Information Delivery Through Broadcasting in Satellite Communication Networks

Leandros Tassiulas
leandros@isr.umd.edu
Electrical Engineering Department
University of Maryland
College Park MD 20742
Tel: (301) 405-6620
FAX: (301) 314-9920

Chi-Jiun Su
csu@hns.com
Hughes Network Systems
11717 Exploration Lane
Germantown MD 20876
Tel: (301) 601-7346
FAX: (301) 428-2750

Abstract

Satellite broadcast is an important candidate for large-scale multimedia information distribution due to the inherent wide-range multicasting capability of satellites and the asymmetry of satellite communications (high bandwidth downlink, limited bandwidth uplink) that matches nicely the information flow asymmetry in multimedia applications. We consider a data broadcasting model that is encountered in most asymmetric satellite communication environments.

The problem of scheduling the data broadcast such that average response time experienced by the users is low is considered. In a push-based system, where the users cannot place requests directly to the server and the broadcast schedule should be determined based solely on the access probabilities, we formulate a deterministic dynamic optimization problem, the solution of which provides the optimal broadcast schedule. Properties of the optimal solution are obtained and then we propose a suboptimal dynamic policy which achieves average response time close to the lower bound. In a pull-based system where the users may place requests about information items directly to the server, the scheduling can be based on the number of pending requests for each item. Suboptimal policies with good performance are obtained in this case as well. If a user has local memory, it can alleviate its access latency by selectively prefetching the items from the broadcast and storing them in the memory. A good memory management strategy can substantially reduce the user's access latency. An

optimal memory management policy is identified, that minimizes the expected aggregate latency. Memory update strategies with limited look-ahead are presented as implementable approximations of the optimal policy as well. We also considered the problem of joint broadcast scheduling and user's cache management and proposed a joint optimization scheme which can achieve the performance up to 40% better than the existing non-joint approach.

1 Introduction

Broadcast data delivery is rapidly becoming the method of choice for disseminating information to massive user populations in Direct Broadcast Satellite systems. Under the broadcasting approach as shown in figure 1, a server continuously and repeatedly broadcasts information to a user community with limited feedback about the user's needs due to the limited uplink communication capability from the user to the server. The information broadcast by the server is organized into units called *information items*. When a user needs a certain information item, it monitors the broadcast channel until the desired item is detected and captures it for use. The transmission of an information item satisfies simultaneously all the pending requests for that item, and hence it has a multicasting effect.

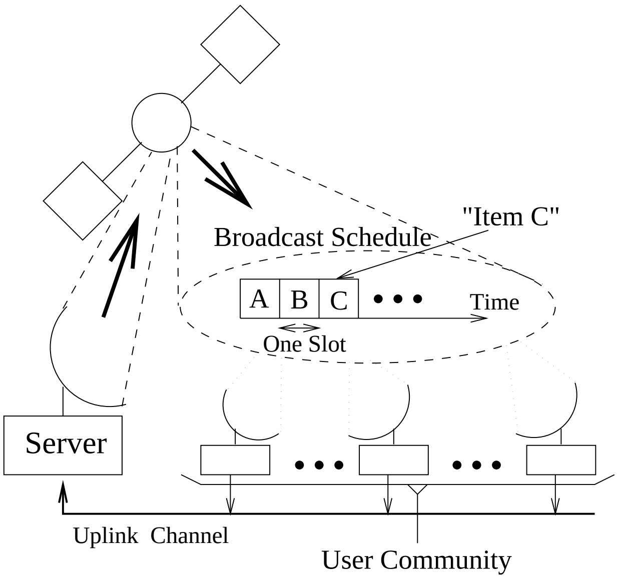

Figure 1: *A Direct Broadcast Satellite Data Delivery System*

Broadcast delivery appears to be a promising alternative to the traditional client-to-server transaction due to the communication asymmetry inherent in several wireless communication environments. Communication asymmetry can generally arise in two ways: either by *physical asymmetry*, when an asymmetric bandwidth or power limitations exist between downlink and uplink communication, or by *asymmetric information flow*, as in an information retrieval system with a large population of users where the amount of information flowing from the server to the users is considerably larger than the flow from the users to the server. The main advantage of broadcast delivery is its scalability: it is independent of the number of users the system is serving.

There are several possible architectures for a broadcast delivery system depending on the capacity of the communication channel from the user population to the server. In *push-based* delivery, the users cannot inform the server about what they actually need due to the lack of, or, limited uplink communication channel from the users to the server. The server should design the schedule relying exclusively on the average user access statistics in this case. In *pull-based* delivery, there is a uplink channel available through which a user can submit access requests to the server and this information can be used for scheduling in a dynamic fashion. When there exists a limited uplink capacity from the users to the server, the users cannot send requests for all the information items in the server's database. In this case, information items are classified into two mutually exclusive sets, namely, $\mathcal{DB}_{push}$, the set of frequently requested items, and $\mathcal{DB}_{pull}$, the set of less popular items. Users are allowed to submit requests only for the items in $\mathcal{DB}_{pull}$ while requests for the items belonging to $\mathcal{DB}_{push}$ are serviced using push-based delivery.

There are several issues to be addressed in data broadcasting. One of them is the choice of delivery architectures — push, pull or hybrid — based on the capacity of the uplink channel. The main challenge from the server's side is the organization of the data in a broadcast schedule so as to minimize the average response time of user's requests. The problem on the users' side is related to user's local memory management in order to reduce the mismatch between the broadcast schedule and user's access pattern. Both server's broadcast scheduling and user's cache management can be jointly considered in an integrated approach which provides broadcast schedules and user's cache management schemes simultaneously. The issues in a multistage system are cache management in intermediate stages and joint buffer management and schedule design in local broadcast stations.

The problem of broadcast schedule design for push-based delivery has been considered in [13], [8], [7], [9], [2], [25] and [14]. The motivation for the work in [13], [8], and [7] was teletext systems. In [8], it was shown that within a rather general class of scheduling policies for broadcast data for which the mean response time exists, a *periodic* policy is optimal in terms of minimizing the mean response time. The results about the parameters of the optimal schedule from [8] were later extended to the case with information items of unequal length and the case where the transmission of an item is successful with some probability in [25]. Scheduling for the pull-based system was considered in [11], [10], [16] and

[5]. The problem was first considered in [11] in the context of videotex systems and later investigated for a quasi video on-demand system with impatient users in [10], [16] and [5]. A push-based broadcast/pull-based unicast hybrid model was considered in [27] and [26]. In [17], Adaptive partitioning of information items for the same model was provided in which items are grouped according to the estimated user's request generation statistics based on the number of requests received through the uplink channel. A push/pull-based broadcast hybrid model was considered in [4] where, at each slot, a coin weighted by the capacity of the uplink is tossed to determine whether a push-based or pull-based item is transmitted.

In [18] and [20], the scheduling problem in a push-based system is formulated as a deterministic MDP. Dynamic scheduling policies are considered where the scheduling decision at a slot is based on the elapsed time since the last transmission of each item. Properties of the optimal policy are identified. Furthermore, a class of policies is identified which has near optimal performance, of the same level or slightly better than the periodic scheduling policies proposed in [8]. Our policies have the advantage of being simple to implement in a real time fashion, adaptive to changes in the access statistics, and readily generalizable to systems with multiple parallel broadcast channels. For a pull-based system, the problem is formulated as a stochastic MDP. Properties of the optimal policy are identified and variations of the real-time scheduling schemes considered for the push-based system are evaluated and compared with previous results. Comparing the performance of push-based and pull-based systems, we observed that in certain cases and for sufficiently large request generation rates, the performance of the two systems is at about the same levels. That is, the availability of a feedback channel for request placement capability does not improve the performance significantly at heavy load.

The latency of user's requests is determined not only by the broadcast schedule but also by user's access pattern. In push-based delivery, due to the lack of the uplink channel, the server has to solely rely on the user's access statistics for broadcast scheduling in order to minimize the latency of user's requests. Therefore, if there are more than one class of users with different access distributions of information items, then it is unavoidable that some classes will suffer large latency. An approach to reduce the latency to a desirable level for each user is to make use of local user storage.

If a user has local storage, it can retrieve information items from the broadcast and store them in its memory prior to the items being requested. If the user makes a request for one

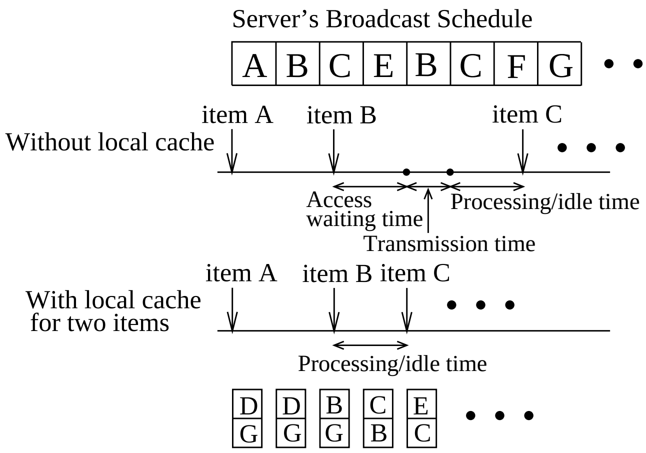

Figure 2: *The use of local cache to reduce the access latency. The arrows labeled with "*item X*" represents the requests for item X generated by the user. The user is assumed to have a cache which can hold two items and the two rows of rectangles correspond to cache states of the user. If item B is stored in the cache when the request for item B is generated, there is no access waiting time and transmission time for that access for item B.*

of the "prefetched" stored items, the response time for this request will be instantaneous. By selectively prefetching information items from the broadcast, the user is effectively able to minimize the mismatch between its access needs and server's broadcast schedule and the average latency of its information requests is reduced as depicted in figure 2. Therefore, user's memory management becomes an important issue to consider in order to minimize the average response time of user's requests. A number of heuristic user's memory management policies was proposed in [6], [1] and [3].

In [23], an optimal memory management policy is identified, that minimizes the expected aggregate latency. Optimal memory update strategies with limited look-ahead are presented as implementable approximations of the optimal policy. Some interesting special cases are given for which the limited look-ahead policies are optimal. It is also shown that the same formulation can be used to find the optimal memory management policy which minimizes the number of deadline misses when users generate information requests which have to be satisfied within some given deadlines.

The following figure 3 shows the functionality of a server's broadcast scheduling policy and a user's cache management policy. The scheduling policy first transforms the set of user's access statistics into one aggregate access statistics and then employs it for scheduling of items in the broadcast. Given the access statistics of a user, the cache management policy decides if an item arriving in the broadcast will be prefetched. When a user needs an item, it checks if the item is in the cache. If not, it has to wait until the item appears in the broadcast.

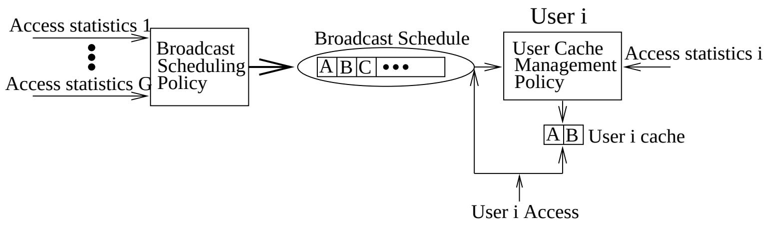

Figure 3: *The functionality of a broadcast scheduling policy and a user's cache management policy*

The broadcast scheduling problem and the user's memory management problem are complementary to each other and they were addressed separately in literature. Traditional broadcast scheduling schemes does not take into account of user's caching although prefetching the broadcast items into user's cache can be effectively employed at the user's side in order to further reduce the mean response time of user's requests. In [21], an integrated approach is considered which produces broadcast schedules and user's memory management schemes simultaneously based on the mix of data access pattern of user's groups. Under the joint approach, broadcast schedules are designed such that the schedules are conducive to user's caching and the overall mean response time can be considerably reduced when caching is done at user's side. A fluid model of the joint problem is identified on which the joint optimization is performed and subsequently the schedules are constructed according to the optimal fluid model parameters. Numerical results show that our joint approach can reduce the mean response time of the non-joint scheme up to 40%. It is also shown that the mean response time of user's requests under any broadcast scheduling and cache management policy is lowerbounded by the solution of the optimization problem. A lowerbound for the performance of any cache management strategy, when the server's broadcast schedule is known, is also provided which can be used as an benchmark to investigate the performance of a cache management policy.

In some cases, the users have no direct access to the server due to bigger antennas and receivers of higher complexity required to receive the direct broadcast from the satellite. One feasible solution for this scenario is multistage information delivery architecture where the satellite transmission is received first by a local ground station, which then redistributes the data to users through a terrestrial wireless system. In [12], broadcast scheduling and buffer management for a two stage information distribution system was considered and several heuristics for joint broadcast scheduling and buffer management were proposed.

The paper is organized as follows. Some examples of systems which distribute information through broadcast delivery are provided in section 2. The problem for broadcast scheduling is formulated in section 3. In section 4.1, the push-based system is studied. The pull-based system is investigated in section 4.2. The performance limit of a pull-based delivery system is discussed in section 4.3. The results are extended for a system with multiple broadcast channels and for information items of unequal length in section 5 and in section 6 respectively. In section 7, user's cache management strategies are provided. Joint broadcast scheduling and user's cache management is investigated in section 8. A multistage broadcast delivery system is discussed in section 9. The paper is concluded with future research directions in broadcast delivery in section 10.

2 Systems which Employ Broadcast Delivery

Some examples of systems which employ broadcast delivery are real-time high speed internet access systems, pull-based internet information distribution systems, data broadcasting systems in TV channels and Internet Broadcast systems.

Hughes' DirecPC and *Philips Clevercast PC* are the systems which provide real-time high speed internet access through broadcast delivery. Since most of the users in these systems usually consume much more data than they can generate, these systems are designed to be asymmetric in bandwidth, that is, higher transmission capacity for the downlink channel and smaller bandwidth for the uplink. Requests for information items are submitted to the server through the low-bandwidth terrestrial link while information items requested by the users are broadcast over the high data rate satellite channel. Users uses receive-only satellite disks to receive data from the server and the public telephone network to transmit the requests to the server.

Intel's Intercast and *Norpak's VBinet* broadcast data to the user's PC through the TV broadcast channel during the vertical blanking interval (VBI) of the TV signal. There are a maximum of 10 VBI lines available, and each line has data transmission capacity equivalent to a 9600 bps modem. In Intercast, TV broadcasters transmit HTML items with their TV programs. The information transmitted may include in-depth stories or statistics on the program broadcast on TV. There may be hyperlinks on the items broadcast which can be used by the users to access other sites with more information through the telephone link with an internet service provider.

AirMedia Live Internet Broadcast Network broadcast information from the content providers such as CNN Interactive, CBS SprotsLine and Quote.com wirelessly to users's PC. AirMedia encodes the information and sends it out across a national paging network and users receive the information at their computers through the Internet Antenna. The software on a user's computer decodes the information, filters it through the user's preference profile and shows it on the screen of the computer.

PointCast, *Marimba's Castanet* and *Backweb* distribute information pulled by the users through the internet. Users subscribe the sets of information of their interest from the server. The server distributes the sets of information each user subscribes through the internet and transmits the update (periodically) whenever the information content changes. These systems do not require the additional hardware. All the necessary software is downloaded to the user during subscription.

3 The Broadcast Model

A slotted system is considered and slot n is the interval $[n, n+1)$. At each slot n, one item is broadcast in the channel and it is denoted by u_n, $u_n \in \{1, ..., M\}$ where M is the total number of possible items in the server's database.

Requests for items are generated by the users. A request for item i generated at time t will be satisfied at the next slot after t at which item i will be broadcast. Let $l_i(t)$ denote the number of slots from the end of slot n, where $t \in [n, n+1)$, until the end of the next slot after n at which the item is broadcast as shown in figure 4. Note that the latency of the request is equal to $l_i(t) + (n+1-t)$. Since the residual time $n+1-t$ from the generation of the request until the end of the slot is independent of the broadcast schedule, we will ignore it in the following and we will use $l_i(t)$ as the measure of the latency that will be experienced by an item i request generated at t.

Let $\rho_i(k,t)$ be the delay, that has been experienced by an item i request generated at time k, $k \leq t$, until time t.

$$\rho_i(k,t) = \min\left(l_i(k), t - k'\right)$$

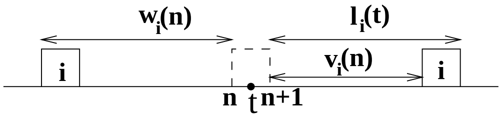

Figure 4: *Illustration of $l_i(t)$, $w_i(n)$ and $v_i(n)$ for a sequence of item i broadcasts*

where $k \in [k'-1, k')$. Denote the sequence of times at which item i requests are generated as $t_n^i, n = 1, 2, ...$ for each item $i = 1, ..., M$. The aggregate delay, $D_i(t)$, of all item i requests up to time t, is

$$D_i(t) = \sum_{t_n^i \leq t} \rho_i(t_n^i, t).$$

and let $\overline{D_i}(t)$ be its expected value, $\overline{D_i}(t) \triangleq E[D_i(t)]$. The aggregate expected delay over all items is

$$\overline{D}(t) = \sum_{i=1}^{M} \overline{D_i}(t)$$

In several applications, the performance of a scheduling policy is quantified by $\overline{D}(t)$ and the objective is to minimize it.

Let $X_i(n)$ be the total number of pending requests for item i at the beginning of slot n. The aggregate delay experienced by all item i requests up to time t is related to the item i request backlog as follows:

$$D_i(t) = \sum_{s=0}^{t-1} X_i(s)$$

Consider the aggregate stream of item requests generated by the whole user population. If the user population is large enough, the process of request generation is stationary with a constant rate λ. A request is for item i with probability b_i, $i = 1, ..., M$, where $\sum_{i=1}^{M} b_i = 1$. Hence, requests for item i are generated according to a stationary process with rate $\lambda_i = b_i\, \lambda$. Let $A_i(n)$ be the total number of requests for item i generated during slot n. The request backlog for item i evolves as follows:

$$X_i(n+1) = \begin{cases} A_i(n) & \text{if } u_n = i \\ X_i(n) + A_i(n) & \text{otherwise} \end{cases}$$

The objective of a scheduling policy is to find a sequence of broadcast items, u_n's, which minimizes the long term average delay

$$\lim_{T\to\infty} \sup \frac{1}{T} \sum_{n=1}^{T} \sum_{i=1}^{M} \overline{X_i}(n).$$

4 Scheduling for a Broadcast Delivery System

The scheduling problem in both push- and pull-based delivery is formulated as a MDP the solution of which provides the optimal broadcast schedule. Low complexity near-optimal policies are provided which yield the performance close to the lowerbound for push-based delivery and better than all the existing heuristics in literature for pull-based delivery.

4.1 Push-based Delivery

When the server is not aware about the user's requests, the broadcast schedule is designed based only on the distribution of user's access probabilities. In [18] and [20], we formulate the scheduling problem to minimize the response time of user's requests in a push-based system as a deterministic MDP. Dynamic scheduling policies are considered where the scheduling decision at a slot is based on the elapsed time since the last transmission of each item. Let $w_i(n)$ be the elapsed time from the beginning of the last transmission of item i before n until the beginning of slot n, as illustrated in figure 4. The evolution of $w_i(n)$ can be given as follows:

$$w_i(n+1) = \begin{cases} 1 & \text{if } u_n = i \\ w_i(n)+1 & \text{otherwise} \end{cases}$$

Hence, $w(n)$, $n = 0, 1, ...$ is a deterministic MDP controlled by u_n.

From the MDP formulation, properties of the optimal policy are identified. The first property is that if the optimal policy which minimizes the mean response time selects item j for transmission at slot n when $w_i(n) = \hat{w}_i$ for $i = 1, ..., M$, it is also optimal to select item j for transmission at slot n when $w_i(n) = \tilde{w}_i$, $i = 1, ..., M$ where $\tilde{w}_l = \hat{w}_l$ for $l \neq j$ $l = 1, ..., M$ and $\tilde{w}_j \geq \hat{w}_j$. The second one is that if $\lambda_i = \lambda_j$ and $w_i < w_j$, then it is not optimal to transmit item i. It immediately follows from the second property that if all the items have the same access probabilities, it is optimal to transmit the item with the largest w_i.

If there are only two items, then the optimal policy can be completely characterized as follows. Without loss of generality assume that $\lambda_2 \geq \lambda_1$. The optimal policy, when there are only two items, is periodic with a period consisting of one transmission of item 1 followed by m ($m \geq 1$) transmissions of item 2 where

$$m = \max\left(1\ ,\ \arg\min_{l \in \mathcal{S}}\ D_o(l)\right).$$

$D_o(.)$ is the mean response time of such a policy and $\mathcal{S} = \left\{\left\lfloor\sqrt{\frac{2\lambda_2}{\lambda_1}} - 1\right\rfloor, \left\lceil\sqrt{\frac{2\lambda_2}{\lambda_1}} - 1\right\rceil\right\}$.

Specifying the exact form of the optimal scheduling policy appears to be an intractable problem in general. We proposed a class of policies, called "*Priority Index Policies*" where the broadcast scheduling is determined based on priority indices of the items. The index of item i is the product $\lambda_i^{\zeta}\ w_i(n)$ where ζ is an exponent that reflects the relative importance of λ_i versus $w_i(n)$ in determining the priority. At each slot n, the Priority Index Policies broadcast the item

$$u_n = arg \max_{i \in \{1,\ldots,M\}}\ \lambda_i^{-\gamma}\ \overline{X_i}(n)\ \ =\ arg \max_{i \in \{1,\ldots,M\}}\ \lambda_i^{\zeta}\ w_i(n)$$

where $\zeta = (1 - \gamma)$.

When all the items have identical request generation rates, the priority index policies for all ζ's generate uniform periodic schedules which are optimal in this case. For $\zeta = \gamma = 0.5$, the index of item i is $\lambda^{0.5} w_i(n) = \sqrt{\lambda_i w_i^2(n)}$. Since $\frac{1}{2}\lambda_i w_i^2(n)$ is the aggregate expected delay experienced by item i requests since the last transmission of item i before slot n, for $\zeta = 0.5$, the item with the largest *Mean Aggregate Delay* (*MAD*) is selected for transmission.

We performed an extensive numerical study of the performance of the system under the priority index policies for various values of ζ. It turns out that the *MAD* policy ($\zeta = 0.5$) has the best performance in most cases. Furthermore, the performance of MAD is very close to the lower bound on the mean response time, that was given in [7].

Comparisons are made for $M = 100$ to $M = 1000$ and $\zeta = 0$ to 1.0 for the following two cases. In the first case, user access probabilities are assumed to follow the zipf distribution version I [28], that is, $b_i = \frac{c}{i}$, $i = 1, \ldots, M$ where c is a normalizing constant given by $c = (\sum_{j=1}^{M} 1/j)^{-1}$. In the second case, we assume item access probabilities follow the zipf distribution version II [15] where $b_i = \frac{i^{\theta} - (i-1)^{\theta}}{M^{\theta}}$ where $i = 1, \ldots, M$. As θ decreases, the access pattern becomes increasingly skewed. The value of θ used in this experiment is $\log(0.8)/\log(0.2)$. Zipf distribution is typically used to model non-uniform access patterns.

The mean response time for the heuristic policies is also compared to the lower bound for a periodic broadcast schedule which is obtained in [7] and is given by

$$\frac{1}{2} \left(\sum_{i=1}^{M} \sqrt{b_i} \right)^2 .$$

Table 1: *Mean Response Time in* slots *for different values of ζ using zipf distribution I (L. B. denotes Lower Bound)*

M	1	0.75	0.6	0.5	0.25	0	L. B.
100	48.49	36.61	33.82	33.36	37.60	50.0	33.31
200	97.56	68.92	62.52	61.41	70.42	100.0	61.36
300	145.21	99.75	89.58	87.81	101.72	150.0	87.77
400	193.69	129.72	115.67	113.22	131.65	200.0	113.18
500	244.29	159.06	141.07	137.93	161.15	250.0	137.90
600	295.68	187.86	165.93	162.11	190.65	300.0	162.08
700	343.00	216.37	190.37	185.86	218.91	350.0	185.83
800	389.06	244.60	214.45	209.25	246.71	400.0	209.21
900	437.05	272.26	238.23	232.34	274.10	450.0	232.29
1000	486.86	299.88	261.74	255.15	301.38	500.0	255.13

Table 2: *Mean Response Time in* slots *for different values of ζ using zipf distribution II*

M	1	0.75	0.6	0.5	0.25	0	L. B.
100	46.63	27.02	23.63	23.14	28.31	50.0	23.08
200	94.62	53.08	46.19	45.13	55.28	100.0	45.03
300	136.23	78.94	68.57	66.95	81.55	150.0	66.86
400	175.98	105.59	91.01	88.69	108.02	200.0	88.58
500	216.31	130.74	113.23	110.35	134.00	250.0	110.25
600	262.66	156.73	135.43	132.06	160.41	300.0	131.94
700	310.82	183.71	157.65	153.66	186.10	350.0	153.57
800	359.62	207.51	179.84	175.22	211.90	400.0	175.18
900	409.25	233.77	202.00	196.80	237.27	450.0	196.77
1000	459.46	260.59	224.17	218.38	262.52	500.0	218.32

According to the results from Table 1 and 2, the *MAD* policy ($\zeta = 0.5$) yields the best performance which is also close to the lower bound for both distributions. *MAD* also gave the best mean response time among the priority index polices for various distributions we have tried (not mentioned here) and its performance is consistently close to the lower bound.

Although the algorithm proposed in [7] also yields a mean response time close to the lower bound, the *MAD* policy has a number of advantages over other existing methods for designing broadcast schedules. First, it automatically generates broadcast schedules without requiring to select the three basic parameters of a periodic schedule — period length, appearance frequencies and inter-appearance gaps of each item in one period. Second, the *MAD* policy does not need to perform any precomputation before the broadcast. It selects the item to transmit at each slot during the broadcast according to the given user's access probabilities. Thus, the policy can adapt the schedules as the user access pattern changes. Third, it can be generalized easily for multi-channel systems. Moreover, the *MAD* policy is easy to implement; it only needs to keep M values of $w_i(n)$ at each slot n in addition to the access probabilities and the only operations required to perform at each slot n are to update the values of $w_i(n)$ and to determine the item with the largest expected aggregate delay of the current request backlog. Therefore, both the computational complexity and the storage requirement of the *MAD* policy is just $O(M)$. On the other hand, the approach in [7] is an *off-line* algorithm which has to construct the whole schedule and store it before the broadcast. This may require a considerable storage when the period of a schedule is large, which is usually the case when there are a large number of items and access probabilities are non-uniform. Furthermore, it is easy to show that the *MAD* policy generates schedules that are periodic.

4.2 Pull-based Delivery

When there is an uplink channel available for the users to submit requests, the server knows the exact number of pending requests for each item at each slot and it can make the scheduling decision based on that information.

In this case, $X(n)$, $n = 0, 1, ...$ is a stochastic MDP controlled by u_n and the properties of the optimal policy for this MDP are as follows. The first property is that if the optimal policy which minimizes the mean response time selects item j for transmission when the request backlogs are $\hat{X}_i$, $i = 1, ..., M$, then it is also optimal to select item j for transmission when the request backlogs are $\tilde{X}_i$, $i = 1, ..., M$ where $\tilde{X}_l = \hat{X}_l$ for $l \neq j$ $l = 1, ..., M$ and $\tilde{X}_j \geq \hat{X}_j$. The second one is that if $\lambda_i = \lambda_j$ and $X_i < X_j$, then it is not optimal to transmit item i. The second property proves the conjecture in [11] that if all the items have the same access probabilities, the optimal policy is to broadcast the item with the largest backlog at each slot. Moreover, a class of policies similar to those for the push-based delivery, were also proposed which, at each slot n, select the item

$$u_n \;=\; \arg \max_{i \in \{1,\ldots,M\}} \;\lambda_i^{-\gamma}\, X_i(n)$$

Similar to the push-based system, when all the request generation rates are equal, the priority index scheduling policies also produce the optimal schedule for the pull-based system.

A number of heuristic scheduling policies for the push-based system were proposed in [11]. Two of them are the *Most Request First* (*MRFL*) policy, which selects the item with the largest number of pending requests and breaks ties in favor of the lowest probability item, and the *Longest Wait First* (*LWF*) policy, which selects the item for which the total waiting time of pending requests is the largest. The *MRFL* policy corresponds to the priority index policy with $\gamma = 0$. Since, according to the simulation results in [11], the *LWF* policy yields significantly better response time characteristics than other heuristic policies, we compare the priority index policies to the *LWF* policy by simulation. The results for 1000 items with zipf I and zipf II distribution are shown in figure 5 and 6 respectively.

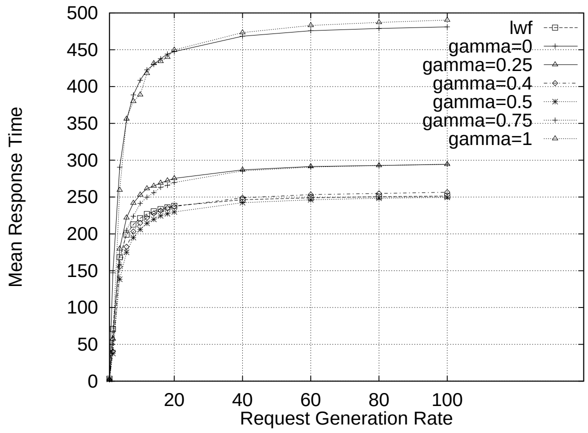

Figure 5: *Mean Response Time (in slots) vs. Aggregate Request Generation Rate (requests per slot) for different values of γ using zipf distribution I for 1000 items*

For light load, the mean response time is insensitive to the particular scheduling algorithm employed. As the request generation rates increases, the policy with $\gamma = 0.5$ exhibits the best mean response time (even slightly better than the *LWF* policy) for all aggregate request generation rates. Note that the policy with $\gamma = 0.5$ is easier to implement than the *LWF* policy since, at each slot, it only needs to keep track of the request backlog for each item while the *LWF* policy has to compute the total waiting time up to the current slot for each item.

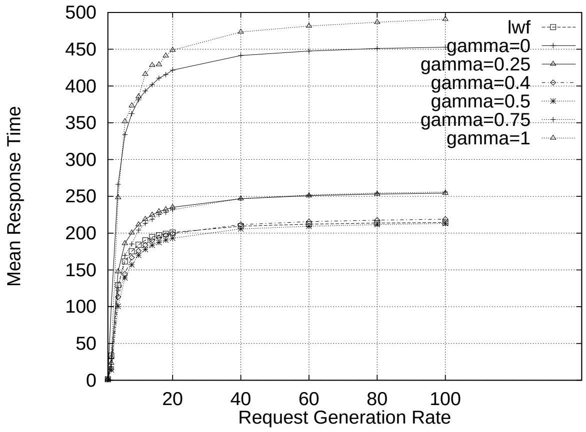

Figure 6: *Mean Response Time (in slots) vs. Aggregate Request Generation Rate (requests per slot) for different values of* γ *using zipf distribution II for 1000 items*

4.3 Performance Limits of a Pull-Based Broadcast System

A pull-based system requires the availability of an uplink channel and has the undesirable property that the uplink channel may become overloaded under heavy aggregate request generation rate. At heavy load, the availability of the uplink channel for request submission does not improve the performance significantly. It was shown in [22] that when the same Priority Index policy,

$$u_n \;=\; \arg \max_{i \in \{1,\ldots,M\}} \; \lambda_i^{-\gamma}\, Y_i(n)$$

where

$$Y_i(n) = \begin{cases} \lambda_i w_i(n) & \text{for push-based delivery} \\ X_i(n) & \text{for pull-based delivery} \end{cases}$$

is used for both push and pull systems, the request generation process is Poisson, and access probabilities are identical, the performance of a pull-based system approaches that of a push-based system. For the case with non-uniform access probabilities, the result can be extended for the MAD policy. We also conjecture that it is true in more general cases and the simulation results in the following support our claim.

Our simulation results also show that the mean response time of a pull-based system approaches that of a push-based system as the aggregate request generation rate increases. Figure 7 shows the simulation results for the case of equal request generation rates (λ_i is the same for all items) and 500 items. As the aggregate request generation rate increases beyond

20, the mean response time of the pull-based system approaches 250 slots which happens to be the mean response time of the optimal schedule for the push-based system.

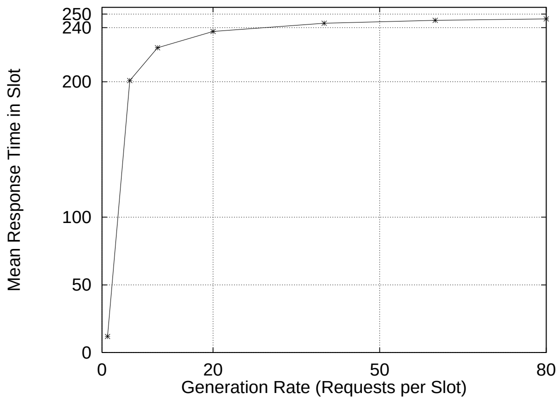

Figure 7: *Mean Response Time (in slots) vs. Aggregate Request Generation Rate (in request per slot) for 500 items with equal generation rates*

For the case with unequal request generation rates, consider the policy with $\gamma = 0.5$ as an example for both pull-based and push-based systems. From figures 5 and 6, the mean response time of the policy with $\gamma = 0.5$ for 1000 items with zipf distribution I and II for the pull-based system are 249.59 and 213.09 slots respectively at aggregate request generation rate 100 requests per slot while the mean response times for the push-based system for zipf distribution I and II are 255.12 and 218.36 slots respectively from tables 1 and 2.

5 Extension to a System with Multiple Broadcast Channels

We also extended our results for the push-based delivery to the multi-channel system in [18]. Sometimes, since the available bandwidth for the wireless broadcast channel is considerably large, the channel has to be divided into a number of sub-channels with smaller bandwidth due to implementation constraints. Therefore, there are more than one broadcast channels available and a number of items equal to the number of channels is broadcast at each slot. Assume that all the users have the ability to tune in to any of the broadcast channels and retrieve the corresponding item. By using a small fraction of the bandwidth, the server may inform the users about which items are being broadcast at which channels at each slot so that a user will know which channel he should tune in at each slot. A system with J

broadcast channels is depicted in figure 8.

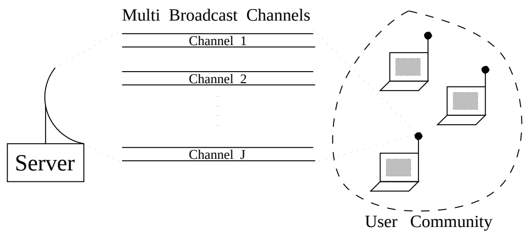

Figure 8: *A System with Multiple Broadcast Channels*

The problem of designing broadcast schedules for a push-based system in this case is different from the single channel case. At each slot n, the server has to select J items to broadcast to the users. The request backlog of any of the J items broadcast during slot n vanishes at the end of slot n.

The existing algorithms for designing broadcast schedules for a push-based system in literature are only intended for a single-channel system and they cannot be easily extended for the multichannel case. The *MAD* policy, however, can be readily generalized for the multichannel case as follows.

At slot n select the J items, for which the quantity $\lambda_j^{0.5} w_j(n)$ is largest and broadcast them. In the same way, the *MAD* policy can be applied to a pull-based system with multiple channels.

The lower bound on the mean response time for the single channel system can be readily generalized for a system with J broadcast channels to be

$$\frac{1}{2J} \left(\sum_{i=1}^{M} \sqrt{b_i} \right)^2$$

The mean response time of the schedules produced by *MAD* policy is compared to the lower bound from $M = 60$ to $M = 100$. For each M, we consider the number of channels to be approximately 5% and 10% of the total number of items. The results are given in Table 3 and 4. The examples show that, if the number of broadcast channels is increased twofold, the mean response time decreases nearly by half. Note that, ideally, we would like to get the mean response time reduced exactly by half when the number of channels is doubled. In all the cases we consider, the schedules produced by the *MAD* policy incur mean response time close to the lower bound.

Table 3: *Mean Response Time in* slots *for zipf distribution I*

M	No. of Channels	Lower Bound	MAD
60	3	7.08	7.10
60	6	3.54	3.57
70	3	8.11	8.13
70	6	4.05	4.07
80	4	6.84	6.86
80	8	3.42	3.44
90	4	7.59	7.61
90	8	3.79	3.83
100	5	6.66	6.68
100	10	3.33	3.35

Table 4: *Mean Response Time in* slots *for zipf distribution II*

M	No. of Channels	Lower Bound	MAD
60	3	4.73	4.79
60	6	2.37	2.47
70	3	5.47	5.50
70	6	2.74	2.82
80	4	4.66	4.70
80	8	2.33	2.37
90	4	5.21	5.28
90	8	2.61	2.69
100	5	4.61	4.62
100	10	2.30	2.31

6 Information Items of Unequal Length

In some applications, information broadcast by the server cannot be organized into items of equal length. When the items are of different length, item length information should be taken into consideration for broadcast scheduling in order to obtain good performance. In [19], we extended our *Priority Index Policies* (*PIP*-γ) to the case with items of unequal length for both push- and pull- based delivery. At each slot n, the policy selects, for transmission,

$$u_n \quad = \quad \arg\max_{i\in\{1,\ldots,M\}} \; (\lambda_i q(i))^{-\gamma} \; Y_i(n)$$

where

$$Y_i(n) = \begin{cases} \lambda_i w_i(n) & \text{for push-based delivery} \\ X_i(n) & \text{for pull-based delivery} \end{cases}$$

and $q(i)$ is the length of the items in slots. Let's call this class of policies "*Priority Index Policies With Length*" (*PIPWL*-γ). γ controls the relative importance of access probabilities, item length and (expected) request backlog for scheduling. When all $q(i)$'s are equal, PIPWL-γ reduces to the PIP-γ.

Simulation results are provided for 1000 items with Zipf distribution II and three types of distribution for the length of information items, $q(i)$'s which are given as follows:

1. **Uniform Distribution**: $q(i)$ is uniformly distributed between 1 and 10 for $i = 1, ..., M$.

2. **Increasing**: $q(i) = l_{min} + \frac{(i-1)(l_{max}-l_{min}+1)}{M}$, $i = 1, ..., M$ with $l_{min} = 1$ and $l_{max} = 10$.

3. **Decreasing**: $q(i) = l_{max} - \frac{(i-1)(l_{max}-l_{min}+1)}{M}$, $i = 1, ..., M$ with $l_{min} = 1$ and $l_{max} = 10$.

For pull-based delivery, we compare our PIPWL-γ policies to both PIP-0.5 (PIP with $\gamma = 0.5$) and LWF policies to investigate the effect of the value of γ and the length of the information item on the performance of the scheduling policies. PIPWL-0.5 yields the best mean response time for all the cases we consider. The mean response time of PIP-0.5, which does not take the length of information items into account, is about 6% to 24% more than that of PIPWL-0.5, which considers item lengths in scheduling. If the difference among the lengths of the items is large (here in this simulation setup, the maximum difference in item lengths is only 10 slots), the use of item length in scheduling can greatly improve the performance of the system. The mean response time of all the policies is the smallest for the case with increasing item lengths and the largest for the case with decreasing item lengths. Note that while $\lambda_i \geq \lambda_{i+1}$, $i = 1, ..., M-1$, for all access probability distributions, $q(i) \leq q(i+1)$ for increasing item lengths and $q(i) \geq q(i+1)$ for decreasing item lengths. Finally, as the request generation rates increase, the mean response time of all the policies (except LWF) for a pull-based system approaches that for a push-based system (See the figures 9-11 and the table 5). This fact further strengthens our conjecture presented in [18] that *the performance of a pull-based system approaches that of a push-based system under heavy load.*

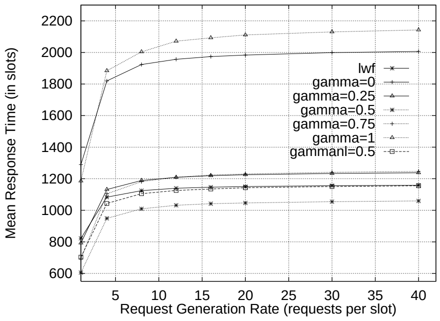

Figure 9: *Mean response time for the case with zipf distribution II and uniformly distributed lengths*

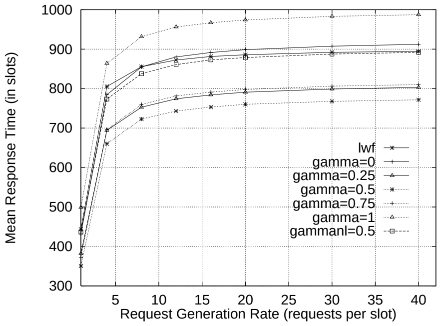

Figure 10: *Mean response time for the case with zipf distribution II and increasing lengths*

For push-based delivery, the performance of our PIPWL-γ policies is compared to PIP-0.5 and the lower bound for a periodic broadcast schedule which was given in [25] as

$$\frac{1}{2}\ \left(\sum_{i=1}^{M}\sqrt{b_i q(i)}\right)^2.$$

The results for 1000 items from the tables 5 show that the PIPWL-0.5 for a push-based

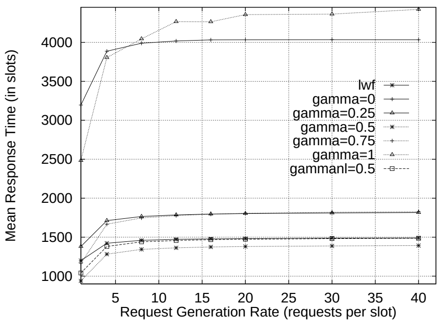

Figure 11: *Mean response time for the case with zipf distribution II and decreasing lengths*

system again yields the best performance which is also very close to the lower bound for all the cases.

Table 5: *Mean Response Time in* slots *for zipf distribution II and for different values of* γ *with equal access probabilities (L.B., U, I and D denote Lower Bound, Uniformly distributed length, increasing length and decreasing length respectively.*

γ	0	0.25	0.5	0.75	1.0	PIP-0.5	L.B.
U	2029.78	1250.04	1071.71	1261.51	2173.48	1167.17	1071.62
I	930.87	815.46	784.50	822.51	1000.30	904.88	784.47
D	4006.69	1827.90	1405.09	1847.16	4499.14	1497.57	1404.84

7 User's Memory Management

In a push-based delivery system, if a user has a local memory, it can alleviate its access latency or deadline misses of its requests by selectively prefetching the items from the broadcast and storing them in the memory.

Suppose the server broadcasts the information items according to a fixed predetermined schedule $\{u_n\}_{n=1}^{\infty}$, where u_n is the information item broadcast at slot n. Since the index of the broadcast schedule can be broadcast to the users ahead of the actual broadcast data either through the same channel in which the data is transmitted or through a different one with lower data transmission rate, we may assume that all the users know the whole broadcast schedule a priori.

A user is generating requests for information items according to his needs. When a request for some item i is generated at some time t, then it is either satisfied immediately if the item resides in the local cache of the user or the user has to wait until the next time this item will appear in the broadcast schedule. After the request is satisfied one way or the other, the user will generate another request for an information item after some random time.

The user has a memory that can hold K information items locally. At the end of each slot n, the user may replace one of the items in the memory with the item transmitted at slot n. We assume that all the users have identical memory update strategy. Since, in addition, all of them monitor the same broadcasting server, the contents of the memory of all users are identical. The set of the K information items residing in the memory during slot n is represented by $C(n)$. The memory update strategy determines the memory contents at each slot and is represented by the sequence $\{C(n)\}_{n=1}^{\infty}$.

We consider a particular group of users with an identical request generation process. If the user population though is large enough, then we may assume that the aggregate request generation rate remains constant and is independent of the number of pending requests while the process of request generation is stationary with rate λ. The request backlog for item i evolves as follows:

$$X_i(n+1) = \begin{cases} 0 & if \quad u_n = i \quad or \quad i \in C(n) \\ X_i(n) + A_i(n) & otherwise \end{cases}$$

A request for item i at time t will be satisfied immediately if $i \in C(\lfloor t \rfloor)$. If $i \notin C(\lfloor t \rfloor)$, then it will be satisfied at the end of the first item i broadcast that will be initiated after t. Let $v_i(t)$ be the amount of time from the end of slot n, where $t \in [n, n+1)$, until the beginning of the first slot after t at which item i is transmitted, which is illustrated in Figure 4. The latency $l_i(t)$ of a item i request generated at time t will be

$$l_i(t) = \begin{cases} 0 & \text{if } i \in C(\lfloor t \rfloor) \\ v_i(t) + 1 & \text{if } i \notin C(\lfloor t \rfloor) \end{cases}$$

The objective of a memory update strategy is to alleviate the impact of the latency on the user by maintaining in the memory the items which are either very likely to be requested by the user or they will not appear in the broadcast for a long time.

The sequence of times at which requests for item i are generated is $t_n^i, n = 1, 2, ...$ for each item $i = 1, ..., M$. Let $L_i(t)$ be the aggregate latency of all requests for item i generated from time 0 to time t,

$$L_i(t) = \sum_{t_n^i \leq t} l_i(t_n^i)$$

and $\overline{L_i}(t)$, be its expected value, $\overline{L_i}(t) \triangleq E[L_i(t)]$. Our objective is to design a cache management strategy which minimize the expected aggregate latency over all items, $\overline{L}(t) = \sum_{i=1}^{M} \overline{L_i}(t)$.

The key in obtaining the optimal cache update strategy is a transformation of the cost such that the impact of the cache update on the total latency becomes disjoint form slot to slot.

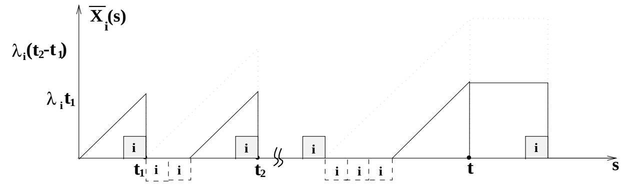

Figure 12: *The expected backlog of item i requests as a function of time when there is caching (solid line) is depicted. The expected backlog without caching (dotted line) is superimposed. The reduction in latency due to caching is equal to the area between the solid and the dotted lines.*

Figure 12 shows the evolution of the expected backlog of item i with caching (solid line) superimposed by the expected backlog without caching (dotted line) for a given sequence of item i broadcast. The shaded small rectangle corresponds to the slots at which item i is broadcast and the slots at which item i is cached are represented by the small dashed rectangles below the time axis. The aggregate expected latency for item i requests with caching (without caching) is equal to the total area under the solid (dotted) curve.

Let's denote by $\overline{L_i^\circ}(t)$ the expected latency of item i requests when there is no caching. The expected aggregate latency of item i requests under a caching strategy $\{C(n)\}_{n=1}^{\infty}$ can be related to the expected aggregate latency without caching as follows:

$$\overline{L_i}(t) = \overline{L_i^\circ}(t) - \sum_{n:n<t, i\in C(n)} (\lambda_i \tau_i^f(n) + \frac{\lambda_i}{2})$$

The above relation can be verified by using figure 12.

The aggregate latency over all items up to time t is

$$\overline{L}(t) = \overline{L^\circ}(t) - \sum_{n=0}^{t-1} \sum_{i\in C(n)} (\lambda_i \tau_i^f(n) + \frac{\lambda_i}{2})$$

The caching strategy that minimizes the aggregate latency is clearly the one that maximizes the following sum

$$\sum_{n=0}^{t-1} \sum_{i\in C(n)} (\lambda_i \tau_i^f(n) + \frac{\lambda_i}{2}) \tag{1}$$

The maximization of the sum in equation (1) is equivalent to a maximum reward path computation in an appropriately defined trellis diagram that captures the evolution of the cache states.

The cache state $C(n)$ at time n depends on the cache state $C(n-1)$ at time $n-1$, the item u_{n-1} broadcast during slot $n-1$, and the action taken by the update strategy. Hence, the set, $\boldsymbol{\mathcal{C}}_n(C)$, of possible cache states at slot n, given that the cache state at slot $n-1$ is C, is

$$\boldsymbol{\mathcal{C}}_n(C) = \{C' : C' \subset (C \cup \{u_{n-1}\}), |C'| = K\}$$

A feasible memory state evolution sequence is any sequence $\{C(n)\}_{n=0}^{\infty}$ with the property $C(n+1) \in \boldsymbol{\mathcal{C}}_{n+1}(C(n))$. The corresponding memory update strategy is uniquely defined.

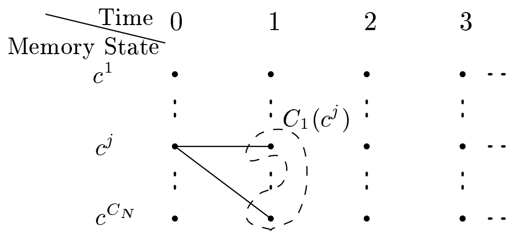

Figure 13: Memory State Trellis Diagram

A trellis diagram is considered one dimension of which is the memory state and the other is the time as shown in Fig.13. Each stage of the trellis corresponds to a certain time instant. All possible memory states appear in every stage. Since the total number of information items which are of interest to the user is M and its memory can hold K items ($K \leq M$), the total number of possible memory states, C_N, is $\binom{M}{K}$. There are directed links from certain states in stage n to certain states in stage $n+1$, that represent possible memory state transitions. Hence, a link is directed from state c^i at stage n to state c^j in stage $n+1$ if $c^j \in \boldsymbol{\mathcal{C}}_{n+1}(c^i)$.

Each memory update strategy from slot 0 to slot t corresponds to a path from stage 0 to stage t in the trellis. Associate with each state c^j, there is a time-dependent "*reward* ",

$$r(c^j, n) = \sum_{i \in c^j} (\lambda_i \tau_i^f(n) + \frac{\lambda_i}{2}),$$

which is equal to the expected latency of user's requests reduced by caching the set of items c^j in the memory. The total reward, or latency reduction incurred by an update strategy is equal to the sum of the reward of each state on the path. The computation of the optimal memory update strategy is equivalent to the computation of a maximum reward path in the trellis.

Although the computation of the optimal update strategy is feasible, the complexity of this approach is still prohibitive for employing the policy in the real time operation of a system. Hence, the value of the policy is mostly theoretical and it can be used as a benchmark for performance comparison with other policies. A class of policies with manageable complexity for real time operation is W-step look-ahead strategies. The optimal memory management policy makes the memory update decision at each slot n such that the total reward until time t is maximized. Instead of that, a *look-ahead* window W may be considered and the memory update decision at slot n can be made such that the cumulative average reward from slot n up to slot $n + W$ is maximized. As the window W increases, the complexity increases and the performance should be improved. The simplest policy of the W-step look-ahead class is the one with $W = 1$ which is equivalent to the PT heuristic proposed in [3]. This policy turns out to be optimal in some special cases of interest. The first case is when all the information items have the same access probabilities. The other case is when the broadcast includes only two information items, the user has memory space for one item and the request generation rates are arbitrary. When the request generation rate distribution though is skewed, the performance of one-step look-ahead policy may get considerably degraded compared to the optimal policy as we demonstrated by a counter example.

In certain cases, an information request needs to be satisfied within its deadline. The primary objective of a cache management policy in this case is to minimize the number of deadline misses. We also showed in [23] that the problem of finding an optimal cache update policy that minimizes the expected number of deadline misses can also be formulated in the same manner as the case without deadlines.

8 Joint Broadcast Scheduling and User's Cache Management

The problem of broadcast scheduling and user's cache management have been considered separately in previous sections. An integrated approach is presented in this section which jointly designs broadcast schedule and performs user's cache management in push-based delivery.

Let's assume that there are G groups of users each of which is made up of users with an identical request generation process. A request from group g is for item i with probability b_i^g, $i = 1, ..., M$ where $\sum_{i=1}^{M} b_i^g = 1$. Requests for item i from each group g are generated according to a stationary process with rate $\lambda_i^g = \lambda^g b_i^g$.

Assume that the server is transmitting data according to some arbitrary broadcast schedule. Assume that each user from group g has a local cache of size K_g and the users of group g are following an identical cache update strategy which determines the content of their caches.

Figure 14 shows the evolution of the expected item i request backlog of group g users, $\overline{X_i^g}(n)$, with caching (solid line) superimposed by the expected request backlog without caching (dotted line) for a given sequence of item i broadcasts. The small rectangles above the horizontal time axis correspond to the slots at which item i is broadcast. At the end of each item i broadcast, all of the requests are granted. The slots at which item i resides in the cache are represented by the small dashed rectangles below the time axis. All of the requests arriving during these slots are granted immediately. The expected item i request backlog of group g, $\overline{X_i^g}(s)$, starts increasing with rate λ_i^g either from the end of the last item i broadcast before s or from the end of last item i residence in the cache before s, whichever happened last. The aggregate expected delay of item i requests with caching (without caching) is equal to the total area under the solid (dotted) curve. The area between solid and dotted curve represents the reduction in the aggregate expected delay of item i requests due to caching.

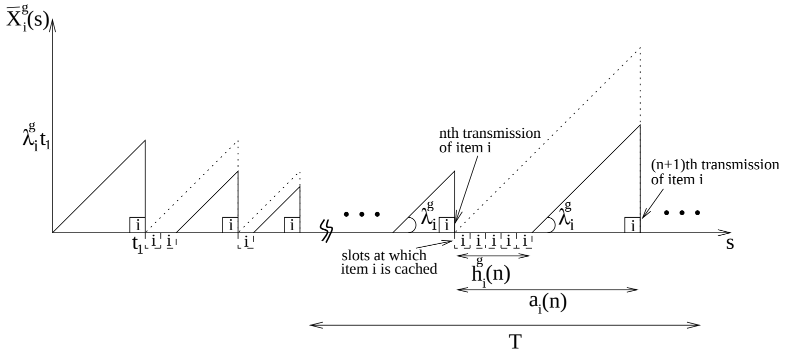

Figure 14: Expected item i request backlog of group g users with caching (solid line) and without caching (dotted line) as a function of time.

The first step for joint design of the server broadcast schedule and the user caching strategy is to lower-bound the average delay of item i requests under a certain broadcast scheduling policy and under a certain cache management policy by the expression made up of average values of the component parameters. The minimum of the lowerbound over all possible values of these average parameters is a lowerbound of any broadcast scheduling and any cache management policy.

Consider an interval of duration T as shown in figure 14. Let N_i^T be the total number of transmissions of item i, $i = 1, ..., M$, during the interval T. Denote the inter-appearance gap between the nth transmission and $(n+1)$th transmission of item i during the interval T by $a_i(n)$. Assume that a user from group g keeps item i for $h_i^g(n)$ slots after the end of the nth transmission where $h_i^g(n) = 0, 1, ..., a_i(n)$ as shown in figure 14.

Consider the class of broadcast schedules where the limit of the average inter-appearance gap of each item i, $\overline{a_i}$, converges, which is defined as follows:

$$\overline{a_i} = \lim_{T\to\infty} \frac{1}{N_i^T} \sum_{n=1}^{N_i^T} a_i(n)$$

The fraction of time item i is transmitted (average transmission frequency) in the schedule, $\overline{f_i}$, is equal to the inverse of its average inter-appearance gap and is given as follows:

$$\overline{f_i} = \frac{1}{\overline{a_i}} = \lim_{T\to\infty} \frac{N_i^T}{\sum_{n=1}^{N_i^T} a_i(n)} = \lim_{T\to\infty} \frac{N_i^T}{T}$$

Also consider the class of cache management policies where the limit of the average caching time of item i by a user from group g, $\overline{h_i^g}$, exists, which is defined as follows:

$$\overline{h_i^g} = \lim_{T\to\infty} \frac{1}{N_i^T} \sum_{n=1}^{N_i^T} h_i^g(n)$$

The expected delay of item i requests from group g with caching between the nth and $(n+1)$th transmission of item i is equal to the area of the solid line triangle between the nth and $(n+1)$th transmission of item i shown in figure 14. Hence, the long-term average aggregate delay of item i requests from group g, D_i^g, with caching is given by

$$D_i^g = \lim_{T\to\infty} \frac{1}{T} \frac{1}{2} \sum_{n=1}^{N_i^T} \lambda_i^g \left(a_i(n) - h_i^g(n)\right)^2 .$$

It can be easily proved that the lowerbound of the long-term average aggregate delay of all the requests experienced by a user is given by

$$\frac{1}{2G}\sum_{g=1}^{G}\sum_{i=1}^{M}\lambda_i^g\overline{f_i}\left(\frac{1}{\overline{f_i}}-\overline{h_i^g}\right)^2$$

Consider the following optimization problem (OP).

$$OP: \qquad D_{avg}^* = \min_{\overline{f_i},\overline{h_i^g},i=1,...,M,g=1,...,G} \frac{1}{2G}\sum_{g=1}^{G}\sum_{i=1}^{M}\lambda_i^g\overline{f_i}\left(\frac{1}{\overline{f_i}}-\overline{h_i^g}\right)^2 \tag{2}$$

$$\text{s.t.} \quad \sum_{i=1}^{M}\overline{f_i} \le 1 \tag{3}$$

$$\sum_{i=1}^{M}\overline{f_i}\,\overline{h_i^g} \le K_g, \quad g=1,...,G \tag{4}$$

The first constraint requires that the server cannot transmit more than one item in each slot and the second constraint says that we cannot store more items than the cache capacity of a user.

The long-term average aggregate delay of all the requests experienced by a user under any broadcast scheduling and cache management policy is lowerbounded by the solution of the above optimization problem (OP). Moreover, for a given broadcast schedule with known average transmission frequencies f_i's, the long-term average aggregate delay of all the requests experienced by a user under any cache management strategy is lowerbounded by

$$\frac{1}{2G}\sum_{g=1}^{G}\frac{(M-K_g)^2}{\sum_{i=1}^{M}\frac{f_i}{\lambda_i^g}}.$$

This lowerbound can be used as a benchmark to investigate of performance of a cache management policy. In the following, we will provide a way to jointly schedule broadcast and do cache management using the parameters from the above optimization problem (OP).

One approach for joint broadcast scheduling and cache management is to construct a broadcast schedule according to the optimal transmission frequencies obtained from the optimization, $\overline{f_i}^*$, $i=1,...,M$, and to follow an optimal cache management policy [23] under the above-designed broadcast schedule.

Since usually there are more than one group with different user's access statistics, the scheduling algorithm transforms the set of user's access statistics into one aggregate access statistics which is used for scheduling as depicted in figure 3. One simple way is to take the average of all the access statistics and this is what most of the scheduling algorithms do. That is,

$$b_i = \frac{1}{G}\sum_{g=1}^{G} b_i^g. \tag{5}$$

Joint broadcast scheduling and user's cache management suggests another approach for obtaining the aggregate user's access statistics. Under the joint approach, the optimal transmission frequencies $\overline{f_i}^*$'s are first obtained from the optimization and the aggregate user's access probabilities are computed from the $\overline{f_i}^*$'s as follows [7]:

$$b_i = \frac{\overline{f_i}^{*2}}{\sum_{j=1}^{M} \overline{f_j}^{*2}} \tag{6}$$

According to the numerical results, the MAD policy consistently constructs a broadcast schedule with average transmission frequencies which are very close to the required ones. Therefore, it is employed to generate broadcast schedule by using user's access probabilities, b_i's which can be obtained from the optimal transmission frequencies, $\overline{f_i}$'s according to equation (6).

Due to high implementation complexity, the optimal cache management provided in [23] is not practical. As the numerical examples in [23] showed that the low-complexity *One-Step Look-Ahead* (*OSLA*) strategy yields performance comparable the optimal policy in many cases, we apply the OSLA scheme for cache management in our numerical experiments.

The performance of our Joint Scheduling and Cache Management (JSCM) approach is compared to that of the Non-Joint Scheduling and Cache Management (Non-JSCM) scheme. The only difference between JSCM and Non-JSCM is user's access probabilities, b_i's, which is the main input parameter to the broadcast scheduling algorithm. In JSCM, b_i's are computed from the average transmission frequencies, $\overline{f_i}$'s, by using equation (6), which are obtained from the optimization problem (OP). In Non-JSCM, b_i's are obtained by taking the average of the given access probabilities over different groups as in equation (5). Then, we employ the OSLA as our cache management strategy under both schedules generated by JSCM and Non-JSCM and compare the mean response time of two approaches.

Numerical experiments are performed for 100 items. Items are numbered from 1 to 100 and users from each group g are assumed to be interested only in M_g consecutive items starting from *Start* $Item_g$ up to (*Start* $Item_g + M_g - 1$) mod $(M+1) \ + \ 1$. For these M_g items, user's access probabilities are assumed to follow zipf distribution [15]: $\tilde{b}_j = \frac{j^\theta - (j-1)^\theta}{M^\theta}$ where $\theta = 0.025$ and $j = 1, ..., M_g$.

For the JSCM approach, the optimization problem (*OP*) is solved by using the Feasible Sequential Quadratic Programming (FSQP) software [24]. The transmission frequencies of the items are converted into access probabilities by equation (6) which are then used as an input to the MAD algorithm. Under the broadcast schedule generated by the MAD, the mean response time of user's requests is evaluated by exercising the OSLA strategy for cache management for each user.

The numerical results in the example compare the mean response time (MRT) of JSCM and Non-JSCM. The first and second big column shows the mean response time without user's caching and the mean response time reduced by caching with the OSLA strategy respectively. The reduction in the mean response time due to caching under any cache management strategy is equal to the aggregate sum of the reward of caching for the items stored in the cache at each slot over the one period of the broadcast schedule [23]. The third column shows the actual mean response time experienced by a user with caching using the OSLA policy. The "*Optimized*" column gives the results under JSCM while the "*Average*" column shows the results for Non-JSCM. The last column shows the % improvement in the mean response time over JSCM by the use of Non-JSCM.

A Numerical Example

No. Of Groups = 5,

$K_1 = 2$, $M_1 = 20$, *Start* $Item_1 = 1$, (1,...,20), and in *decreasing* order.

$K_2 = 4$, $M_2 = 20$, *Start* $Item_2 = 21$, (21,...,40), and in *decreasing* order.

$K_3 = 6$, $M_2 = 20$, *Start* $Item_3 = 41$, (41,...,60), and in *decreasing* order.

$K_4 = 8$, $M_2 = 20$, *Start* $Item_4 = 61$, (61,...,80), and in *decreasing* order.

$K_5 = 10$, $M_2 = 20$, *Start* $Item_5 = 81$, (81,...,100), and in *decreasing* order.

	MRT Without Caching		MRT Reduced by Caching		MRT Actually Experienced		% Improvement over the use of Average Parameters
	Optimized	Average	Optimized	Average	Optimized	Average	
Group 1	25.21	10.49	22.47	5.90	2.74	4.59	40.31%
Group 2	28.49	10.49	26.40	6.70	2.08	3.49	40.40%
Group 3	31.47	10.49	29.91	7.83	1.56	2.66	41.35%
Group 4	29.76	10.49	28.61	8.51	1.16	1.97	41.12%
Group 5	28.46	10.49	27.64	9.09	0.82	1.40	41.43%
Average	28.68	10.49	27.01	7.61	1.67	2.82	40.78%

With JSCM, MRT without caching is quite large but a large portion of MRT is reduced by caching. Although MRT without caching for Non-JSCM is smaller than that for JSCM in all cases, caching reduces only a smaller portion of MRT for Non-JSCM than JSCM. Therefore, the improvement in the actual MRT experienced by a user with caching over Non-JSCM by JSCM is up to 40%. It shows that considerably better performance can be achieved if we design broadcast schedules which are more amenable to user's caching.

9 Multistage Information Delivery Systems

Satellite broadcast data delivery plays a key role in disseminating information to the geographically scattered groups of users in the recently proposed National/Global Information Infrastructure. Since different groups of users may have interest in different types of information and bigger antennas and receivers of higher complexity are needed to receive information directly from the satellite broadcast, one feasible alternative to direct satellite broadcast to the users is multistage information delivery architecture, as shown in figure 15, where the satellite transmission is received first by a local ground station, which redistributes the data to the users through a terrestrial wireless system.

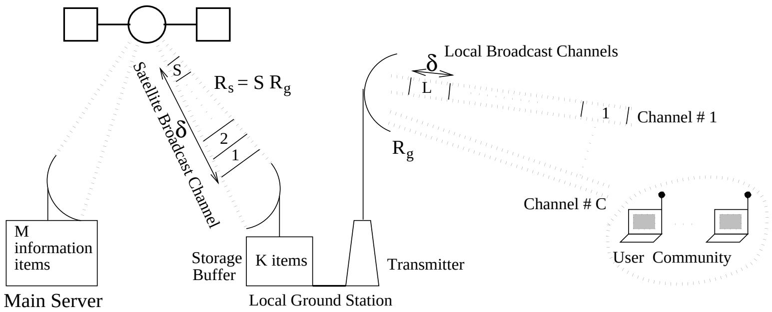

Figure 15: *Multistage Information Distribution Scenario*

The main server is transmitting information items through a satellite to geographically scattered ground stations with transmission rate R_s items per second. Each ground station is receiving the information items broadcast by the satellite and has a local storage buffer which can hold K items. Each ground station is retransmitting, through C local broadcast channels, the information items stored in the local buffer to a local user community, which is listening to the broadcast of the local ground station, with transmission rate R_g items per seconds where $R_s = S\ R_g$, S is a positive integer. This is a reasonable assumption since, usually, transmission through a satellite has a much larger bandwidth than local transmission.

Time on the local broadcast channel is divided into slots of equal size (δ seconds as depicted in figure 15) and the slot length represents the time to broadcast an information item by the local ground station at the local broadcast. At each slot n, the main server is transmitting a set of S different items under a given broadcast schedule. At the same time, the local ground station is transmitting a set of C items out of K items stored in the local buffer, through C local broadcast channels. As S items are arriving on the satellite broadcast channel, the local ground station has to decide whether it will prefetch some or all of the items or not and if it will, which items residing in the local buffer will be replaced with the newly prefetched items. Hence, a good joint buffer management and broadcast scheduling is needed in order to minimize the latency or deadline misses of user's requests.

In [12], a two-stage information delivery system was considered and for a given satellite broadcast schedule, two heuristics for buffer management at the local station were proposed which tries to optimize the buffer management when the broadcast schedule at the location tries to follow the MAD policy. Two approaches differs in the steps they take in joint buffer management and broadcast scheduling. The first heuristic determines the content of the local buffer based on the satellite's schedule before constructing the local's schedule while the second approach determines the schedule based on the local user's access statistics and then tries to manage the local buffer such that the desired local schedule can be followed.

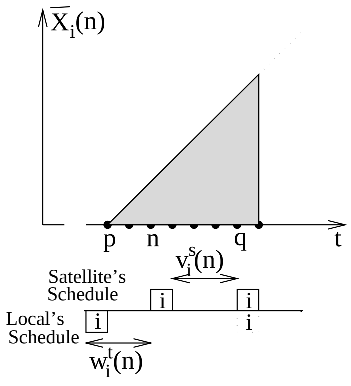

Figure 16: *The evolution of expected request backlog of item i under the given satellite schedule. Small rectangles above the line represent the transmission of item i under satellite's schedule while those below the line are the item i broadcasts under the local's schedule. The dotted rectangle represents the possible item i broadcast after the slot n at the local station*

$w_i^t(n)$ is the elapsed time since the beginning of last item i transmission under the local's schedule and $v_i^s(n)$ is the number of slots from the end of slot n to the beginning of the next item i transmission after n under the satellite's schedule, as defined in figure 4. Suppose item i is broadcast by the satellite in slot n and slot q under the given satellite's schedule and broadcast by the local station in slot $p-1$.

Under this scenario, if item i is not currently stored in the local buffer, the next earliest slot after slot n the local station can broadcast is slot q. At slot n, the first approach computes the shaded area of the triangle shown in figure 16 for each item currently stored in the buffer as well as the item arriving on the satellite broadcast. If the arriving item has the bigger value, it replaces the item stored in the buffer with the smallest value. Otherwise, the buffer is kept unchanged. Then, an item is selected among the items in the buffer according to the MAD policy.

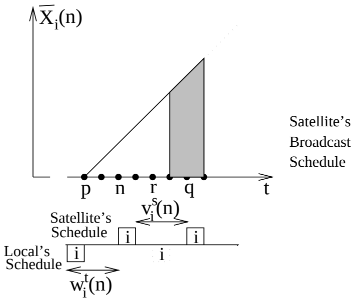

Figure 17: *The evolution of expected request backlog of item i under the given satellite schedule. Small rectangles above the line represent the transmission of item i under satellite's schedule while those below the line are the item i broadcasts under the local's schedule. The dotted rectangle represents the item i broadcast after the slot n under the desired schedule at the local station*

The second heuristic first computes the next transmission of each item according to the MAD policy based on the current buffer content and the items arriving on the satellite broadcast during that period. Then under this desired schedule, the shaded area in figure 17 is computed for each item currently stored in the buffer as well as the item arriving on the satellite broadcast. Then the item with the smallest value is discarded and the MAD policy is applied to the items in the buffer for the next broadcast schedule.

10 Future Research

In push/pull hybrid delivery, there exists an uplink feedback channel of some capacity available and users can submit requests for a certain portion of the server's database. Usually, a user sends a request to the server either if the item is specified as a pull-based item or if the time it has been waiting for the item to appear in the broadcast exceeds some pre-specified threshold. User's access statistics cannot be assumed to be known in many cases and have to be estimated based on the number of requests received from the uplink request channel. In this case, the information items have to be *dynamically* specified either as *push-based items* or as *pull-based items* based on the number of requests received. What makes the problem more complicated is that once the item has been *declared* to be a push-based item, the server will not observe the requests for the item. Therefore, the dynamic partitioning of the server's database (into push-based and pull-based sets) is required and it has to be adaptive to the changes in user's access pattern. Designing an efficient, adaptive push/pull hybrid delivery system is one of the major challenge in broadcast delivery.

Most of the previous research in broadcast delivery has focused on minimizing the mean response time of user's requests. One possible objective will be to minimize the *maximum* response time of user's requests instead of average access delay. In many applications, user's requests have to be satisfied within some deadlines. Consider a video-on demand system with batching as an example of pull-based delivery where viewers may turn away when the waiting time is more than they can tolerate. The objective in these applications should be to minimize the average number of deadline misses and most of the previous results may be different for this objective. Therefore, another future research task in broadcast delivery is to consider the problem in terms of other useful objectives.

One of the implicit assumptions in this work is that the communication channel is *error-free* and each user always receives the item broadcast by the server correctly. When the broadcast medium is wireless, the assumption is not necessarily valid. The problem of distributing information by broadcast delivery in error-prone wireless environment is one of the future research issues worth considering.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. "Broadcast Disks: Data Man-

agement for Asymmetric Communication Environments". Technical Report CS-94-43, Dept. of Comp. Science, Brown University, October 1994.

[2] S. Acharya, M. Franklin, and S. Zdonik. "Dissemination-based Data Delivery Using Broadcast Disks". *IEEE Personal Communications*, 2(6):50–60, December 1995.

[3] S. Acharya, M. Franklin, and S. Zdonik. "Prefetching from a Broadcast Disk". In *Proc. 12th Int'l. Conf. Data Eng.*, New Orleans, LA, February 1996.

[4] S. Acharya, M. Franklin, and S. Zdonik. "Balancing Push and Pull for Data Broadcast". In *Proc. ACM SIGMOND*, Tuscon, AZ, May 1997.

[5] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. The Maximum Factor Queue Length Batching Scheme for Video-on-Demand Systems. IBM Research Report RC 20635, IBM, T.J. Watson Research Center, Yorktown Heights, New York, November 1996.

[6] M. H. Ammar. "Response Time in a Teletext System: an Individual User's Perspective". *IEEE Transaction on Communication*, COM-35(11):1159–1170, November 1987.

[7] M. H. Ammar and J. W. Wong. "The Design of Teletext Broadcast Cycles". *Performance Evaluation*, 5(4):235–242, December 1985.

[8] M. H. Ammar and J. W. Wong. "On the Optimality of Cyclic Transmission in Teletext Systems". *IEEE Transaction on Communication*, COM-35(1):68–73, January 1987.

[9] T. Chiueh. "Scheduling for Broadcast-based File Systems". *Proc. of the Mobidata Workshop*, November 1994. Rutgers University, NJ.

[10] A. Dan, D. Sitaram, and P. Shahabuddin. "Scheduling Policies for an On-Demand Video Server with Batching". *Proc. of Second ACM Intl. Conf. on Multimedia*, pages 15–23, October 1994.

[11] H. D. Dykeman, M. H. Ammar, and J. W. Wong. "Scheduling Algorithms for Videotex System under Broadcast Delivery". *Proceedings of ICC' 86*, pages 1847–1851, 1986.

[12] Ozgur Ercetin and Leandros Tassiulas. "Information Delivery in two-stage Satellite-Terrestrial Wireless Systems ". In *Proc. 32nd Annual Conf. on Information Sciences and Systems*, Princeton, New Jersy, March 1998.

[13] J. Gecsei. *"The Architecture of Videotex Systems"*. Prentice-Hall, Inc., Englewood Cliffs, N.J.07632, 1983.

[14] S. Hameed and N. H. Vaidya. Log-time Algorithms for Scheduling Single and Multiple Channel Data Broadcast. In *Proc. of MOBICOM'97*, Budapest, Hungary, September 1997.

[15] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, Massachusetts, second edition, 1981.

[16] Hadas Shachnai and Philip S. Yu. "The Role of Wait Tolerance in Effective Batching: A Paradigm for Multimedia Scheduling Schemes". Technical Report RC 20038 (88607), IBM Research Report, April 1995.

[17] K. Stathatos, N. Roussopoulos, and J. S. Baras. "Adaptive Data Broadcast in Hybrid Networks". Technical report, Center for Satellite and Hybrid Communication Networks, University of Maryland, College Park, April 1997.

[18] C. J. Su and L. Tassiulas. "Broadcast Scheduling for Information Distribution". In *Proc. IEEE INFOCOM'97*, volume I, pages 109–117, Kobe, Japan, 1997.

[19] C. J. Su and L. Tassiulas. "Broadcast Scheduling for the Distribution of Information Items with Unequal Length". In *Proc. 31th Conf. on Information Science and Systems*, The Johns Hopkins University, MD, 1997.

[20] C. J. Su and L. Tassiulas. "Broadcast Scheduling for Information Distribution". *Wireless Networks Journal*, 1998. Accepted for publication.

[21] C. J. Su and L. Tassiulas. "Joint Broadcast Scheduling and User's Cache Management for Efficient Information Delivery". In *Proc. MobiCom'98*, pages 33–42, Dallas, Texas, October 1998.

[22] Chi-Jiun Su. *Information Distribution through Broadcast Delivery.* PhD dissertation, Polytechnic University, Brooklyn New York, Electrical Engineering Department, January 1998.

[23] L. Tassiulas and C. J. Su. "Optimal Memory Management Strategies for a Mobile User in a Broadcast Data Delivery System". *IEEE JSAC Special Issue on Networking and*

Performance Issues of Personal Mobile Communications, 15(7):1226–1238, September 1997.

[24] Andri Tits. "FSQP". In *URL, http://www.isr.umd.edu/Labs/CACSE/FSQP/fsqp.html.*

[25] N. H. Vaidya and S. Hameed. Data Broadcast in Asymmetric Wireless Environments. In *Proc. of Workshop on Satellite-based Information Services (WOSBIS)*, Rye, New York, November 1996.

[26] S. R. Viswanathan. *"Publishing in Wireless and Wireline Environments"*. PhD dissertation, Rutgers University, Department of Computer Science, November 1994.

[27] J. W. Wong and H. D. Dykeman. "Architecture and Performance of Large Scale Information Delivery Networks". *Proc. of 12th International Teletraffic Congress*, pages 440–446, 1989.

[28] G. K. Zipf. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley, Reading, Massachusetts, 1949.