Optimal Memory Management Strategies For a Mobile User in a Broadcast Data Delivery System

Leandros Tassiulas, Chi-Jiun Su

Manuscript received September 17 1996; revised April 1 1997. This research was supported in part by NSF under grant NCR-9406415, a CAREER award NCR-9502614 and by the AFOSR under grant 95-1-0061.

L. Tassiulas is with the Department of Electrical Engineering, University of Maryland, College Park, MD 20742 USA.

C.-J. Su is with the Department of Electrical Engineering, Polytechnic University, Brooklyn, NY 11201 USA and is currently visiting the University of Maryland, College Park .

Abstract

Recently data broadcasting has been considered as a promising way of disseminating information to a massive number of users in a wireless communication environment. In a broadcast data delivery system, there is a server which is broadcasting data to a user community. Due to the lack of communication from users to the server, the server cannot know what a user needs. In order to access a certain item, a user has to wait until the item appears in the broadcast. The waiting time will be considerably long if the server's broadcast schedule does not match the user's access needs. If a user has a 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 which is a major concern in a broadcast data delivery system. In this paper, an optimal memory management policy is identified, that minimizes the expected aggregate latency. We present optimal memory update strategies with limited look-ahead as implementable approximations of the optimal policy. Some interesting special cases are given for which the limited look-ahead policies are optimal. We also show 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.

Keywords

Broadcast Data Delivery, Client Prefetching, Information Distribution Systems, Local Memory Management, Mobile Computing.

I. Introduction

Data broadcasting is a candidate to play a leading role for data delivery in a wireless asymmetric communication environment since it reduces the relatively expensive client-to-sever communication and it is scalable in such a way that it is independent of the number of users the server is serving. It has been proposed for both wireline and wireless environment. The Direct Broadcast Satellite (DBS) system is a good example to which broadcast data delivery can be effectively applied. Some application examples include the distribution of time-sensitive information such as stock prices, traffic and weather information, and mail-order catalog and mutual fund information for information retrieval services. In the *Data-cycle* projects at Bellcore [1], a database circulates on a high bandwidth network (140 $Mbps$) and users query this data by filtering relevant information using a special massively parallel transceiver. Wireless data broadcasting was also considered as an efficient way, in terms of energy and bandwidth, for disseminating information to a massive number of users in [2], [3], [4], and [5].

Under the broadcasting approach as shown in Fig.1, a server continuously and repeatedly broadcasts data to a user community without any feedback about the user's needs due to the limited uplink communication capability from the user to the server. Data broadcast by the server are 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. There is some latency from the time the need of an information item arises until the time the item is actually broadcast by the server. This latency depends on the broadcast schedule of the server, as well as the user access pattern.

There has been a lot of work in the past on the problem of designing a broadcast schedule such that the average latency is minimized [6], [7], [8], [9], [10] and [11]. The approach is to determine the broadcast

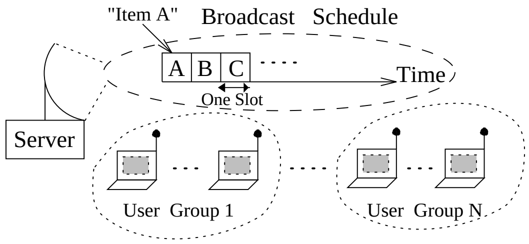

Fig. 1. A Broadcast Data Delivery System in a Wireless Communication Environment

frequency of each information item in accordance with the user access frequency of the item and then to distribute the broadcast slots of each item as uniformly as possible. 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 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. Therefore, user's memory management becomes an important issue to consider in order to minimize the average response time of user's requests. As information items pass by in the broadcast, the user has to decide whether an item will be prefetched and if it will, which item residing in the memory will be replaced with the newly prefetched item.

There are basically two different ways of caching: *demand-driven* caching, in which users obtain data from the broadcast only as a result of memory fault, and *prefetching*, in which users bring data from the broadcast in advance of any requests. Acharya et al. proposed a simple information item replacement policy called P and a cost-based demand-driven caching heuristic, called *PIX*, in [9]. P is a demand-driven caching policy which keeps the items with the highest probability of access in the memory. In *PIX*, the cost of replacement of an item already in the memory with the newly fetched one was considered to be the ratio of the access probability of the information item (P) and its broadcast frequency (X). The ratio is called the *pix* value. The *PIX* replacement algorithm ejects the item in the memory which has the lowest *pix* value. Demand-driven caching, however, does not fully exploit the dissemination-based nature of the broadcast, which is particularly conducive to user's prefetching as claimed in [12]. A simple prefetching heuristic, called PT, was proposed in [12]. The PT computes the value of an item by taking the product of the access probability of the information item (P) with the time (T) that will elapse before that item appears on the broadcast again. This is called the item's *pt* value. PT finds the item in the memory with the lowest *pt* value, and replaces it with the currently broadcast item if the latter has a higher *pt* value. In [13], Ammar considered a teletext broadcast delivery system with correlated user's requests from user's perspective and proposed a *Linked Items* prefetching scheme. The scheme requires the availability of space in each information item to store control information. The control information in item i is a list of linked items, the items that are most likely to be requested next by a user. After a request for an item i is satisfied, the user enters a phase to prefetch the D most likely referenced items associated with item i, where D is the size of the local memory in units of information items. This phase is terminated when the D items are fetched or when the user submits a new request.

In this paper, optimal memory update strategies are identified. The broadcast data dissemination model is considered in section II. In section III, we identify an optimal memory update policy that minimizes the expected aggregate latency over all information items. A memory state trellis diagram is identified and it is shown that the computation of the optimal memory update policy is equivalent to the computation of the maximum reward path in the trellis. Computational issues are also discussed and the memory update policies with limited look-ahead are given as implementable alternatives to the optimal strategy. In sections IV-A, IV-B and IV-C, some interesting special cases are considered for which limited look-ahead policies are optimal. Some numerical examples are given in section IV-D to show the trade-off between the performance and implementation complexity. Finally, we consider in section V the problem of minimizing deadline misses when requests for information items need to be satisfied within some specified time periods.

II. Broadcast Data Delivery

Time on the broadcast channel is divided into slots of equal size and the slot length represents the time to broadcast an information item. Slot n corresponds to the time interval $[n, n+1)$. The server broadcasts the information items according to a fixed predetermined schedule $\{u_n\}_{n=0}^{\infty}$, where u_n is the information item broadcast at slot n. Assume that there are M possible information items. An important class of schedules for the applications is the periodic schedule where the transmission sequence is repeated after some period T and the schedule is completely specified by its period $\{u_0, u_1, ..., u_{T-1}\}$. During each period, each information item is transmitted at least once and possibly multiple times. It has already been proved in [7] that optimal schedules which minimize the user's expected access latency are *periodic*. The directory of a broadcast schedule (the index of the broadcast data) 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. In [14], [3] and [15], a number of methods have been proposed to multiplex the directory information together with the data on the same channel. Therefore, in the following, we may assume that all the users know the whole broadcast schedule a priori.

A user is generating requests for information items according to its 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 the item appears 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 latency from the time a request is generated until the item is transmitted by the server is the performance measure of interest in data broadcast delivery.

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. In the finite user population case, the rate of request generation is affected by the number of users who are waiting for an information item broadcast by the server. Since they will not generate a new request while they are waiting, the rate of request generation will drop as the number of pending requests increases. If the user population though is large enough and an individual user request generation rate is appropriately normalized such that the aggregate rate is equal to λ, 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. In the case of a nonhomogeneous user community with more than one distribution of requests for information items, the results of the paper hold for each group of users with identical request distribution.

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 $\lambda_i = \lambda b_i$. Let $A_i(n)$ be the total number of requests for item i occurred during slot n. Let $X_i(n)$ be the number of users requesting item i at the beginning of slot n. The request backlog for item i evolves as follows:

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

A request for item i generated 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 is initiated after t. Let $\tau_i^f(t)$ be the amount of time from t until the beginning of the first slot after t at which item i is transmitted, which is illustrated in Fig.3. The latency $l_i(t)$ of a request for item i generated at time t will be

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

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 more likely to be requested by the user or which will not appear in the broadcast for a long time. We consider two different performance measures of a memory update strategy. One is the average aggregate latency of user's requests and the other is the fraction of user's requests which are not satisfied within a pre-specified time period.

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, $\overline{L_i}(t) \triangleq E[L_i(t)]$. The aggregate expected latency over all items is

$$\overline{L}(t) = \sum_{i=1}^{M} \overline{L_i}(t)$$

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

In certain cases, an information request needs to be satisfied within a certain time period from the time it is generated. That time period is the "*deadline*" of the request. In this case, what it really matters is whether the request will be satisfied within the deadline or not, while the average latency is not of primary importance. There is a distinct set of deadlines corresponding to each item i, $\{d_i^1, d_i^2, ..., d_i^{a_i}\}$ where a_i is the total number of possible deadlines for item i and a request for item i is generated by a user with one of these deadlines. Assume that we have a stationary stream of requests with rate λ_i^j for item i with deadline d_i^j, where $\lambda_i = \sum_{j=1}^{a_i} \lambda_i^j$. Let $q_i^j(t)$ be a random variable which denotes whether a type j item i request generated at time t misses its deadline d_i^j or not and it is given by

$$q_i^j(t) = \begin{cases} 1 & \text{if } l_i(t) > d_i^j \\ 0 & \text{otherwise} \end{cases}$$

Let $Q_i^j(t)$ be the aggregate deadline misses of all type j item i requests generated from time 0 to time t,

$$Q_i^j(t) = \sum_{t_n^i \leq t} q_i^j(t_n^i)$$

and $\overline{Q_i^j}(t)$, be its expected value, $\overline{Q_i^j}(t) \triangleq E[Q_i^j(t)]$. The aggregate expected deadline misses over all information items is

$$\overline{Q}(t) = \sum_{i=1}^{M} \sum_{j=1}^{a_i} \overline{Q_i^j}(t)$$

and a possible decision criterion of a memory management strategy is to minimize $\overline{Q}(t)$.

In section III, we consider the problem of designing a memory management strategy which minimize the expected aggregate latency over all items, $\overline{L}(t)$, and the case of requests for information items with deadlines is discussed in section V.

III. Memory Update To Minimize Latency

The key in obtaining the optimal memory update strategy is the transformation of the cost such that the impact of the memory update on the total latency becomes disjoint from slot to slot. The first step for this transformation is the following lemma. For simplicity, let's assume, in the rest of the section, that the time t is an integer.

Lemma 1: The aggregate latency of item i requests is related to the item i backlog as follows:

$$L_i(t) = \int_0^t X_i(s)ds + X_i(t)\left(\tau_i^f(t) + 1\right) \tag{1}$$

Proof: The result in the lemma is of the same flavor as *Little's* law and is better shown using a pictorial argument. A sample path of the evolution of item i request generation is shown in Fig.2. The request generation instants correspond to the jumps of the curve which are of magnitude 1. The latency of a request for an item is the amount of time from its generation time instant to the end of the first transmission of the item. Hence, the aggregate latency equals the total area under the curve in Fig.2. Note that the sum in the right side of equation 1 is equal to that area where the first term corresponds to the area from time 0 to t and the second term to the remaining rectangle, extending from time t to $t + \tau_i^f(t) + 1$. $\diamond$

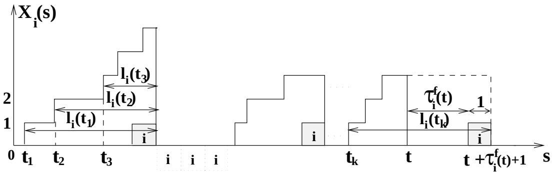

Fig. 2. The evolution of the item i request backlog is depicted as a func tion of time. At the end of each item i broadcast, the small shaded rectangle, all the requests are granted. The small dotted line rectangles below the time axis correspond to the slots at which item i is stored in the memory by the user. All the requests arriving during these slots are granted immediately. The latency $l_i(t_k)$ of the item i request generated at time t_k is also depicted.

By taking expectation on both sides of equation (1), the expected latency is

$$\overline{L_i}(t) = \int_0^t \overline{X_i}(s)ds + \overline{X_i}(t)\left(\tau_i^f(t) + 1\right)$$

where $\overline{X_i}(s) \triangleq E[X_i(s)]$. An analytic expression for $\overline{X_i}(s)$ is obtained in the following.

Let $\tau_i^b(t)$ be the amount of time from the end of the last slot before t at which item i was transmitted until time t. Let $\tau_i^c(t)$ be the amount of time from the end of the last slot before t at which item i was residing in the memory of the user until time t. If item i is residing in the memory of the user at time t, then $\tau_i^c(t) = 0$ by definition. The quantities, $\tau_i^b(t)$ and $\tau_i^c(t)$, are illustrated in Fig.3. Note that the pending requests for item i, $X_i(s)$, are accumulated starting either from the end of the last item i transmission before s or from the end of the most recent item i residence in the memory before s, whichever happened last. Since information items are generated by a stationary process with rate λ_i, we have $\overline{X_i}(s) = \lambda_i \min(\tau_i^b(s), \tau_i^c(s))$.

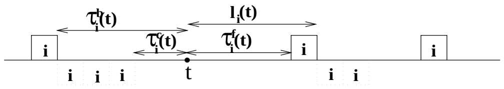

Fig. 3. Illustration of parameters, $\tau_i^f(t)$, $\tau_i^b(t)$, $\tau_i^c(t)$ and $l_i(t)$ for a sequence of item i broadcasts.

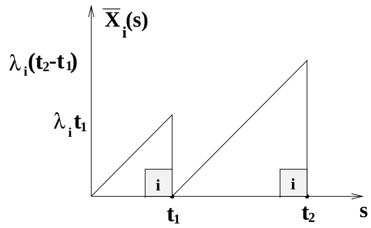

Fig. 4. The expected backlog as a function of time when there is no use of local memory

In Fig.4, we see the evolution of the expected backlog of item i for a certain sequence of item i broadcasts when there is no use of local memory.

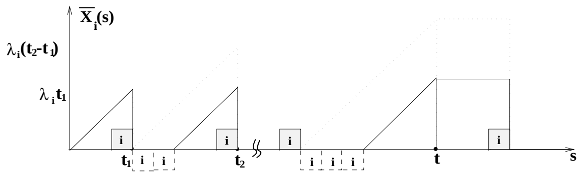

Fig. 5. The expected backlog as a function of time when there is caching (solid line) is depicted. The expected backlog without caching (dotted line) is superimposed. The reduction in latency due to caching is equal to the area between the solid and the dotted lines.

Fig.5 shows the evolution of the expected backlog with caching (solid line) superimposed by the expected backlog without caching (dotted line) for a given sequence of item i broadcast. The small shaded rectangle corresponds to the slots at which item i is broadcast and the slots at which item i is residing in the memory are represented by the small dotted rectangles below the time axis. The aggregate expected latency 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 when there is no caching. The reduction of the latency due to caching is made explicit in the following lemma.

Lemma 2: The expected aggregate latency of item i under a caching strategy $\{C(n)\}_{n=1}^{\infty}$ is as follows:

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

Proof: The expected aggregate latency under $\{C(n)\}_{n=0}^{\infty}$ is the area under the solid curve as depicted in Fig.5. This can be expressed as the latency without caching, which is the area under the dotted curve, reduced by the area between the solid and dotted curves. We just need to show that the second term on the right side of equation 2 is equal to the area between the dotted and solid curves in Fig.5.

The superposition of the dotted and solid curves in Fig.5 consists of the superposition of triangles, as depicted in Fig.6, and the superposition of quadrangles as in Fig.7. For the triangles in Fig.6, the area between the dotted and solid curves is

$$\begin{aligned}
&\frac{1}{2}\lambda_i(t_3-t_1)(t_3-t_1) - \frac{1}{2}\lambda_i(t_3-t_2)(t_3-t_2) \\
&= \frac{1}{2}\lambda_i\left\{\left(\tau_i^f(t_1)+1\right)^2 - \left(\tau_i^f(t_2)+1\right)^2\right\} \\
&= \frac{1}{2}\lambda_i\sum_{n=t_1}^{t_2-1}\left\{\left(\tau_i^f(n)+1\right)^2 - \left(\tau_i^f(n+1)+1\right)^2\right\} \\
&= \frac{1}{2}\lambda_i\sum_{n=t_1}^{t_2-1}\left\{\left(\tau_i^f(n)+1\right)^2 - \left(\tau_i^f(n)\right)^2\right\} \\
&= \sum_{\substack{n\,:t_1\le n\le t_3\\ i\in C(n)}}\left(\lambda_i\tau_i^f(n)+\frac{\lambda_i}{2}\right)
\end{aligned}$$

For the quadrangles in Fig.7(a), the area between the dotted and solid line is equal to that between the bigger dotted and solid triangles in Fig.7(b) and therefore it can also be expressed as the second term on the right side of equation 2. ◇

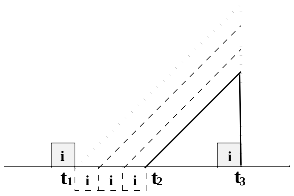

Fig. 6. A Superposition of triangles

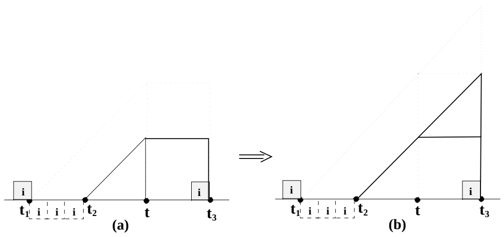

Fig. 7. A Superposition of quadrangles which can be represented by superposition of triangles

The aggregate latency over all information 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

$$\sum_{n=0}^{t-1} \sum_{i \in C(n)} (\lambda_i \tau_i^f(n) + \frac{\lambda_i}{2}) \tag{3}$$

The maximization of the sum in equation (3) is equivalent to the computation of a maximum reward path in an appropriately defined trellis diagram that captures the evolution of the memory states.

The memory state $C(n)$ at time n depends on the memory state $C(n-1)$ at time $n-1$, the information item u_{n-1} broadcast during slot $n-1$, and the action taken by the update strategy. If the item u_{n-1} broadcast at slot $n-1$ is already residing in the memory, then the item will be just refreshed in the memory and the memory state will remain the same. If not, the memory state will be one of the subsets with cardinality K of the set $\{C(n-1) \cup \{u_{n-1}\}\}$, depending on which item in the memory will be replaced by the arriving one. Hence, the set, $\mathcal{C}_n(C)$, of possible memory states at slot n, given that the memory state at slot $n-1$ is C, is

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

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

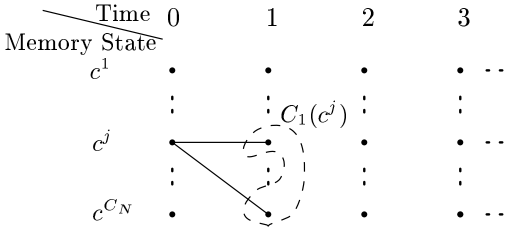

Fig. 8. Memory State Trellis Diagram

Consider a trellis diagram one dimension of which is the memory state and the other is the time as shown in Fig.8. 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 $c^j \in \mathcal{C}_{n+1}(c^i)$.

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, a time-dependent "*reward* ",

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

The total reward, or latency reduction incurred by the 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.

Shortest path problems in a trellis often arise in several contexts and can be solved by backward or forward dynamic programming. One of the notorious instances of this problem arises in sequential decoding for which *Viterbi's algorithm* was invented.

Let's denote by $\rho_n(c_j)$, the length of a maximum reward path (the cumulative reward gained) from state j at stage n $(n \leq t)$ up to stage t. The lengths of the maximum reward path satisfy the following recursion,

$$\rho_n(c_j) = \max_{c \in \boldsymbol{\mathcal{C}}_{n+1}(c_j)} \rho_{n+1}(c) + r(c_j, n) \quad n < t$$

while $\rho_t(c_j) = r(c_j, t)$. By solving the above recursion, the maximum reward path is computed and simultaneously the optimal policy as well. The cardinality of each set $\boldsymbol{\mathcal{C}}_n(c)$ is either equal to 1 or equal to $K+1$. Therefore, the complexity of computing the longest paths in stage n given the longest paths in stage $n+1$ is $O(C_N K)$. Hence, the computation of the longest path up to stage t and, subsequently, of the optimal memory update policy is of complexity $O(t\ C_N K)$.

Note that the complexity of the optimal memory update strategy within a time interval $(0, t)$ is linear with respect to t. To compute the reward associated with a state at stage n, we need a vector, $\tau^f(n) = \left(\tau_1^f(n), \tau_2^f(n), ..., \tau_K(n)\right)$, which is determined by the given broadcast schedule. Since the broadcast schedule is available ahead of time, the update strategy computation is indeed feasible. Nevertheless, 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 considered next.

IV. Memory Management With Limited Look-ahead

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 PT heuristics proposed in [12]. Let's call it *one-step look-ahead* ($OSLA$) policy. This policy updates the memory in each slot n such that the reward $r(C(n+1), n+1)$ is maximized for $C(n+1) \in \boldsymbol{\mathcal{C}}_{n+1}(C(n))$. That is, if $u_n \notin C(n)$, then the quantity $\lambda_i \tau_i^f(n) + \frac{\lambda_i}{2}$ is computed for all items, $i \in C(n)$ and for $i = u_n$ and the item j, for which $\lambda_j \tau_j^f(n) + \frac{\lambda_j}{2}$ is the smallest, is discarded. This policy turns out to be optimal in some special cases of interest.

A. Uniform Information Request Generation Rate

The first case is when all the information items have the same access probabilities. Therefore, the request generation rates are $\lambda_i = \tilde{\lambda},\ i = 1, 2, ...M$. The cumulative average reward from stage 1 to stage t then becomes

$$R(t) = \tilde{\lambda} \sum_{n=1}^{t-1} \sum_{i \in C(n)} (\tau_i^f(n) + \frac{1}{2}) \tag{4}$$

Hence, the policy that maximizes the reward is independent of the request generation rates and depends solely on the broadcast schedule. The *OSLA* policy maximizes the overall cumulative average reward in (4) over all policies in this case. This fact is true in a stronger sense as it is expressed in the following theorem.

Theorem 1: When the request generation rates are identical, the *OSLA* policy maximizes the reward in each slot n, that is, for any policy π and at any slot n, $n = 0, 1, 2, ..$

$$r(C^O(n), n) \geq r(C^\pi(n), n) \tag{5}$$

where $C^O(n)$ and $C^\pi(n)$ denote the set of items residing in the memory at slot n under *OSLA* policy and policy π respectively.

Proof: See Appendix. $\diamond$

B. Two Information Items

Another case where the *OSLA* policy maximizes equation (3) is when the broadcast includes only two information items, the user has memory space for one item and the request generation rates are arbitrary. A cache update decision needs to be made only if the broadcast item is different from the item in the cache. In this case, the item that is chosen to be kept in the cache at slot n is

$$\arg \max_{i \in \{A,B\}} \lambda_i(\tau_i^f(n) + \frac{1}{2}).$$

Theorem 2: When the server broadcasts only two items and user's memory can hold one item only, the *OSLA* policy is optimal in the sense that it maximizes the reward in each slot n, that is, for any policy π

$$r(C^O(n), n) \geq r(C^\pi(n), n), \quad n = 0, 1, 2, ...$$

Proof: See Appendix. $\diamond$

C. A Case Where Limited Look-ahead Policy is Optimal

A special case where the optimal update policy is of the limited look-ahead type is the following. The server broadcasts multiple information items while a certain group of users is only interested in two of those, let's say items A and B, and each user of that group possesses memory space for only one item. In this case, the users can make memory updates only at the slots when either item A or item B is transmitted. If, at a slot of this type, the broadcast item is the same as the item in the cache, then no update decision needs to be made. If the items are different, assume, without loss of generality, that item A is in the cache and item B is broadcast. We distinguish two cases as depicted in Fig.9.

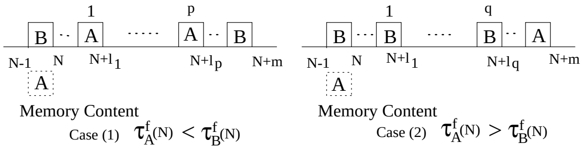

Fig. 9. Two possible cases with $p \geq 1$ and $q \geq 1$ when we look $\max(\tau_A^f(N), \tau_B^f(N))$ slots ahead at time instant N when the arriving item is different from the one already stored in the memory.

In the first case, there are p ($p \geq 1$) occurrences of item A on the broadcast before the first transmission of item B after N and the last of those p item A's is transmitted in slot $N + l_p - 1$. In the second case, there are q ($q \geq 1$) transmissions of item B before the first appearance of item A after N in the broadcast and the last of those q item B's is transmitted in slot $N + l_q - 1$. The items other than items A and B are transmitted during the remaining unmarked slots between $N - 1$ and $N + m$.

In order to make the update decision, the optimal policy needs to consider l_p (or l_q) slots look-ahead. The optimal decision, then, is to keep the item k in the memory, where

$$k = \arg \max_{i \in \{A,B\}} \sum_{j=0}^{W} \lambda_i (\tau_i^f(N + j) + \frac{1}{2}). \tag{6}$$

and where $W = l_p - 1$ or $l_q - 1$ as appropriate.

To show the optimality of the policy, we consider a partition of the joint evolution of the broadcast sequence and the cache content under the optimal policy as depicted in Fig.10. Cycle i begins at time instant n_i and ends at n_{i+1} with $i = 0, 1, ...$ and $n_0 = 0$.

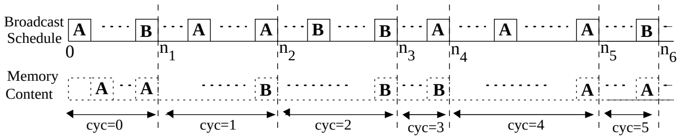

Fig. 10. Decomposition of the given broadcast schedule under the optimal memory update policy into cycles

Cycle 0 ($cyc = 0$) is defined as the interval from slot 0 until the end of first slot at which the information item transmitted by the server in this slot is different from the one stored in the memory. In addition to cycle 0, there are three types of cycles: *type 1* cycle which corresponds to the interval from N to $N + l_p$ as shown in Fig.9 case (1), *type 2* cycle which corresponds to the interval from N to $N + l_q$ as shown in Fig.9 case (2) and *type 3* cycle which spans from the end of type 1 or type 2 cycle up to the beginning of type 1 or type 2 cycle. Type 3 cycle occurs only when the information item stored in the memory under the W-step look-ahead policy is the same as the one broadcast by the server at the last slot of type 1 or type 2 cycle. In Fig.10, cycle 1 and cycle 2 are of type 1, cycle 3 and cycle 5 are of type 3 and cycle 4 is of type 2.

Let R_j be the average reward gained due to caching during the cycle j,

$$R_j = \sum_{l=n_j}^{n_{j+1}-1} \lambda_i(\tau_i^f(l) + \frac{1}{2})\mathbf{1}\{i \in C(l)\}$$

Theorem 3: The policy with W-step look-ahead where $W = l_p - 1$ or $W = l_q - 1$ as defined above maximizes the reward in each cycle j. That is, for any policy π and at any cycle j, $j = 0, 1, 2, ...$

$$R_j^W \geq R_j^\pi$$

where R_j^W and R_j^π be the average reward gained during the cycle j under the policy with W-step look-ahead and under policy π respectively.

Proof: See Appendix. $\diamond$

D. Numerical Examples

In this section, in order to gain some insight in the trade-off between the performance improvement and the computational complexity, we present some numerical examples with different number of look-ahead steps, different cache sizes and different broadcast schedules for both latency and deadline cases (Please refer to section V for the case with deadlines). Although we can provide examples only for a small number of items due to the combinatorial explosion even for a few tens, the examples reflect the fact that the performance gain due to the increase in the number of look-ahead steps is dependent on the type of server's broadcast schedule and the set of user's access statistics parameters we consider.

For the first case in which the expected latency is the performance criterion, an example is presented with $M = 10$, $\lambda = 1$ and zipf distribution version I [16] where $b_i = \frac{c}{i}$, $i = 1, ..., M$ where c is a normalizing constant given by $c = (\sum_{j=1}^{M} 1/j)^{-1}$. For the case with deadlines, an example is given with $M = 15$, $\lambda = 100$ and zipf distribution [17] where $b_i = \frac{i^\theta - (i-1)^\theta}{M^\theta}$ where $i = 1, ..., M$. The deadlines are assumed to be uniformly distributed between 1 and 10. As θ increases, 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. Two types of broadcast schedules are used in the experiment. One is generated by the Ammar's algorithm [6] which produces near-optimal schedules, namely *Ammar's*, and the other is a schedule which is *not* constructed according to the statistical parameters of user's access pattern, namely *Mismatched*.

The results in Tables I (II) show that the expected latency (number of deadline misses) reduced by caching is insensitive to the number of look-ahead steps for the parameter set we are considering. It may be due to the small number of information items in our experiment. However, the number of look-ahead steps may play a greater role when the number of items is large and the group of users we are considering is interested only in a portion of the set of information items as in the following example. Our cache management strategy is able to reduce more latency (or deadline misses) when the server's schedule is *not* designed according to user's access parameters (compare the results for Ammar's and Mismatched

TABLE I

Expected latency reduction due to caching for different cache sizes and different look-ahead steps with zipf distribution where L denotes the No. of look-ahead steps

	Cache Size						
L	1	2	3	4	5	6	7
1	1.05	1.80	2.41	2.92	3.35	3.70	3.96
2	1.05	1.80	2.41	2.92	3.35	3.70	3.96
3	1.06	1.80	2.41	2.92	3.35	3.70	3.96
4	1.06	1.80	2.41	2.92	3.35	3.70	3.96
5	1.06	1.80	2.41	2.92	3.35	3.70	3.96
6	1.06	1.80	2.41	2.92	3.35	3.70	3.96
7	1.06	1.80	2.41	2.92	3.35	3.70	3.96
8	1.06	1.80	2.41	2.92	3.35	3.70	3.96
9	1.06	1.80	2.41	2.92	3.35	3.70	3.96

Under Ammar's Schedule with period 83

	Cache Size						
L	1	2	3	4	5	6	7
1	2.63	4.41	5.67	6.56	7.14	7.52	7.81
2	2.63	4.43	5.70	6.59	7.15	7.53	7.81
3	2.63	4.43	5.70	6.59	7.15	7.54	7.82
4	2.63	4.43	5.70	6.59	7.15	7.54	7.82
5	2.63	4.43	5.70	6.59	7.15	7.54	7.82
6	2.63	4.43	5.70	6.59	7.15	7.54	7.82
7	2.63	4.43	5.70	6.59	7.15	7.54	7.82
8	2.63	4.43	5.70	6.59	7.15	7.54	7.82
9	2.63	4.43	5.70	6.59	7.15	7.54	7.82

Under the Mismatched Schedule with period 83

schedules). The latency (or the number of deadline misses) we can reduce by the use of local memory is indeed increasing with the size of the memory for all cases.

In the following, we will present an example which shows that $OSLA$ policy, in some cases, may perform poorly compared to the optimal policy.

Consider a case in which the server's schedule is given as in Fig.11 and the access probabilities of a user group are $b_A = \frac{2}{3}$, $b_B = \frac{1}{3}$ and $b_i = 0$ for $i = C, ..., U$.

B	C	D	E	F	G	H	I	J	K	L	M	N	O	A	P	A	Q	R	A	S	T	A	U

0 1 14 15 16 17 19 20 22 23 24

Fig. 11. A Server schedule with period 25 and information items $\{A, B, ..., T, U\}$

Without loss of generality, assume that item A is initially stored in the memory in slot 0. Since $\sum_{j=1}^{W} \lambda_A(\tau_A^f(j) + \frac{1}{2}) > \sum_{j=1}^{W} \lambda_B(\tau_B^f(j) + \frac{1}{2})$ for $W = 1, ..., 7$, at the end of slot 0, $OSLA$ policy (in fact, all the policies with look-ahead steps less than eight) will not replace item A, which is stored in the memory, with the arriving item B. However, by keeping item A, $OSLA$ policy have no more opportunity to store item B in the memory until the end of slot 24.

If, at time 0, though, we look ahead more than eight slots (note that $\sum_{j=1}^{8} \lambda_A \left(\tau_A^f(j) + \frac{1}{2}\right) = \sum_{j=1}^{8} \lambda_B \left(\tau_B^f(j) + \frac{1}{2}\right)\right)$, we will see that we can further reduce the latency by replacing item A with item B at the beginning of slot 1 and keeping it up to the end of slot 22. This is what the optimal policy

TABLE II

Expected number of deadline misses reduced by caching for different cache sizes and different look-ahead steps with zipf distribution and uniformly distributed deadlines where L denotes the No. of look-ahead steps

	Cache Size						
L	1	2	3	4	5	6	7
1	10.02	14.05	17.05	19.36	21.02	22.38	23.64
2	10.02	14.11	17.20	19.45	21.21	22.48	23.76
3	10.05	14.15	17.20	19.45	21.24	22.64	23.86
4	10.05	14.15	17.20	19.45	21.24	22.64	23.86
5	10.08	14.15	17.20	19.45	21.24	22.65	23.90
6	10.08	14.15	17.20	19.45	21.24	22.67	23.90
7	10.08	14.15	17.20	19.45	21.24	22.67	23.94
8	10.08	14.15	17.20	19.45	21.24	22.67	23.94
9	10.08	14.16	17.20	19.45	21.24	22.67	23.94

Under Ammar's Schedule with period 55

	Cache Size						
L	1	2	3	4	5	6	7
1	51.71	57.17	61.19	63.57	65.31	67.33	68.89
2	51.71	57.19	61.19	63.64	65.36	67.40	68.97
3	51.71	57.19	61.19	63.67	65.54	67.50	69.10
4	51.71	57.19	61.20	63.67	65.63	67.51	69.10
5	51.71	57.19	61.20	63.67	65.68	67.55	69.10
6	51.71	57.19	61.20	63.73	65.77	67.62	69.12
7	51.71	57.19	61.20	63.73	65.77	67.63	69.12
8	51.71	57.19	61.20	63.73	65.78	67.64	69.13
9	51.71	57.19	61.20	63.73	65.78	67.64	69.13

Under the Mismatched Schedule with period 200

will choose to do. This example belongs to the third special case we consider in section IV-C for which W-step look-ahead policy is optimal. Here, $W = 21$ at slot 1, $W = 7$ at slot 15, $W = 5$ at slot 17, $W = 2$ at slot 20 and $W = 1$ at slot 23 in equation (6) for the optimal policy.

By using the equation (3) in section III for the interval from slot 1 to slot 22, the optimal policy reduces average latency by 95.33 slots whereas *OSLA* policy (all the the policy with look-ahead steps less than eight) reduces the latency by 72.67 slots. The policy with more than 8 slot look-ahead can reduce the average latency 22.66 slots more. This difference of latency reduction is approximately equal to the shaded area in Fig.12. In fact, the shaded area may be arbitrarily large for a certain broadcast schedule so that *OSLA* policy will perform even worse compared to the optimal policy. This shows that *OSLA* policy may be inadequate in terms of performance in certain cases and policies which look ahead more than one slot may need to be considered.

V. Information Requests with Deadlines

In this section, we will consider the case in which users generate information requests which have to be satisfied within some specified deadlines and show that the problem can be formulated in the same way as the case without deadlines in section III. The type j item i requests which are generated during the interval $[n, n + \tau_i^f(n) + 1 - d_i^j]$ will miss the deadlines d_i^j if the user does not make use of its local memory as depicted in Fig.13.

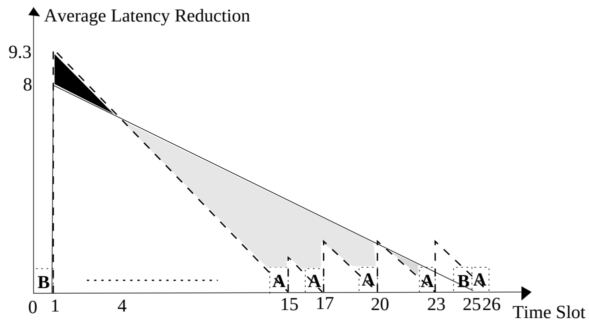

Fig. 12. The figure shows the average latency reduction as a function of time slots. The area under the solid triangle represents the average latency reduction gained by caching when the optimal policy is follo wed and the area under the dashed triangles corresponds to the latency reduction for OSLA policy.

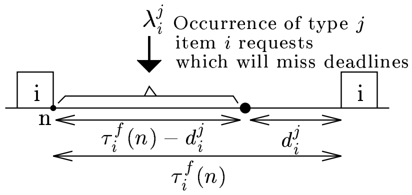

Fig. 13. Deadline misses of type j item i requests without the use of local memory

However, if the user has a local memory and item i is stored in the memory, the type j item i requests which occur during the slots in which item i is residing in the memory will be satisfied immediately and will no longer miss deadlines as shown in Fig.14.

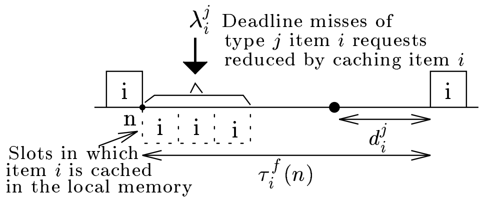

Fig. 14. Deadline misses of type j item i requests reduced by caching item i in the local memory

Let $R_i^j(t)$ be the expected number of deadline *misses* of item i requests with deadline d_i^j up to time t, which can be *reduced* by caching, and it is given by

$$R_i^j(t) = \sum_{n:n<t, i\in C(n)} \lambda_i^j \; \mathbf{1}\left\{(\tau_i^f(n) + 1 - d_i^j) > 0\right\}$$

where $\mathbf{1}\{.\}$ is the indicator function. The overall reduction of deadline misses for all information items with all deadlines up to slot t due to caching is

$$R(t) = \sum_{i=1}^{M}\sum_{j=1}^{a_i} R_i^j(t)$$

Therefore, minimizing the total number of requests that miss deadlines with caching is equivalent to maximizing the reduction of deadline misses by caching. Moreover, the optimal memory update strategy that minimizes the deadline misses of user's requests is equivalent to the computation of the maximum reward path in the memory state trellis diagram where the time-dependent reward associated with memory state c^j at instant n is now given by

$$r(c^j, n) = \sum_{i \in c^j} \sum_{l=1}^{a_i} \lambda_i^l \ \mathbf{1}\left\{(\tau_i^f(n) + 1 - d_i^l) > 0\right\}$$

VI. Concluding Remarks

A strategy for a mobile user's local memory management is identified such that the data access needs of the user are matched optimally to broadcast schedule of the server. The strategy relies on the computation of the maximum reward path in the appropriately defined memory state trellis diagram. Limited look-ahead policies are also presented as low implementation complexity alternatives to the optimal policy. The simplest of them is *OSLA* strategy. This policy is found to be optimal when all the information request generation rates are the same. When the request generation rate distribution though is skewed, the performance of *OSLA* policy may get considerably degraded compared to the optimal policy as it was demonstrated by a counter example. The choice of look-ahead steps is a trade-off between implementation complexity and good performance.

User's memory management is just one of two important issues to be addressed in data broadcasting. The other complementary problem is the organization of data in a broadcast schedule in order to minimize user's access latency or the number of deadline misses depending on the performance objective we consider. Both problems have been addressed separately in literature. What remains to be done is to obtain an integrated approach that produces broadcast schedules and user's memory management strategies simultaneously, based on the mix of data access patterns of the user population.

VII. Appendix

Proof for the theorem 1: Let $Y^O(n) = (Y_1^O(n), ..., Y_K^O(n))$ be the vector with elements, the times $\tau_k^f(n)$'s for all K items in the memory at slot n under *OSLA* policy, arranged in decreasing order, i.e., $Y_k^O(n) \geq Y_{k+1}^O(n)$ for $k = 1, ..., K-1$. Let $Y^\pi(n)$ be the corresponding vector for policy π. We will show that

$$Y^O(n) \geq Y^\pi(n) \qquad (7)$$

for $n = 0, 1, ..$ where the inequality holds elementwise. Inequality (7) clearly implies inequality (5). Therefore, the theorem will be proved if the inequality (7) is shown. The proof is by induction.

For $n = 0$, (7) holds with equality. We will show that if (7) holds for $n = N$, then it will hold for $n = N + 1$ as well.

Note that according to the definition of $\tau_k^f(n)$, if $u_{n-1} \neq k$, then $\tau_k^f(n-1) \neq 0$ and $\tau_k^f(n) = \tau_k^f(n-1) - 1$

while if $u_{n-1} = k$, then $\tau_k^f(n-1) = 0$ and $\tau_k^f(n)$ assumes a new positive value which is equal to the number of slots from the beginning of slot n until the beginning of first item k broadcast.

Assume that item A is transmitted in slot N. Then, $\tau_A^f(N) = 0$. There are two cases to consider: $A \in C^O(N)$ and $A \notin C^O(N)$. For the case with $A \in C^O(N)$, since we have assumed $Y^O(N) \geq Y^\pi(N)$, $A \in C^\pi(N)$ as well. Therefore, it still holds that

$$Y^O(N+1) \geq Y^\pi(N+1).$$

For $A \notin C^O(N)$, the following cases are distinguished.

For $(y_K^O(N) - 1) > \tau_A^f(N+1)$, *OSLA* policy does not replace any of the items in the memory with the arriving item A and the set of items in the memory remains the same, i.e., $C^O(N+1) = C^O(N)$. Therefore,

$$y_k^O(N+1) = y_k^O(N) - 1 \quad for \quad k = 1, ..., K \tag{8}$$

For $(y_K^O(N) - 1) < \tau_A^f(N+1)$, *OSLA* policy replaces the information item corresponding to index K in the vector $Y^O(N)$ with the arriving item A and then item A will correspond to some index l in the vector $Y^O(N+1)$ such that $y_{l-1}^O(N) - 1 > \tau_A^f(N+1)$ and $y_l^O(N) - 1 < \tau_A^f(N+1)$ for $2 \leq l \leq K$ and $y_1^O(N) - 1 < \tau_A^f(N+1)$ for $l = 1$. The elements of the vector $Y^O(N+1)$ under *OSLA* policy are now given by

$$\begin{aligned} y_k^O(N+1) = &\quad y_k^O(N) - 1 \quad\quad for \quad k = 1, .., l-1 \ \text{ and } \ l \neq 1 \\ y_l^O(N+1) = &\quad \tau_A^f(N+1) \\ y_k^O(N+1) = &\quad y_{k-1}^O(N) - 1 \quad for \quad k = l+1, .., K \end{aligned} \tag{9}$$

The set of information items stored in the memory at slot $N+1$ under policy π will be

$$C^\pi(N+1) \subseteq \{\{A\} \cup C^\pi(N)\} \quad \text{with} \quad |C^\pi(N+1)| = K$$

Due to the assumption of $Y^O(N) \geq Y^\pi(N)$ and equations (8) and (9), irrespective of whether item A is stored in the memory under policy π at slot $N+1$, it is apparent that

$$Y^O(N+1) \geq Y^\pi(N+1)$$

Therefore, by induction,

$$Y^O(n) \geq Y^\pi(n) \quad n = 0, 1, ...$$

and the theorem is proved. $\diamond$

Proof for the theorem 2: We will prove the theorem by induction.

Clearly, $r(C^O(0), 0) = r(C^\pi(0), 0)$.

Assume that $r(C^O(N), N) \geq r(C^\pi(N), N)$. We will show that the inequality still holds for $n = N+1$ too.

There are four cases to consider depending on the sequence of broadcasts and the item stored in the memory at slot N under the $OSLA$ policy as shown in Fig.15. Assume that item A and item B are the two items broadcast by the server.

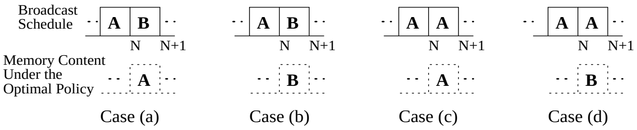

Fig. 15. Four possible cases

First, consider cases (a) and (d) in Fig.15 in which the information item broadcast by the server at slot N is different from the one stored in the memory at slot N under the $OSLA$ policy. For case (a), if $\lambda_B(\tau_B^f(N+1)+\frac{1}{2}) > \lambda_A(\tau_A^f(N+1)+\frac{1}{2})$, the $OSLA$ policy replaces item A currently stored in the memory with the arriving item B. Otherwise, item A will be kept intact in the memory at slot $N+1$ under the $OSLA$ policy. In any case, $r(C^O(N+1), N+1) \geq r(C^\pi(N+1), N+1)$. The same thing goes for case (d).

For cases (b) and (c) in which the item stored in the memory at slot N under the $OSLA$ policy is the same as the one transmitted at slot N, the $OSLA$ policy cannot replace the item in the memory with the arriving one except refreshing the item currently stored in the memory. Therefore, for the inequality to hold for slot $N+1$, we need to show that we can obtain bigger reward by keeping the same item in the memory as the arriving item in slot $N+1$.

Consider case (b). According to $OSLA$ policy which chooses to keep item B in the cache at slot N,

$$\lambda_B(\tau_B^f(N)+\frac{1}{2}) > \lambda_A(\tau_A^f(N)+\frac{1}{2})$$

But since $\tau_A^f(N) > \tau_B^f(N)$ (in fact, $\tau_B^f(N)=0$), $\lambda_B > \lambda_A$. Therefore,

$$\lambda_B(\tau_B^f(N+1)+\frac{1}{2}) > \lambda_A(\tau_A^f(N+1)+\frac{1}{2})$$

because $\tau_A^f(N+1) = \tau_A^f(N) - 1$ and $\tau_B^f(N+1) \geq 0$. Consequently,

$$r(C^O(N+1), N+1) \geq r(C^\pi(N+1), N+1)$$

For case (c), there are two subcases to consider as shown in Fig.16. Assume that, at slot $N-q-1$ where $q \geq 1$, item B is broadcast by the server the last time before slot $N-1$. In case (c.1), the optimal policy replaces item B by the arriving item A in slot $N-p$ where $p = 0, ..., q-1$ whereas item A is already residing in the cache in slot $N-q-1$ under the optimal policy in case (c.2).

In case (c.1), according to $OSLA$ policy,

$$\lambda_A\left(\tau_A^f(N-p)+\frac{1}{2}\right) > \lambda_B\left(\tau_B^f(N-p)+\frac{1}{2}\right)$$

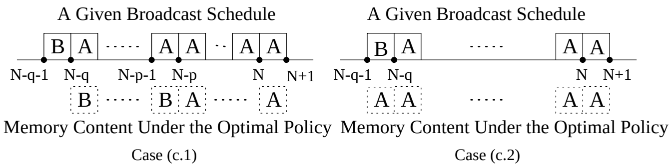

Fig. 16. Two possible subcases for case (c) in which item B is transmitted, the last time before slot $N-1$, in slot $N-q-1$

In addition, since $\tau_B^f(N-p) > 0$ and $\tau_A^f(N-p) = 0$, $\lambda_A > \lambda_B$. Therefore, since $\tau_B^f(N+1) = \tau_B^f(N-p)-p-1$ and $\tau_A^f(N+1) \geq 0$,

$$\lambda_A(\tau_A^f(N+1)+\frac{1}{2}) > \lambda_B(\tau_B^f(N+1)+\frac{1}{2})$$

Hence,

$$r(C^O(N+1), N+1) \geq r(C^\pi(N+1), N+1)$$

In case (c.2), according to *OSLA* policy,

$$\lambda_A(\tau_A^f(N-q)+\frac{1}{2}) > \lambda_B(\tau_B^f(N-q)+\frac{1}{2})$$

and the same argument as in case (c.1) can be applied.

Therefore, by induction, the theorem is proved. $\diamond$

Proof for the theorem 3: We are going to prove it by induction.

For $j=0$, $R_0^W = R_0^\pi$ since no memory update is required for any policy.

Assume that $R_J^W \geq R_J^\pi$. We will show that the inequality still holds for $j = J+1$. Both cycle J and cycle $J+1$ can be of type 1 or type 2 or type 3.

Fig.'s 17 and 18 show the possible sequence of cycle types when the first cycle is of either type 1 or type 2.

Case I: Cycle J is of either type 1or type 2 and cycle $J+1$ is of type 1 as in Fig.'s 17(i) and 18(iii). Consider the case in Fig.17(i) where item A is arriving in the broadcast and item B is stored in the memory at the end of cycle J. If

$$\sum_{i=n_{J+1}}^{n_{J+2}-1} \lambda_B(\tau_B^f(i)+\frac{1}{2}) > \sum_{i=n_{J+1}}^{n_{J+2}-1} \lambda_A(\tau_A^f(i)+\frac{1}{2}),$$

then item B will remain in the memory in cycle $J+1$ under the W-step look-ahead policy. Otherwise, the W-step look-ahead policy will replace item B currently stored in the memory with the arriving item A and keep it until the end of slot v at which item B is arriving in the broadcast and

$$\sum_{i=v+1}^{n_{J+2}-1} \lambda_B(\tau_B^f(i)+\frac{1}{2}) > \sum_{i=v+1}^{n_{J+2}-1} \lambda_A(\tau_A^f(i)+\frac{1}{2}).$$

If no such slot v exists, item A is kept in the memory until the end of $J+1$ cycle under the W-step look-ahead policy. In any case, $R_{J+1}^W \geq R_{J+1}^\pi$.

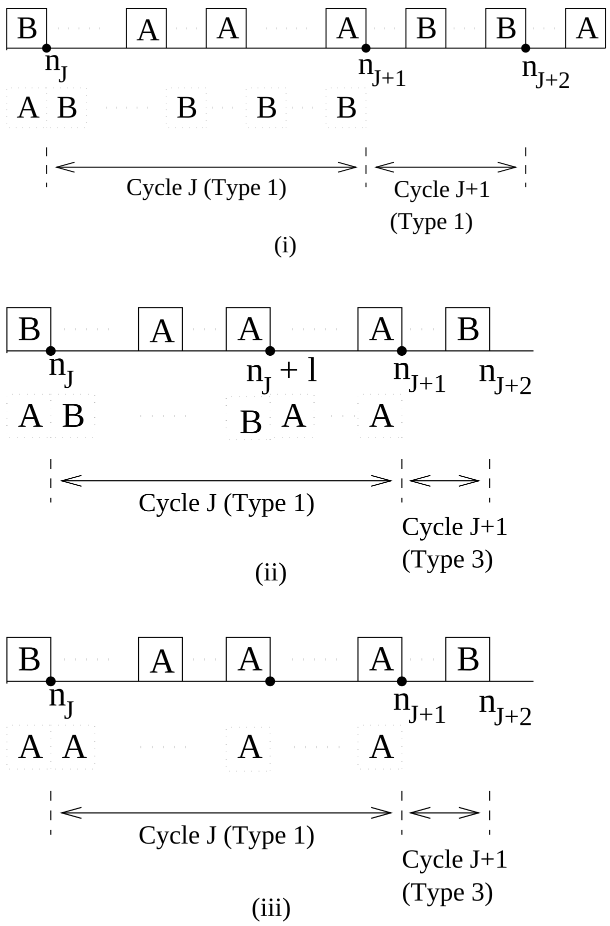

Fig. 17. Three possible cases for cycle $J+1$ when cycle J is of type 1

Case II Cycle J is of type 3 and cycle $J+1$ is of either type 1 or type 2. In this case, the inequality still holds for cycle $J+1$ for the same reason as in case I.

Case III Cycle $J+1$ is of type 3 as in Fig.'s 17(ii), 17(iii), 18(i) and 18(ii). For this case, since the information item broadcast by the server is the same as the one stored in the memory under the W-step look-ahead policy, we cannot make the replacement of information item but just refresh the item in the memory. Hence, we need to show that it is better to keep the same item in the memory as the arriving one in the broadcast for cycle $J+1$.

Consider the case in Fig.17(ii) where item B stored in the memory is replaced by the arriving item A at slot n_J+l where $1 \le l < n_{J+1} - n_J$. According to W-step look-ahead policy,

$$\sum_{i=n_J+l}^{n_{J+1}-1} \lambda_B(\tau_B^f(i) + \frac{1}{2}) < \sum_{i=n_J+l}^{n_{J+1}-1} \lambda_A(\tau_A^f(i) + \frac{1}{2})$$

Therefore, $\lambda_A > \lambda_B$ because $\tau_A^f(i) < \tau_B^f(i)$ for $i = n_J + l, ..., n_{J+1} - 1$.

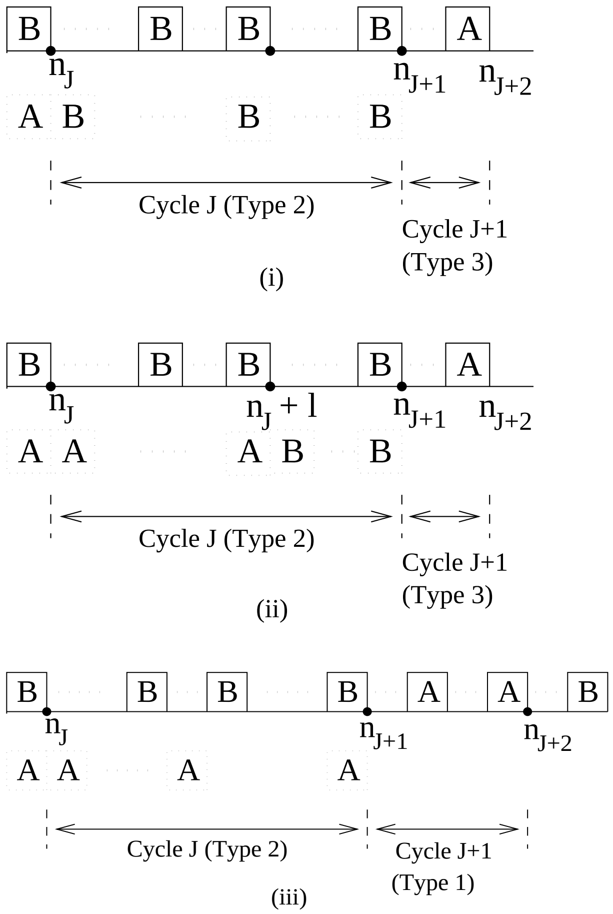

Fig. 18. Three possible cases for cycle $J+1$ when cycle J is of type 2

Then, since $\tau_A^f(i) > \tau_B^f(i)$ for $i = n_{J+1}, ..., n_{J+2}-1$,

$$\sum_{i=n_{J+1}}^{n_{J+2}-1} \lambda_B(\tau_B^f(i) + \frac{1}{2}) < \sum_{i=n_{J+1}}^{n_{J+2}-1} \lambda_A(\tau_A^f(i) + \frac{1}{2})$$

and therefore,

$$R_{J+1}^W \geq R_{J+1}^\pi.$$

Similarly, it can be proved for cases in Fig.'s 17(iii), 18(i) and 18(ii).

Hence, by induction,

$$R_j^W \geq R_j^\pi \quad j = 0, 1, ...$$

◇

Acknowledgments

The authors would like to thank the reviewers for their thoughtful comments that helped to improve the presentation of this paper.

References

[1] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz, and A. Weinrib, "The Datacycle Architecture," *Communications of the ACM*, vol. 35, no. 12, pp. 71–81, December 1992.

[2] T. Imielinski and B. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," *Communications of the ACM*, vol. 37, no. 10, pp. 18–28, October 1994.

[3] T. Imielinski, S. Viswanathan, and B. Badrinath, "Energy Efficient Indexing on Air," *ACM SIGMOD*, pp. 25–36, 1994.

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

[5] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination-based Data Delivery Using Broadcast Disks," *IEEE Personal Communications*, vol. 2, no. 6, pp. 50–60, December 1995.

[6] M. H. Ammar and J. W. Wong, "The Design of Teletext Broadcast Cycles," *Performance Evaluation*, vol. 5, no. 4, pp. 235–242, December 1985.

[7] M. H. Ammar and J. W. Wong, "On the Optimality of Cyclic Transmission in Teletext Systems," *IEEE Transaction on Communication*, vol. COM-35, no. 1, pp. 68–73, January 1987.

[8] J. W. Wong, "Broadcast Delivery," *Proceedings of the IEEE*, vol. 76, no. 12, pp. 1566–1577, December 1988.

[9] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments," Tech. Rep. CS-94-43, Dept. of Comp. Science, Brown University, October 1994.

[10] C. J. Su and L. Tassiulas, "Broadcast Scheduling for Information Distribution," *Proc. IEEE INFOCOM'97*, Kobe, Japan, April, 1997.

[11] C.-J. Su, L. Tassiulas, and V. Tsotras, "A New Method to Design Broadcast Schedules in a Wireless Communication Environment," Tech. Rep., Institute For Systems Research, University of Maryland, College Park, 1996.

[12] 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.

[13] M. H. Ammar, "Response Time in a Teletext System: an Individual User's Perspective," *IEEE Transaction on Communication*, vol. COM-35, no. 11, pp. 1159–1170, November 1987.

[14] T. Imielinski, S. Viswanathan, and B. Badrinath, "Power Efficient Filtering of Data on Air," *Proc. of 4th Inter. Conf. on EDBT*, March 1994.

[15] T. Imielinski, S. Viswanathan, and B. Badrinath, "Data on Air: Organization and Access," *IEEE Transactions in Data and Knowledge Engineering*, July 1996.

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

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

Biography of Leandros Tassiulas

Leandros Tassiulas (S'89, M'91) was born in 1965, in Katerini, Greece. He obtained the Diploma in Electrical Engineering from the Aristotelian University of Thessaloniki, Thessaloniki, Greece in 1987, and the M.S. and Ph.D. degrees in Electrical Engineering from the University of Maryland, College Park in 1989 and 1991 respectively. From 1991 to 1995 he was an Assistant Professor in the Department of Electrical Engineering, Polytechnic University, Brooklyn, NY. In 1995 he joined the Department of Electrical Engineering, University of Maryland, College Park, where he is now an Associate Professor . He holds a joint appointment with the Institute for Systems Research and is a member of the Center for Satellite and Hybrid Communication Networks, established by NASA. His research interests are in the field of computer and communication networks with emphasis on wireless communications and high-speed network architectures and management, in control and optimization of stochastic systems and in parallel and distributed processing.

Dr. Tassiulas received a National Science Foundation (NSF) Research Initiation Award in 1992 an NSF Faculty Early Career Development Award in 1995 and an Office of Naval Research Young Investigator Award in 1997. He coauthored a paper that got the INFOCOM '94 best paper award.

Biography of Chi-Jiun Su

Chi-Jiun Su (S'95) received the B.S. degree in communication engineering from the National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1993, and the M.S. degree in electrical engineering from Polytechnic University, Brooklyn, NY, in January 1996, where he is currently a Ph.D. candidate at the Department of Electrical Engineering.

He was awarded Special Polytechnic Graduate Fellowship in 1993 and was a research assistant at the Center for Advanced Technology in Telecommunications (CATT) at Polytechnic University. He is currently visiting the University of Maryland, College Park as a Faculty Research Assistant at the Institute for Systems Research. His main research interests include large scale information dissemination systems, mobile computing, design and management of wireless communication and high speed networks.

Mr. Su is a member of Eta Kappa Nu.