the popularity of the documents in the web site. The document classification problem is to determine which documents should be pushed and which documents must be pulled. The bandwidth division problem is to determine how much of the server bandwidth to devote to pushed documents and how much of the server bandwidth should be reserved for pulled documents. We propose simple and elegant solutions for these problems. We report on experiments with our system that validate our algorithms.*

## 1. Introduction

In the world of mobile computing devices, available communication bandwidth and component energy are two huge and very real constraints. For example, these limitations lead to the major issue of scalability in wireless networks. Simply put, mobile clients are unable to maintain constant communication with the server due to the the high energy cost of communication, frequent disconnections and the limited amount of bandwidth clients have available to communicate with the server.

To solve this scalability problem, wireless networks have adapted the standard of using *broadcast push* as the preferred method of communication between the server and all clients. Broadcast push employs point-to-multipoint communication (*broadcast*) and sends documents from the server to clients in the absence of explicit client requests (*push*) [1, 3, 4, 5, 14, 15, 16]. Broadcast push is scalable in that the addition of new clients does not change the server workload or the client-perceived response time.

Broadcast push can be combined with a backchannel over which feedback and request communication can occur in a *hybrid data dissemination* scheme [10, 1, 4, 14]. In the hybrid scheme, the set of documents is partitioned into two groups: the broadcast push documents and the client request driven pull documents. The former are cyclically and repeatedly transmitted on the broadcast push channel. The latter are delivered on the broadcast pull channel upon client requests. In either case, end-users request documents as usual with any standard request mechanism which forwards them to a client-side server extension (i.e., special client proxy) that handles communication.

Hybrid data dissemination can be evaluated along various performance metrics, and much work has focused on the average server-side delay before a document is transmitted (e.g., [14]). Client-side latency can be minimized by assigning broadcast push to deliver the most popular (*hot*) data and the broadcast pull to deliver less popular (*cold*) data [14]. The resulting hybrid scheme strives for the scalability of broadcast push and avoids clogging a broadcast channel with cold data items [1, 8]. However, the hybrid scheme introduces three inter-related data management problems at the server, and the primary contribution of this paper is an integrated solution for these problems.

In the hybrid scheme, the server must dynamically assign documents either to the broadcast pull or to the broadcast push channels (*document classification*) [14]. Furthermore, the server must also partition dynamically its bandwidth between broadcast pull and broadcast push (*bandwidth division*). Document classification and bandwidth division are inter-related issues because a given bandwidth division determines the performance of a document clas-

*Supported in part by NSF grants ANI-0123705, and ANI-0325353.
†Supported in part by NSF grant ANI-0123929.

Proceedings of the 7th International Conference on Mobile Data Management (MDM'06)
0-7695-2526-1/06 $20.00 © 2006 IEEE
Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 22:28:02 UTC from IEEE Xplore. Restrictions apply.
IEEE COMPUTER SOCIETY

sification choice and, conversely, a given document classification determines a bandwidth split that optimizes performance. In turn, both document classification and bandwidth division depend on the popularity of data items because download latency is smaller when hot items are assigned to broadcast push, cold items to broadcast pull, and the bandwidth is divided appropriately between the two channels. Therefore, the server must estimate the document popularity (*push popularity* problem). The estimation of document popularity is complicated by the fact that no requests are made by clients for broadcast push documents. In particular, if the popularity wanes for a specific document and that document was on the broadcast push, the shift in client interests is not reflected in request logs. In turn, the server would not know that it is time to demote a document to the broadcast pull channel.

In this paper, we give an integrated algorithm for solving simultaneously and to near-optimality the bandwidth division and document classification problems. Our algorithm was empirically evaluated and compared to an existing fixed division scheme [14] and our algorithm generated a document division which results in lower average latency than previous schemes. This also has the benefit of reducing the energy consumption at the mobile clients by reducing required waiting time. The underlying reason is that if document selection is addressed separately from bandwidth division, a certain bandwidth split can be fixed to a level that is suboptimal for a certain assignment of documents to channels. More generally, the performance trade-offs differ quantitatively and qualitatively under the combined scheme. For example, the assignment led to broadcast push latency that is significantly faster than pull latency due to the higher relative popularity of push items over pull documents.

In Section 2 we briefly examine our system model then fully explain our document selection and bandwidth division methods. In Section 3, we report on the experimental validation of our algorithms. In Section 4 we survey some work related to our own. Finally, Section 5 summarizes our observations and contributions.

## 2 Our Broadcast Push Method

The solution we propose for hybrid data dissemination uses the system model which is shown in Figure 1. As the figure shows, this model follows the hybrid, multichannel data dissemination model similar to that in previous work [1, 4, 10, 14]. Our model contains three channels over which communication occurs. The request channel is a low bandwidth channel over which clients can send requests to the server. This channel is mainly unidirectional in that the server will not respond to the client requests with the requested data. Instead, the server will place the client requests on the broadcast pull channel, which will then schedule the requests to be sent out. Additionally, the broadcast push channel will be used to distribute very popular documents to the clients without requiring any direct client requests. The decision of which documents to push out, and which to require client requests for, is handled by our document selection component, which runs the document selection algorithm, which we explain next.

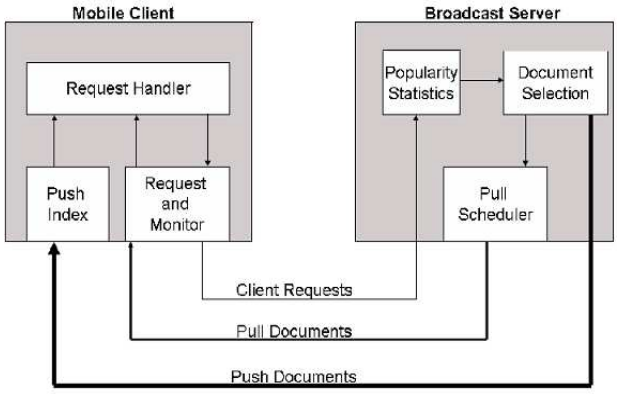


**Figure 1. System model**

| Parameter | Description |
|---|---|
| $n$ | number of documents |
| $\lambda$ | observed request rate $\lambda$ |
| $\alpha$ | pull over-provisioning factor |
| $L$ | current required latency |
| $B$ | total available system bandwidth |
| $S$ | array of document sizes $S_i$ |
| $p$ | array of document probabilities $p_i$ |
| $\epsilon$ | tolerance factor |

**Table 1. Parameters for Algorithms 1 and 2.**

### 2.1 Document Selection and Bandwidth Division Algorithm

Our solution to document classification and bandwidth division is to use an integrated algorithm that minimizes average latency. Algorithm 1 shows this solution which solves the bandwidth division and document selection problems, and works in conjunction with a subroutine shown as Algorithm 2. Algorithm 1 uses a tolerance factor $\epsilon > 0$, which is an arbitrarily small positive number, and finds a solution that has latency within $\epsilon$ of the optimum for the given bandwidth and popularities. The algorithm also assumes that the list of documents passed in is ordered by decreasing popularity (meaning item 1 is the most popular, item $n$ the least popular) and that the list includes the popularity of the items on the push channel, which can be computed using the technique presented in Section 2.2. The parameters for the algorithms are summarized in Table 1.

Proceedings of the 7th International Conference on Mobile Data Management (MDM'06)
0-7695-2526-1/06 $20.00 © 2006 IEEE
Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 22:28:02 UTC from IEEE Xplore. Restrictions apply.
IEEE COMPUTER SOCIETY

Algorithm 1 proceeds in the following manner. It first pre-computes the sums $\sum_{i=1}^{k} \lambda p_i S_i$ (which is a running total of the bandwidth requirements for the first k items) and $\sum_{i=1}^{k} S_i$ (which is a running total of the size of the first k items), placing the totals in the arrays *rspt* and *sizeTotal* respectively (Lines 1–4). This will help to optimize the run time because these values would otherwise have had to be computed during every loop. The algorithm then sets the initial minimum latency to be 0 and maximum latency to be the amount of time required to send all the documents out over individual connections with each client(Lines 5-6). A binary search is then performed, for which each loop consists of taking the average latency between the current minimum and maximum latencies and passing that average latency to the subroutine in Algorithm 2, which will use that average latency to calculate the number of items $k$ that should be placed on the broadcast push channel (recall that the list of items is ordered by popularity, so this $k$ refers to the $k$ most popular documents)(Line 8-9).

**Algorithm 1** Bandwidth Division and Document Classification

**Require:** $n, \lambda, \alpha, B, S, p$, and $\epsilon$ as defined in Table 1, and $p_i \geq p_{i+1}$ $(1 \leq i < n)$

**Ensure:** $k$ is the optimal number of documents on the push channel, pullBW is the optimal pull bandwidth, pushBW is the optimal push bandwidth

1: **for** $i = 1, \ldots, n$ **do**
2: $\text{rspt}_i \leftarrow \text{rspt}_{i-1} + p_i S_i \lambda$
3: $\text{sizeTotal}_i = \text{sizeTotal}_{i-1} + S_i$
4: **end for**
5: $\text{lMax} \leftarrow \text{sizeTotal}_n / B$
6: $\text{lMin} \leftarrow 0$
7: **while** $\text{lMax} - \text{lMin} > \epsilon$ **do**
8: $L \leftarrow (\text{lMax} + \text{lMin})/2$
9: $k \leftarrow \text{tryLatency}(L, p, \lambda, n)$
10: $\text{pullBW} \leftarrow \alpha(\text{rspt}_n - \text{rspt}_k)$
11: $\text{pushBW} \leftarrow B - \text{pullBW}$
12: **if** $\text{pushBW} \geq (\text{sizeTotal}_k/(2L))$ **then**
13: $\text{lMax} \leftarrow L$
14: **else**
15: $\text{lMin} \leftarrow L$
16: **end if**
17: **end while**

**end**

Using the return value of $k$ most popular documents, Algorithm 1 calculates the amount of bandwidth that should be given to the pull channel based on the choice of $k$ (Line 10). Notice that in this calculation, a value $\alpha > 1$ is used that measures the target level of over-provisioning for the pull channel. More precisely, the actual bandwidth we reserve for pull is $\alpha$ times what an idealized estimate predicts. Queuing theory asserts that $\alpha > 1$ guarantees bounded queuing delays, whereas $\alpha \leq 1$ leads to infinite queuing delays. As such, the parameter $\alpha$ can also be thought of as a safety margin for the pull channel.

**Algorithm 2** tryLatency

**Require:** $n, \lambda, L, p$ as defined in Table 1

**Ensure:** Returns the number $k$ of items pushed given that average latency of $L$ is required

1: **while** $\max - \min > 1$ **do**
2: $k \leftarrow (\max + \min)/2$
3: **if** $(p_k \lambda L) > 1/2$ **then**
4: $\min \leftarrow k$
5: **else**
6: $\max \leftarrow k$
7: **end if**
8: **end while**
9: Return $k$

**end**

After calculating the pull bandwidth, the remaining bandwidth is partitioned to the push channel and it is determined whether the amount of push bandwidth provided is actually enough to sustain the amount needed for the push channel (Lines 11–16). If there is enough bandwidth, then the latency could be lowered, and the max latency is decreased and the search performed again. Likewise, if there is not enough bandwidth, the latency is increased and the search performed again. This continues until the $\epsilon$ value is met, at which point the number of items for the push channel (and therefore which items), the push channel bandwidth and the pull channel bandwidth are all returned to be used to divide the bandwidth and documents for the system.

Let us now examine the details of Algorithm 2. Algorithm 2 requires as input a latency along with the request rate($\lambda$), number of documents ($n$) and the list items with popularities $p$. It then calculates and returns $k$, the number of items that should be placed on the broadcast push channel. The starting point for Algorithm 2 is a method suggested by [6] that minimizes the bandwidth $B$ to achieve a target latency $L$. The known method is not directly applicable to document classification and bandwidth division because our goal, on the contrary, is to minimize the latency $L$ given a fixed amount of available server bandwidth $B$.

Algorithm 2 operates by using two bandwidth costs, one if the item is kept on the pull channel and one if the items is placed on the push channel. If document $i$ is assigned to the pull channel, it will use bandwidth $\lambda p_i S_i$. If document $i$ is assigned to the push channel, it will use bandwidth $S_i/L$, which is also the rate at which the document must be broadcast to give *worst-case* response time $L$. As it was stated in [6] a document should be pushed if $\lambda p_i S_i > \frac{S_i}{L}$. Because the items are passed in with an order of most popular (first item) to least popular (last item), a binary search can be performed on the items to find the item $k$ at which the division should occur.


Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 22:28:02 UTC from IEEE Xplore. Restrictions apply.
IEEE COMPUTER SOCIETY

### 2.2 Report Probabilities

Document selection and bandwidth division rely on estimates $p$ of document popularity. The values of $p$ can be estimated by sampling the client population as follows. The server publishes a report probability $s_i$ for each pushed document $i$. Then, if a client wishes to access document $i$, it submits an explicit request over the backchannel for that document with probability $s_i$. In principle, clients would not need to submit any request for push documents, but if they do send requests with probability $s_i$, the server can use those requests to estimate $p_i$. At the same time, the report probability $s_i$ should be small enough that server is almost surely not going to be overwhelmed with requests for pushed documents. In particular, we consider the objective of minimizing the maximum relative inaccuracy observed in the estimated popularities of the pushed documents. In this case, we show analytically that each report probability should be set inversely proportional to the predicted access probability for that document.

First, the server calculates the rate $\beta$ of incoming reports that it can tolerate. Presumably, $\beta$ is approximately equal to the rate that the server can accept connections over the backchannel minus the rate of connection arrivals for pulled documents. Therefore, the value of $\beta$ can be estimated from the access probabilities and the current request rate, all scaled down by a safety factor to give the server a little leeway for error. Then, the $s_i$'s have to be set such that $\sum_{i=1}^{k} \lambda p_i s_i \leq \beta$, where documents $1, \ldots k$ are on the push channel. The expected number of reports $\mu_i$ that the server can expect to see for $i$ over a unit time period is $\lambda p_i s_i$. Using standard Chernoff bounds, the probability that number of reports is more than $(1+\delta)\mu_i$ is roughly $e^{\frac{-\mu_i \delta^2}{4}}$, and that the probability that number of reports is less than $(1-\delta)\mu_i$ is roughly $e^{\frac{-\mu_i \delta^2}{2}}$. If the goal is to minimize the expected maximum relative inaccuracy of the reports, all of the upper tail bounds should be equal and all of the lower tail bounds should be equal. That is, all $\mu_i$ should be equal, or equivalently it should be the case that for all $i$, $1 \leq i \leq k$, $s_i = \frac{\beta}{\lambda p_i k}$. Hence, each document should have a report percentage inversely proportional to its access probability.

## 3 Experimental Evaluation

The objective of the experiments is to validate the algorithms introduced in the previous section. In particular, the development of Algorithm 1 made several idealized assumptions about the environment and these assumptions need to be investigated experimentally. The choice of $\alpha$ is a major parameter in the following experiments. We also wish to verify that lower delays are achieved by an integrated algorithm that does both document classification and bandwidth division. Finally, the scalability of various popularity estimation algorithms remains to be verified.

### 3.1 Methodology

The experimental analysis leverages on an existing prototype middleware. Because this middleware was purely simulation based, it is used to validate our algorithm regardless of the environment provided. The results shown are measured in relative time units. The middleware supports the hybrid dissemination scheme utilizing broadcast push and pull as shown in Figure 1. It acts as a reverse-proxy to a Web server for the delivery of documents that are materialized views [11]. A simulated client uses the middleware and generates Poisson requests for documents with a Zipf probability distribution. In this paper, we report on the case in which the size of the documents is fixed to 0.5KB, and we have additional evidence suggesting that results are fundamentally the same with variable sized documents.

An objective of this evaluation was to isolate algorithm performance from network factors, such as network congestion or routing transients. On the other hand, scalability is asserted when requests are generated by a large number of clients. Our solution was to run both the client and server on the same machine so that network effects would not be visible. (Although the emulation runs on a single machine, the middleware is capable of running on a distributed environment [11].) Aggregate requests from multiple clients was simulated by a background request filler. The filler simulates a specified number of clients, and sends requests to the server. The requests by the filler are treated identically to those made by another distinguished client, except that we record latency only for the requests from the distinguished client. There were 100 total documents available to be requested. All experiments were run for 10000 requests and figures reflect the average statistics from these runs.

### 3.2 Document Classification and Bandwidth Division Evaluation

Figure 2 shows the effects of various values of $\alpha$ on the average latency of Algorithm 1. The curve in Figure 2 is jagged because an infinitesimal change in $\alpha$ can have a discrete effect in the number of items pushed. Figure 2 shows that the value of $\alpha$ that minimizes average latency is between 2.0 to 3.0. We adopt $\alpha = 2.0$ in the rest of the paper — although this is not the actual minimum, any value in the range produces similarly good results. Note that as $\alpha$ changes in figure 2 our system adjusts the bandwidth division and document classification to maintain optimality. This in part explains why the average latency is near optimal for a relatively wide range of $\alpha$.

Figure 3 can be interpreted as a brute force search for


Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 22:28:02 UTC from IEEE Xplore. Restrictions apply.
IEEE COMPUTER SOCIETY

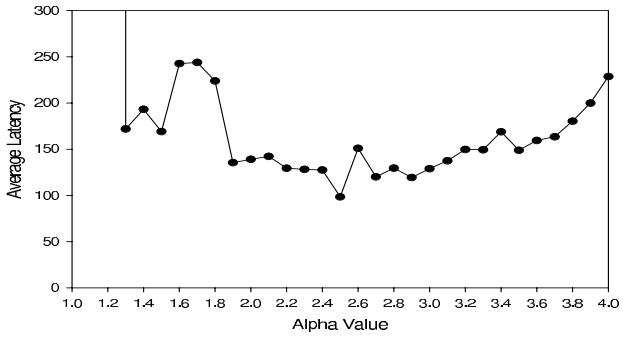


**Figure 2. Effects of various $\alpha$ values**

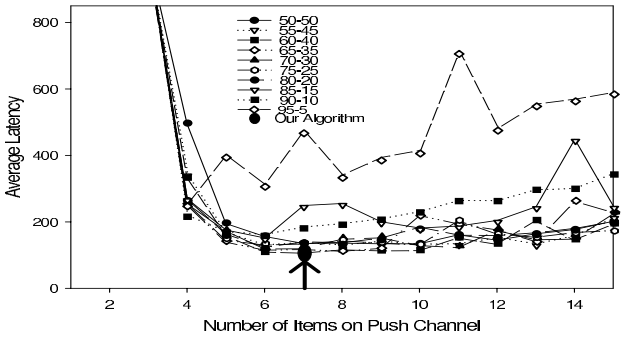


**Figure 3. Optimality of Algorithm 1 (arrow identifies single point found by the algorithm)**

a good bandwidth split and document classification by trying several closely spaced values of $k$ and *pushBW*. In the chart legend, the first number in the bandwidth split refers to pull. In addition to the points plotted in the figure, we verified that if less than half of the bandwidth was devoted to pull, the latency was suboptimal. In this scenario, Algorithm 1 assigns the most popular 7 documents on the push channel, and allocates 63% percent of the bandwidth to pull. The figure shows the algorithm's outcome with a circular point and an arrow pointing to it. The solution produced by our algorithm is better than any other point in the diagram. More specifically, our algorithm chose a split of 63/37 and the closest brute force curve in the figure is the 65/35 curve. The 65/35 line was also the lowest in the graph. Algorithm 1 chose $k = 7$ point as the number of push documents, which is also the minimum point on the 65/35 curve. Thus, Algorithm 1 chose a better bandwidth split than the brute force approach and a document classification that was just as good.

### 3.3 Report Probabilities Evaluation

In order to determine the usefulness of our proposed push popularity scheme, we compare it to a solution found

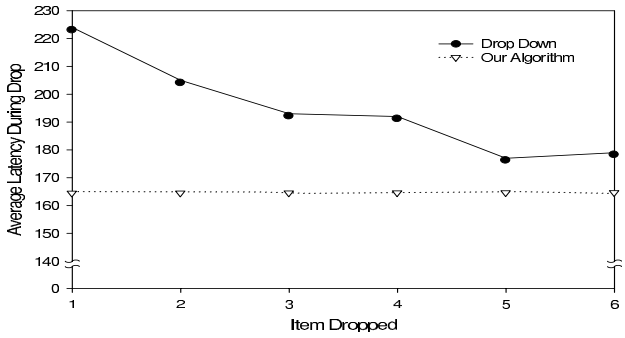


**Figure 4. Drop down method versus our probability method.**

in a comparable work to our own. The solution for the push popularity problem proposed in [14] was to occasionally drop each pushed document $i$ off of the push channel so that clients would have to make explicit requests to $i$. However, there is a danger that these explicit requests for $i$ could overload the server. Thus, in [14] it was recommended that $i$ should be dropped as short of a period of time as possible. The shortest possible time the the document can be dropped is one broadcast cycle. However, we show here that even such a short drop disrupts the server, while our proposed method does not suffer from such disruptions.

Figure 4 shows the average latency over the next 5 broadcast cycles when the $i^{th}$ most popular document is dropped from the push channel for one broadcast cycle. The flat line represents the average response time using our method for push popularity. If the most popular document is dropped, then we see a 35% increase in average latency over the next 5 broadcast cycles. If the 6th most popular document is dropped, we see an 8% increase in average latency over the next 5 broadcast cycles. This increase is in comparison to using the simple yet effective scheme we propose. Thus, based on all else being equal, using our scheme will maintain lower response times than using the drop down and check scheme.

## 4 Related Work

Hybrid data dissemination in wireless networks has received much attention in the research community [10, 1, 4, 14, 2]. These works focus on using multiple channels to distribute data. However, most of these papers focus on scheduling items on their respective channels or on organizing the broadcast [13, 12]. They try to distribute data across all channels evenly and in a way that will minimize user latency. However, each channel has both a popular and unpopular aspect to the data on the channels. We introduced a new way to look at using multiple channels, with the chan-


Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 22:28:02 UTC from IEEE Xplore. Restrictions apply.
IEEE COMPUTER SOCIETY

nels defined to serve specific types of documents, one dedicated to popular documents and one dedicated to unpopular documents. In addition, we focus on how to divide the documents and not on only how to schedule the documents once given a division.

The document classification problem was introduced in [14]. In addition to directly related work, some other work has been done addressing the issue of hot and cold documents and of bandwidth division, though not in the context we are describing. In [1, 9, 3, 15] the issue of mixing pull and push documents together on a single broadcast channel is examined. The idea is that popular documents are similarly considered hot, and are continuously broadcast while all other documents are cold. These documents are request through a back channel and scheduled for broadcast. Similarly, in [1] the authors discuss how to divide the broadcast channel bandwidth between hot and cold documents. The main difference between previous work and ours is previous work deals with a broadcast environment with a single channel and focuses on scheduling items, not how to divide them into hot and cold with separate channels and limited bandwidth. We are looking into the division of both documents and bandwidth to minimize latency.

The hybrid scheme relies on estimates of the popularity of documents because popularity determines the assignment of documents to dissemination modes. Popularity estimation can be approached separately for pulled and for pushed documents. Pull popularity can be solved in sublinear space by monitoring the client request stream [7]. As for push popularity, the problem is complicated by the absence of a client request stream. One solution is to occasionally drop each pushed document from the push channel, thus forcing clients to send explicit requests. Such requests can be counted and the document popularity estimated [14].

## 5 Conclusion

It has been shown that data broadcasting and selective tuning offer efficient solutions to the energy deficiency faced by mobile clients and the scalability issues present in a wireless network. Hybrid data broadcasting schemes attempt to combine these two techniques to achieve scalability with the maximum possible energy saving. In this paper, we examined three fundamental problems at a hybrid data broadcasting server. We argued that the document classification problem and bandwidth division problem should be solved in an integrated manner. We then presented a simple, yet essentially optimal, algorithm for the integrated problem which focused on latency. This also provides the benefit of reducing energy at the mobile clients by decreasing the required waiting time. We validated the optimality of our algorithm experimentally. We proposed solving the push popularity problem by having each client request a hot document $D_i$ with some probability $s_i$, which the server sets in the push index. We looked at the difference in using our push popularity scheme versus using a scheme which simply drops an item off the push channel in order to test its popularity. We showed that dropping an item off the hot channel for one broadcast cycle can appreciably increase the average latency for approximately five broadcast cycles. Our proposed scheme does not suffer from such disruptions.

**Acknowledgments:** We would like to thank the reviewers of MDM 06 and WebDB04 where an earlier version of this work was presented.

## References

[1] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull data broadcast. In *ACM SIGMOD*, 1997.
[2] R. Agrawal and P. K. Chrysanthis. Efficient data dissemination to mobile clients in e-commerce applications. In *WECWIS*, 2001.
[3] D. Aksoy and M. Franklin. RxW: A scheduling approach for large-scale on-demand data broadcast. *ACM/IEEE Transactions on Networking*, 7(6), 1999.
[4] K. C. Almeroth, M. H. Ammar, and Z. Fei. Scalable delivery of Web no-pages using cyclic best-effort (UDP) multicast. In *INFOCOM*, 1998.
[5] M. Altinel, D. Aksoy, T. Baby, M. Franklin, W. Shapiro, and S. Zdonik. DBIS toolkit: Adaptable middleware for large scale data delivery. In *ACM SIGMOD*, 1999.
[6] Y. Azar, M. Feder, E. Lubetzky, D. Rajwan, and N. Shulman. The multicast bandwidth advantage in serving a web site. In *3rd NGC*, 2001.
[7] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking frequent items dynamically. In *ACM PODS*, 2003.
[8] Y. Guo, M. Pinotti, and S. Das. A new hybrid scheduling algorithm for asymmetric communication systems. *ACM M2CR*, 5(3), 2001.
[9] A. Hall and H. Taubig. Comparing push- and pull-based broadcasting or: Would "microsoft watches" profit from a transmitter? In *WAE*, 2003.
[10] H.P.Hung and M.S.Chen. A general model of hybrid data dissemination. In *MDM*, 2005.
[11] V. Penkrot, J. Beaver, M. Sharaf, S. Roychowdhury, W. Li, W. Zhang, P. Chrysanthis, K. Pruhs, and V. Liberatore. An optimized multicast-based data dissemination middleware: A demonstration. In *ICDE*, 2003.
[12] E. Pitoura and P. K. Chrysanthis. Multiversion wireless data broadcasting. *IEEE Transactions on Computers*, 51(10), 2002.
[13] O. Shigiltchoff, P. K. Chrysanthis, and E. Pitoura. Multiversion data dissemination: Broadcast organizations and caching. *Information Systems Journal*, 29(6), 2004.
[14] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *VLDB*, 1997.
[15] P. Triantafillou, R. Harpantidou, and M. Paterakis. High performance data broadcasting systems. *MONET*, 7(4), 2002.
[16] W. Zhang, W. Li, and V. Liberatore. Application-perceived multicast push performance. In *IPDPS*, 2004.


Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 22:28:02 UTC from IEEE Xplore. Restrictions apply.
IEEE COMPUTER SOCIETY