TECHNICAL RESEARCH REPORT

Novel Information Distribution Methods to Massive Mobile User Populations

by C-J. Su, L. Tassiulas

CSHCN T.R. 97-14
(ISR T.R. 97-46)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

NOVEL INFORMATION DISTRIBUTION METHODS TO MASSIVE MOBILE USER POPULATIONS*

Chi-Jiun Su
Leandros Tassiulas

Electrical Engineering Department and
Institute For Systems Research
University of Maryland
College Park MD 20742

ABSTRACT

Broadcast data delivery is encountered in many applications where there is a need to disseminate information to a large user community in a wireless asymmetric military communication environment. In this paper, we consider two types of broadcast data delivery systems, namely, push-based and pull-based, and provide a low-complexity near-optimal scheduling algorithms for both systems. By using a numerical study, we also discuss the performance limit of a pull-based system. In addition, we identify the optimal memory management policy for the users in a push-based broadcast delivery system and propose implementable alternatives to the optimal policy.

INTRODUCTION

Dissemination of accurate, timely and consistent information to warfighters in a battlefield plays a crucial role in winning a battle in a modern warfare [5]. The information ranges from the weather at the battlefield to the arsenal, number and position of the enemy. The communication channel between mobile warfighters and the central control command is wireless and asymmetric in most cases. Communication asymmetry can arises in a number of ways. One type is *physical asymmetry* when an asymmetric bandwidth or power limitations exist between downlink and uplink communication. An example of this type is the case when the stationary central control command have powerful broadcast transmitters while mobile warfighters have little or no transmission capability. Another type of asymmetry arises in *information flow* an example of which happens when a massive number of mobile warriors are simultaneously requesting information from a few command centers. Broadcasting is a promising candidate to play a leading role in this asymmetric communication environment since it reduces the relatively expensive client-to-server communication and it is scalable in such a way that it is independent of the number of clients the server is serving.

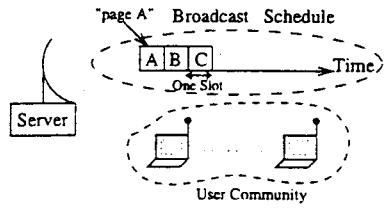

Figure 1: *A Broadcast Data Delivery System in a Wireless Communication Environment*

In a broadcast data delivery system, depicted in figure 1, a server (the central control command) is continuously and repeatedly broadcasting data to a user community (geographically scattered mobile warfighters). There are two basic architectures for a broadcast delivery system: *push-based* broadcast delivery in which 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 and *pull-based* broadcast delivery in which there is a uplink channel available through which a user can send a request to the server to tell what it is waiting for.

*Prepared through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0002

Information broadcast by the server is organized into units called *pages*. When a user needs a certain page, it waits until the desired page appears on the broadcast and captures it for use. There is some latency from the time the need of a page arises until the time the page is actually broadcast by the server. This latency depends on the broadcast schedule of the server. For a push-based system, due to the limitation imposed by the asymmetric communication channel, the server may know only the past access pattern of the users or an estimate of the user's access probability. The server relies on this information in order to broadcast the pages according to a schedule that results in low latency for the user's requests. For the other system, the server knows the exact number of pending requests for each page at each slot and can make use of the page request backlog information to decide which page to broadcast in each slot so as to minimize the latency of user's requests.

Two major issues arise in data delivery systems: a) the organization of the data in a broadcast schedule so as to minimize the *average response time* ([3], [9], [1], [7], [6] and [4]) and b) the user's memory management in order to reduce the mismatch between the broadcast schedule and user's access pattern ([8] and [2]).

If the user has local storage, i.e. memory, it can retrieve pages from the broadcast and store it in its memory prior to the pages being requested. If the user makes a request for one of the "prefetched" stored pages, the response time for this request will be instantaneous. By selectively prefetching information pages 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 minimized. The user's memory management becomes an important issue to consider in order to minimize the average response time of information requests. As pages pass by on the broadcast, the user has to decide whether a page will be prefetched and if it will, which page residing in the memory will be replaced with the newly prefetched page.

We propose a scheduling policy for a push-based system which, based on the user's access probability, generates periodic broadcast schedules with mean access latency close to the lower bound. It is an low-complexity on-line algorithm that can adapt to the changes in user's access pattern. Moreover, it has a unique advantage that it can be readily generalized for a system with multiple broadcast channels. A suboptimal scheduling algorithm with good performance is also provided for a pull-based system. By a numerical study, we point out the crucial fact that as the request generation rate increases, the achievable performance of the pull- and push-based systems becomes almost identical. For the second problem, 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.

BROADCAST SCHEDULING

Time on the broadcast channel is divided into slots of same size that is equal to the time to broadcast a page. Slot n is the interval $[n, n+1)$. At each slot n, one page 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 pages.

Consider the aggregate stream of page requests generated by the whole user population. For a sufficiently large user population, we may assume that the process of request generation is *Poisson* with rate λ page per time slot.

A request is for page i with probability b_i, $i = 1, ..., M$, where $\sum_{i=1}^{M} b_i = 1$. Hence, requests for page i are generated according to a Poisson process with rate $\lambda_i = b_i \lambda$. Let $A_i(n)$ be the total number of requests for page i generated during slot n which is a Poisson random variable with rate λ_i. Let $X_i(n)$ be the total number of pending requests for page i at the beginning of slot n.

The request backlog for page i evolves as follows:

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

A PUSH-BASED SYSTEM

When the server is not aware about the user's requests, the broadcast schedule is designed based only on the distribution of page requests, that is, b_i, $i = 1, ..., M$.

There are two quantities related to each page i that affect the scheduling decision at each slot n, namely, page i request generation rate λ_i and the parameter $w_i(n)$ which is the amount of time from the end of the last slot before t at which page i was transmitted until the end of slot t, as shown in figure 2.

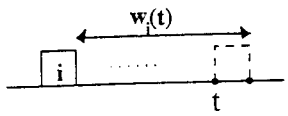

Figure 2: *Illustration of the parameter* $w_i(n)$

The evolution of $w_i(n)$ can be given as follows:

$$w_i(n+1) = \begin{cases} 1 & \text{if } u_n = i \\ w_i(n)+1 & \text{otherwise} \end{cases} \quad (2)$$

Assume that $w_i(0) = 1$ for $i = 1, ..., M$ without loss of generality.

The likelihood of page i being transmitted at n increases with λ_i and $w_i(n)$. We consider the policies where the broadcast scheduling is determined based on priority indices of the pages. The index of page i is the product $\lambda_i^\gamma \; w_i(n)$ where γ is an exponent that determines the relative importance of λ_i versus $w_i(n)$ in determining the priority.

The page scheduled to be broadcast at slot n is

$$u_n = arg \max_{i \in \{1,...,M\}} \lambda_i^\gamma \; w_i(n) \quad (3)$$

The above class of policies is called *priority index policies* in the following. Note that when all the pages have the identical request generation rates, the priority index policies for all γ's generate uniform periodic schedules which are the optimal in this case.

We performed an extensive numerical study of the performance of the system under the priority index policies for various values of γ and their mean response time is compared to the lower bound for a periodic broadcast schedule which is obtained in [3]. Numerical experiments are made for $M = 100$ to $M = 1000$ and $\gamma = 0$ to 1.0 for the case in which user access probabilities are assumed to follow zipf distribution [10]. According to the results from Table 1, the policy with $\gamma = 0.5$ yields the best performance which is also close to the lower bound. The policy with $\gamma = 0.5$ can be interpreted as follows. For $\gamma = 0.5$, the index of page i is $\lambda^{0.5} w_i(n) = \sqrt{\lambda_i w_i^2(n)}$. $\frac{1}{2}\lambda_i w_i^2(n)$ is the aggregate expected delay experienced by page i requests since the last time before slot n at which page i was broadcast. Hence, for $\gamma = 0.5$, the page with the largest *Mean Aggregate Delay* (*MAD*) is selected for transmission.

Although the algorithm proposed in [3] also yields the mean response time close to the lower bound,

Table 1: *Mean Response Time in* slots *for different values of* γ *using zipf distribution (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

MAD policy has a number of advantages over other existing methods for designing broadcast schedules. It is an on-line algorithm that can adapt to the changes in user's access pattern. Moreover, it can be easily generalized for multi-channel systems. It is also easy to implement and both the computational complexity and the storage requirement of the *MAD* policy is just $O(M)$. Furthermore, *MAD* policy has the important feature that the schedules it generates are periodic.

A PULL-BASED SYSTEM

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

Here, we consider a class of heuristic scheduling policies which are in the same flavor as the priority index scheduling policies for the push-based system and it is as follows:

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

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

Since, according to the simulation results in [4], the (*LWF*) policy, which elects the page for which the total waiting time of pending requests is the largest, yields significantly better response time characteristics than other heuristic policies, we compare the priority index policies to *LWF* policy by simulation. The results for 1000 pages is shown in figure 3.

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

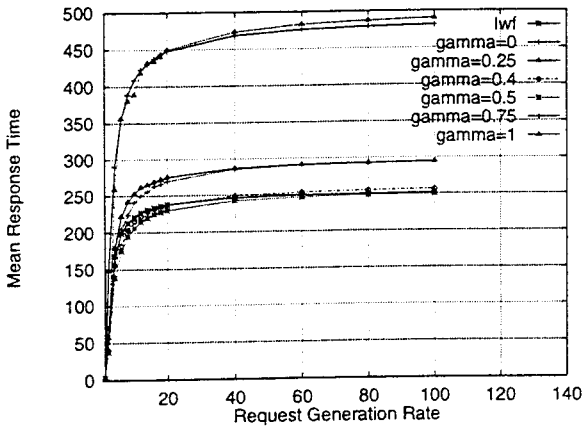

Figure 3: *Mean Response Time (in slots) vs. Aggregate Request Generation Rate (requests per slot) for different values of γ using zipf distribution for 1000 pages*

slightly better than the *LWF* policy) for all aggregate request generation rates. The policy with $\gamma = 0.4$ performs close to the *LWF* policy and the policy with $\gamma = 1.0$ gives the worst performance. In addition to its superior performance, the policy with $\gamma = 0.5$ is easier to implement than the *LWF* policy.

PERFORMANCE LIMITS OF A PULL-BASED BROADCAST SYSTEM

A pull-based system requires the availability of a uplink channel and has the undesirable property that the uplink channel may become overloaded under heavy aggregate request generation rate. Our simulation results 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.

Figures 4 shows the simulation results for equal request generation rate case (λ_i is the same for all pages). As the aggregate request generation rate increases beyond 20, the mean response time of the pull-based system approaches half of the total number of pages which happens to be the mean response time of the optimal schedule for the push-based system. It is also true for the case with unequal request generation rates (please refer to table 1 and figure 3).

USER'S MEMORY MANAGEMENT IN A PUSH-BASED SYSTEM

Consider a push-based system where each user has a cache that can hold K pages locally. The server is broadcasting the pages according to a fixed predeter-

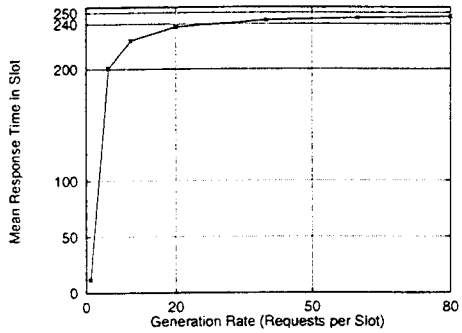

Figure 4: *Mean Response Time (in slots) vs. Aggregate Request Generation Rate (in request per slot) for 500 pages with equal generation rates*

mined schedule $\{u_n\}_{n=1}^{\infty}$. At the end of each slot n, the user may replace one of the pages in the cache with the page transmitted at slot n. We assume that all the users have the prior knowledge of the broadcast schedule of the server and follow an identical cache update strategy. Since, in addition, all of them monitor the same broadcasting server, the contents of the cache of all users are identical. The set of the K pages residing in the cache during slot n is represented by $C(n)$. The cache update strategy determines the cache contents at each slot and is represented by the sequence $\{C(n)\}_{n=1}^{\infty}$.

A request for page 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 page i broadcast that will be 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 page i is transmitted, which is illustrated in figure 5.

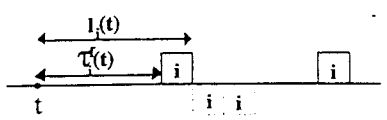

Figure 5: *Illustration of parameters, $\tau_i^f(t)$ and $l_i(t)$ for a sequence of page i broadcasts.*

The latency $l_i(t)$ of a page i request 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 the cache update strategy is to alleviate the impact of the latency on the user by maintaining in the cache the pages which are either

very likely to be requested by the user or will not appear in the broadcast for a long time. The sequence of times at which page i requests are generated is t_n^i, $n = 1, 2, ...$ for each page $i = 1, ..., M$. Let $L_i(t)$ be the aggregate latency of all page i requests generated from time 0 to time t, that is, $L_i(t) = \sum_{t_n^i \leq t} l_i(t_n^i)$. Let $\overline{L_i}(t)$, be the expected value of $L_i(t)$. Therefore, our objective is to design a cache management strategy which minimize the expected aggregate latency over all pages, $\overline{L}(t) = \sum_{i=1}^{M} \overline{L_i}(t)$.

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.

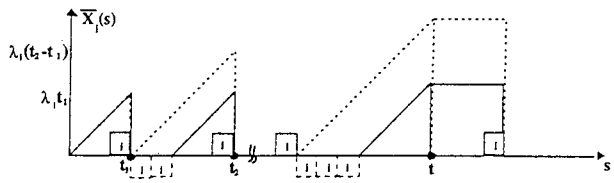

Figure 6: *The expected backlog of page i requests as a function of time when there is caching (solid line) is depicted. The expected backlog without caching (dotted line) is superimposed. The reduction in latency due to caching is equal to the area between the solid and the dotted lines.*

Figure 6 shows the evolution of the expected backlog of page i with caching (solid line) superimposed by the expected backlog without caching (dotted line) for a given sequence of page i broadcast. The shaded small rectangle corresponds to the slots at which page i is broadcast and the slots at which page i is cached are represented by the small dashed rectangles below the time axis. The aggregate expected latency for page 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 page i requests when there is no caching. The expected aggregate latency of page 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 6.

The aggregate latency over all pages 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

$$\sum_{n=0}^{t-1} \sum_{i \in C(n)} (\lambda_i \tau_i^f(n) + \frac{\lambda_i}{2}) \quad (4)$$

The maximization of the sum in equation (4) 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 page u_{n-1} broadcast during slot $n-1$, and the action taken by the update strategy. Hence, the set, $\mathcal{C}_n(C)$, of possible cache states at slot n, given that the cache state at slot $n-1$ is C, is

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

A feasible cache state evolution sequence is any sequence $\{C(n)\}_{n=1}^{t}$ with the property $C(n+1) \in \mathcal{C}_{n+1}(C(n))$. The corresponding cache update strategy is uniquely defined. Associated with each state c^j, there is a time-dependent "*reward*",

$$r(c^j, n) = \sum_{i \in c^j} (\lambda_i \tau_i^f(n) + \frac{\lambda_i}{2}) \quad (5)$$

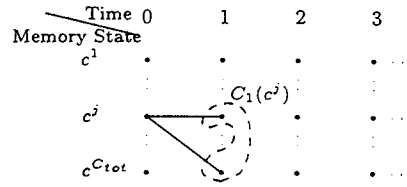

Figure 7: *Cache State Trellis Diagram*

Consider a trellis diagram one dimension of which is the cache state and the other is the time as shown in Figure 7. Each stage of the trellis corresponds to a certain time instance. All possible cache states appear in every stage. Since the total number of pages which are of interest to the user is M and its cache can hold K pages ($K \leq M$), the total number of possible cache states, C_{tot}, is $\binom{M}{K}$. There are directed links from

certain states in stage n to certain states in stage $n+1$, that represent possible cache 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 cache update strategy from slot 0 to slot t corresponds to a path from stage 0 to stage t in the trellis. The total reward, or latency reduction incurred by the strategy is equal to the sum of the rewards of each state in the path. The computation of the optimal cache update strategy is equivalent to the computation of a maximum reward path in the trellis.

Since the complexity of the optimal policy is prohibitive for employing the policy in a real time operation of the system, 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 readily suggested by the optimal policy. The optimal policy makes the cache 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 cache update decision at slot n can be made such that the cumulative average reward 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 $W = 1$. Let's call it *one-step look-ahead* (*OSLA*) policy. This policy updates the cache in each slot n such that the reward $r(C(n+1), n+1)$ is maximized for $C(n+1) \in \mathcal{C}_{n+1}(C(n))$. This policy turns out to be optimal in some special cases of interest. The first case is when all the pages have the same access probabilities. The other case is when the broadcast includes only two pages, the user has cache space for one page and the request generation rates are arbitrary. However, *OSLA* policy may perform poorly compared to the optimal policy in some cases.

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.

Remark

References

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

[2] 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.

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

[4] 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.

[5] R. J. Douglas (Program Manager). "Battlefield Awareness and Data Dissemination (BADD) Program". In *Web site at http://yorktown.dc.isx.com/iso/battle/badd.html*, 1996-2000.

[6] C. J. Su and L. Tassiulas. "Broadcast Scheduling for Information Distribution". *Proc. IEEE INFOCOM'97*, 1997. To appear.

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

[8] L. Tassiulas and C.-J. Su. "Memory Management Strategies for a Mobile User in a Broadcast Data Delivery System". *Proc. 30th Conf. on Information Science and Systems*, 1996.

[9] J. W. Wong. "Broadcast Delivery". *Proceedings of the IEEE*, 76(12):1566–1577, December 1988.

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