Supporting Mobile Device Communications In The Presence of Broadcast Servers*

Anup Mayank
Department of Computer Science and Engineering,
University of California Riverside, Riverside, California 92521, USA
E-mail: mayank@cs.ucr.edu

Chinya V. Ravishankar
Department of Computer Science and Engineering,
University of California Riverside, Riverside, California 92521, USA
E-mail:ravi@cs.ucr.edu

Abstract: Broadcast data dissemination is well-suited for mobile wireless environments, where bandwidth is scarce, and mutual interference must be minimized. However, broadcasting monopolizes the medium, precluding clients from performing any other communication. We address this problem in two ways. First, we segment the server broadcast, with intervening periods of silence, during which the wireless devices may communicate. Second, we reduce the average access delay for clients using a novel cooperative caching scheme. Our scheme is fully decentralized, and uses information available locally at the client. Our results show that our model prevents the server from monopolizing the medium, and that our caching strategy reduces client access delays significantly.

Keywords: Mobile Ad-hoc networks, Broadcasting, Cooperative Caching

1 Introduction

Mobile users are increasingly interested in a range of information, such as stock quotes, pricing information at shopping malls, weather, news and traffic information, schedules at bus stands, railway stations, and airports. Push-based information dissemination can be much more effective and scalable than the pull model in such applications. Its advantages are well-known [24, 23, 26, 2, 8, 21, 18, 9]. In contrast, the pull model requires clients to send each request to the server, and increases traffic and resource consumption at both clients and the server [21].

Broadcasting is a good choice for disseminating information in mobile wireless systems. However, when the medium is shared, as in the 802.11 protocol suite, broadcasting monopolizes the medium, preventing clients from performing any other communication. Blindly interrupting the broadcast to create slots for clients to communicate will increase client wait times proportionally, degrading an important performance metric. Wireless protocols do make several communication channels available, but it is better to treat the set of channels as a common bandwidth resource. We show how to share each channel between the broadcast program and other communications.

Work to date has arbitrarily assumed that clients perform no other communication besides listening to the broadcast. It has therefore focused merely on mechanisms to reduce client access times. Some approaches, for example, reduce latency by repeating popular items several times in a broadcast cycle [18, 9]. Others use client feedback [21] to reduce broadcast cycle length. Such approaches are inadequate since clients are still held captive by the broadcast server.

1.1 Broadcast Scenarios and Caching

Consider a hypothetical scenario with a number of mobile users with PDAs in a shopping mall. To help customers and improve sales, the mall broadcasts a variety of information such as mall maps, store names, prices, sales under way, advertisements, and so on. Customers use their PDAs

*Supported by grants from Tata Consultancy Services, the DiMI program of the University of California, and contract F30602-01-2-0535 of DARPA's FTN program.

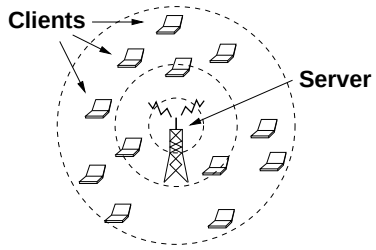


Figure 1: Broadcast preempts other communications.

to listen to the broadcast, and wait for information relevant to their own shopping objectives (see Figure 1).

However, if the mall broadcasts constantly, it will effectively be ''jamming" the medium, making any other communication impossible for user PDAs. The mall will surely lose customers and revenues. Interrupting the broadcast to allow PDAs to communicate increases wait latencies, prolongs wait times, aggravate clients, and also lose revenues for the mall.

Clients can also reduce latency by caching data. Local caching methods such as prefetching [18] do reduce average access latency, but are limited by the amount of storage at each node. It would be better for caches to cooperate.

With cooperative caching, it suffices for each node to cache only a subset of items. When a desired item is absent from its local cache, a node tries to locate it in one of the other caches, using a lookup mechanism. Work already exists on cooperative caching in mobile wireless systems [9, 5, 7, 4]. Unfortunately, such approaches require caches to exchange messages to maintain global state, which is impossible when the broadcast server monopolizes the medium. Besides, message exchanges introduce considerable overhead. We do not see such models as appropriate for the environments we consider.

1.2 Our Contributions

We address these issues for mobile ad-hoc networks using 802.11-style protocols, where a shared medium is used for all communication. There are two interesting aspects to our approach. First, it segments the server broadcasts, dividing the broadcast cycle into a fixed number of segments interleaved with periods of silence, during which clients can communicate among themselves. Second, we propose a fully decentralized cooperative caching mechanism for use by clients, which requires no message exchanges for maintaining information on cache contents. Earlier work on caching for push-based dissemination are either non-cooperative [18] or are based on information exchanges between mobile nodes [9] for maintaining cache states. We show that our mechanism greatly reduces average object access delays, and is both fault tolerant and scalable.

2 Related Work

Each mobile node in the COCA [7] architecture, is either in a High Activity (HA) or Low Activity (LA) state. When its state changes from HA to LA, it retrieves and caches a set of less frequently accessed objects. More frequently accessed items would be already cached in the local cache of HA nodes, which can access replicas of less frequently accessed data items cached nearby. They do not explicitly address the issue of the server blocking out client communications.

Client tune-in time is the time a client remains in active mode to retrieve desired objects from the broadcast. Energy usage increases with tune-in time. Several broadcast schemes have been proposed to minimize client tune-in times, such as the hashing scheme of [10], the B+ tree based indexing scheme of [11], the Alphabetic Huffman tree (AHT) of [17] and the Prediction based indexing scheme of [12]. However, such methods do not address client-side caching issues, and retain the strategy of continuous broadcasts, preempting all other communications by clients.

A scheduling algorithm for correlated data is proposed in [25], in which data is accessed in a group, and a caching strategy is used at the client to improve the performance of the scheduling mechanism. However, this caching approach involves no cooperation among clients. Besides, their broadcast is continuous, and preempts all client communications.

Data is cached at the routing layer in CachePath and CacheData [5]. When one node routes a data item to another, it caches either the path to the cached data item or the data item itself, depending on the distance to server, the caching node, and route stability.

In GCLP [19], the server periodically sends information about its contents and physical location to a set of selected nodes called content location servers. A client locates an object by sending a query along suitable routes. When the query reaches a content location server, it returns the name of the nearest content server to the client. The client can obtain the requested object from the server.

The scheme in [16] allocates data to the servers, based on the movement pattern of users, so that a mobile user can obtain most recent replica from a nearby server, instead of sending requests to multiple hop away main server.

In [18], the local cache at the client end is used to store data items prefetched from the broadcast cycle. This scheme considers only the single cache at each client. It is not suitable if the cache size is small and total number of broadcasted data items is large.

The approach in [9] uses a cooperative caching approach among mobile clients in broadcast based information system. Mobile clients rely heavily on message exchanges among themselves for caching of data items. Unfortunately, [9] uses a continuous broadcasting model, so the broadcast medium is monopolized.

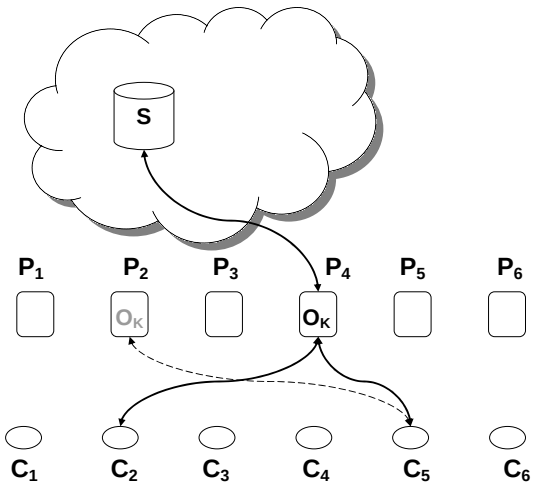


Figure 2: Highest Random Weight cache selection

3 Hash-Based Cooperative Caching

Let $C_1, \ldots, C_n$ be the names (or IDs) of a set of cooperating caches. Cooperative caching will be most effective if any given object O_k were never cached at more than one of these caches, and if each client could independently determine the identity of this cache. The Higest Random Weight (HRW) approach [20] was the first to show how to use hashing to allow clients to agree, with no communication, about which cache should hold an object O_k. A related idea appeared subsequently under the name of consistent hashing [13].

Consider Figure 2, in which the set of clients $C_1, C_2, \cdots, C_6$ retrieve objects from a remote server S. To reduce latency and speed up access, we use a set of caches (or proxies) $P_1, P_2, \cdots, C_6$ which retrieve objects from S on demand, and cache them locally. Each client C_i chooses a proxy P_j, from which it attempts to retrieve each desired object O_k. The specific choice of proxy by each client can have a very significant effect on the performance of this setup.

Let us say that the client C_2 needs the object O_k at some time, and that the client C_5 subsequently needs the same object O_k. If C_2 selects P_4 and C_5 selects P_2, we will have misses at both P_2 and P_4, assuming that the caches are cold. There will be two accesses to the remote server S, and both C_2 and C_5 will see long delays. Besides, the object O_k will be cached at both P_2 and P_4, wasting space.

Clearly, it would be best for client to agree on the proxy cache that they will all access for each such object O_k. We want to distribute object requests uniformly across all proxy caches, and also minimize the number of objects replicated across these caches, with no replication in the ideal case. Finally, we want to be able to accommodate proxy cache failures, so that all clients will default to the same alternative cache if their common first-choice cache is unavailable.

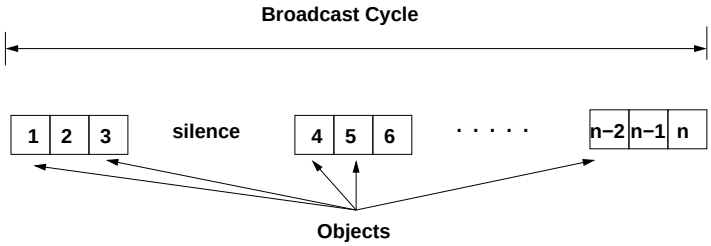


Figure 3: Segmented broadcasting model

The HRW approach [20] presents an elegant and efficient solution to this problem. It proceeds by using a hash function H as follows. Each client computes the n hash values $H(C_1, O_k), \cdots, H(C_n, O_k)$ independently, and selects π_k, the cache that that yields the highest hash value. This π_k is called the *prime cache* in the cluster for the object O_k. Conversely, all objects O_k^i that have π_k as their prime are called *prime objects* for π_k.

Since all clients apply HRW using the same H and to the same cache cluster $C_1 \ldots, C_n$, each client will independently select the same π_k as the prime for any given O_k. In the example in Figure 2, C_2 and C_5 will independently compute the values $H(P_1, O_k), H(P_2, O_k), \cdots, H(P_6, O_k)$, and pick the highest of these values. In our example, the highest value is obtained for $H(P_4, O_k)$, so C_2 and C_5 will both request P_4 for the object O_k.

The crucial idea in HRW is to always cache the object O_k only at π_k, the prime cache for O_k in the group $C_1, \cdots, C_n$. This strategy ensures that there is no object duplication across the cache cluster, and optimizes space utilization. Since clients can agree without any communication on which cache to use, the scheme is very efficient. The work in [20] discusses suitable H, and shows that HRW is very efficient and effective, and that it randomizes very well, so that every cache is likely to be the prime for the same number of objects.

HRW is also very robust in the face of cache failures. If the clients find that the prime π_k for some object O_k has become inaccessible, they simply pick the cache π_n' that yields the next highest hash value. Now, any request assigned to π_k becomes automatically reassigned to π_k' at each client. Since HRW is randomizing, each of the remaining proxies receives an equal share of these reassignments, ensuring that cache loads continue to be balanced.

When a cache C_i comes back up or is added to the cluster, the objects reassigned to it are exactly those which yield a higher hash value for C_i than any other cache in the cluster. Thus, HRW ensures that the fewest possible number of objects are reassigned in the case of cache failures or cache addition.

4 Segmented Broadcasting and Hash-Based Caching

Our approach divides the broadcast cycle into segments,

with intervening periods of silence. The server broadcasts a subset of objects in each segment, but yields the medium to clients during the inter-segmental silences. With longer silences, the clients will have more use of the medium, but must tolerate longer wait times for receiving objects from the broadcast. The segment size and silence interval are parameters to be adjusted according to the number of objects to be broadcasted, average object size, the client interest patterns, and waiting times considered reasonable. There will be frequent intervals when the medium is free, even with large broadcast data sets. Clients can use these intervals for communication, and for obtaining desired objects from peers.

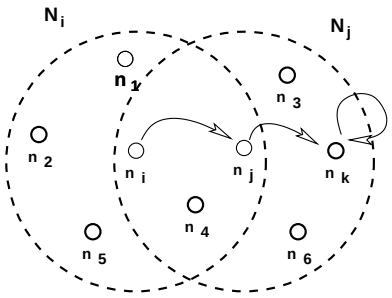


Figure 4: HRW probes of 1-hop neighborhoods.

4.1 Cooperative Caching Using Local Information

We use cooperative caching to reduce access delays, and to allow clients to obtain data items from other clients in their neighborhood, rather than wait for the server to broadcast it. To be effective, cooperative caching must minimize object replication across caches, and incur low communication overhead. This can be tricky to accomplish in mobile ad-hoc systems, since mobile ad-hoc environments are dynamic, with nodes joining and leaving the system as they please.

A node may not know all other nodes in the system, but it usually knows its 1-hop neighborhood. It is typical, for instance [19], for 1-hop neighbors to exchange periodic hello messages. Moreover, 1-hop neighborhoods are likely to remain stable over short or medium durations, as in the case of passengers waiting for a particular flight or train, or customers near a particular exhibition stall.

Each node has a cache of limited size for holding a subset of the objects broadcasted by the server, so that replication within a neighboring set of mobile nodes must be minimized. It has been typical, as in [9], for caches to control replication by communicating their contents to each other and agreeing on a global caching policy. However, this method requires a great deal of communication, and is unrealistic for power- and bandwidth-limited wireless systems.

4.1.1 HRW Search of 1-Hop Neighborhoods

When node n_i needs an object O, it first checks its own cache. If O is not found, n_i determines the next broadcast time for O from the broadcast schedule. (It is typical [18, 9] for the server to periodically disseminate its schedule for broadcasting objects. This schedule can be cached and shared with other clients.) If this delay is too high, n_i tries to locate O in its peer caches as follows.

Let the set $N_i = \{n_{i1}, \ldots, n_{ik}\}$ be the set of 1-hop neighbors of n_i. Node n_i applies HRW, and selects the node n_j yielding the highest hash value. From n_i's perspective, n_j is the prime for O in the set N_i, so that O would be at n_j if it were in the set N_i. Therefore, n_i sends n_j a request for the object O. If n_j has O, it returns O to n_i. Otherwise, n_j continues the search, using HRW on its own 1-hop neighborhood N_j.

As the search continues, n_j may itself turn out to be the prime cache for O within its own neighborhood N_j, in which case the search bottoms out. Such a search can be continued until a node is prime for its own neighborhood, (as is n_k in Figure 4), or for a fixed number of hops from n_i.

We choose to continue the search through n_j rather than some randomly chosen $n_k \in N_i$ for a specific reason. Since n_j is the prime for O in the neighboorhood N_i, subsequent requests for O in N_i will also be sent to n_j. Continuing the search through n_j allows n_j to cache O, increasing future hit rates.

4.2 Caching of Objects and Cache Replacement

Each object O_i obtained by n_j is cached at n_j. If n_j's cache is full, a victim object is selected, as explained below, and evicted to make space for O_i. If a node receives a request for object O, it is likely to soon receive another request for O, due to locality effects. Hence, nodes cache all arriving objects, even if they are not prime objects.

4.2.1 Cache Replacement Mechanism

Since wireless devices have limited storage capacity, the cache replacement policy can have a major influence on performance. Our cache replacement policy favors popular and prime objects, increasing hit rates and reducing average access latency.

Our replacement policy tries to preserve prime objects, since they are likely to be requested by peers. When a cache must make space, it will first evict non-prime objects, starting with those having the lowest access probability. If no non-prime objects are present, some prime object must be evicted. Prime objects are also evicted starting with those having the lowest access probability.

The access probability function of an object O_i is calculated as

$$\psi(O_i) = \min\{1, \frac{T_{avg}}{T_r - T_c}\},$$

where T_{avg} is the cumulative average request arrival interval of the object, T_r is its last reference time, and T_c is the current time. T_{avg} is recomputed as

$$T_{avg}^{new} = \beta * T_{avg}^{old} + (1 - \beta)(T_r - T_c),$$

where β is a positive constant less than 1. In our experiments we have used $\beta = 0.5$. Our access probability function is similar to those used in [22, 6].

4.3 Analysis of Search Method

As in Figure 4, let n_i and n_j have 1-hop neighborhoods N_i and N_j respectively. Let a request for object O originate at n_i, and let the application of HRW yield $n_j \neq n_i$ as prime in N_i, and n_k as prime in N_j. Since HRW orders nodes linearly, n_j and n_k can not both be in N_i, unless $n_j = n_k$. The probability that $n_j = n_k$ is high when the neighborhoods N_i and N_j share most of their nodes, that is, when $N_i \cap N_j$ has higher cardinality than $N_i \setminus N_j$ and $N_j \setminus N_i$.

Let x be the distance between n_i and n_j, and let R be the transmission range of a node. Then the area of intersection $A(x)$ of 1-hop neighborhoods is

$$A(x) = 2R^2 \cos^{-1}\left(\frac{x}{2R}\right) - \frac{x\sqrt{4R^2 - x^2}}{2}$$

If $A = \pi R^2$ is the area of a single 1-hop neighborhood, the fraction of overlap area at any separation x is $A(x)/A$. Since x can vary in the range $(0, R)$, we can obtain the expected fraction of overlap area over the range $(0, R)$ as

$$\frac{1}{R}\int_0^R \frac{A(x)}{\pi R^2}\,dx = \left| \frac{\left(4R^2 - x^2\right)^{\frac{3}{2}}}{6\pi R^3} - \frac{4}{\pi}\sqrt{1 - \frac{x^2}{4R^2}} + \frac{2x}{\pi R}\cos^{-1}\left(\frac{x}{2R}\right)\right|_0^R \approx 0.6884$$

If nodes are uniformly distributed, the number of nodes in a region is roughly proportional to its area. Consequently, as the search progresses, the expected overlap between two successive 1-hop neighborhoods is more than 68%. For a given object O, n_j will be picked randomly in N_i, since HRW is randomizing. Thus, the probability $\Pr[n_k = n_j]$ of the HRW search stopping at any given step exceeds 0.68. If we model the search as a series of Bernoulli trials, each with a probability $p = 0.6884$ of success, the expected number of trials to success is given by the mean of the Geometric distribution $G(k) = (1-p)^k p$, which is simply $\frac{1}{p} = \frac{1}{0.6884} \approx 1.45$. Thus, the search will bottom out quickly when the node distribution is uniform and sufficiently dense.

5 Experiments and Results

Our goal is to minimize the average *access delay* for clients, that is, the time to retrieve a document, either from the peer nodes or from broadcast by the server. Our experiments show that our mechanism greatly reduces access delays.

Parameter	Range	Default
# Objects (D)	500–5000	1000
# Mobile nodes (N)	25–200	100
Node velocity (V)	0.0–2.0 m/s	1.0 m/s
Zipf parameter	0.0–1.0	0.7
Inter segment delay	10–120 sec	60 sec
Cache size (% of total object size)	5%–40%	20%
Segment size (# objects)	100–500	100

Table 1: Simulation parameters

α	h=0	h=1	h=2	h=3	h≥4
1.0	19.6	60.7	68.8	69.8	70.0
0.8	13.5	46.0	52.4	53.5	53.7
0.6	9.1	32.1	36.8	37.6	37.7
0.4	5.6	21.1	24.5	25.1	25.2

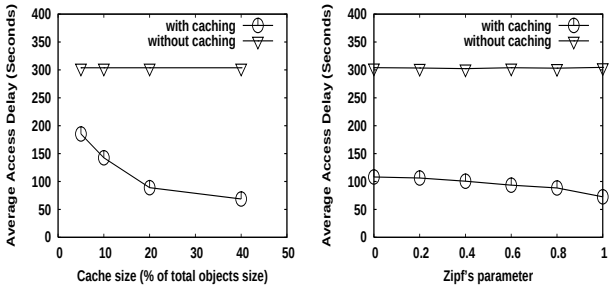
Table 2: Hit rates

We used ns-2, version 2.26 [1] in our experiments. N mobile nodes with 802.11 MAC layer, were randomly dispersed in a $2000m \times 2000m$ square, each moving according to the random waypoint model [3]. A stationary server with a range covering this entire region broadcasted a set of D objects at a bandwidth of 11 Mbps. Each object is of size of 1KB. At each step, a node moved to a random destination at a randomly chosen velocity between $(0, 2.0)$ m/s, and remained there for a pause time of 1 minute.

The bandwidth between mobile nodes was 2Mbps, and the communication range was 250 m. Requests were generated by randomly selecting a node n_i, and having it generate a request for an object O_j according to the Zipf popularity model [27]. O_j will ultimately be retrieved from peer nodes or from the broadcast. Requests were distributed exponentially with an average rate of 10 requests per second. For each set of experiments, 10,000 requests were generated and average delay between request generation and object retrieval was computed.

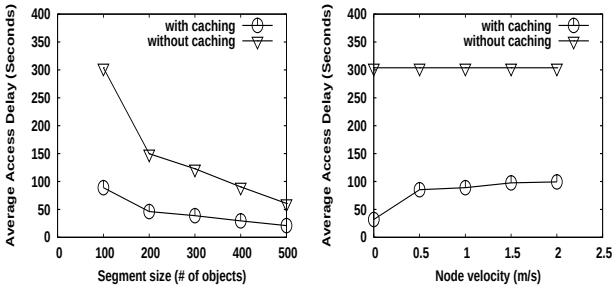
5.1 Hop Count and Hit Rates

We first evaluated how the hit rate increased with the number of hops. This is an important metric, since going to nodes farther away also increases access delays and traffic. We varied the Zipf parameter α, but kept all other parameters at their default values (Table 1). Table 2 shows the cumulative object hit rate as we go h hops away from the source. We see that hit rates are quite high for small h, especially when α is high. It also suffices to go 3 hops, since the increase in hit rates for more hops is marginal. Our search mechanism is extremely efficient. We can achieve even better performance by increasing the cache size and number of nodes.



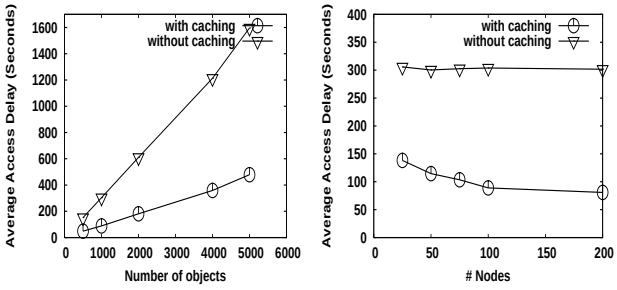
(a) Cache Size (b) Zipf parameter

Figure 5: Effects of cache size and Zipf parameter



(a) Segment size (b) Node mobility

Figure 6: Effects of segment size and node mobility



(a) Broadcast length (b) Node density

Figure 7: Effects of broadcast length and node density

5.2 Effect of Cache Size

Larger caches allow mobile nodes to store more objects, thus more objects are served from the local and peer cache. This leads to a reduction in average access delay observed by nodes, and this effect is quite obvious in Figure 5(a).

5.3 Effect of Zipf Parameter

The Zipf parameter α is a measure of skew in object popularity. Higher α indicates that some objects are requested more frequently than others. Caching popular objects minimizes average access delays.

Our caching mechanism favors caching prime and popular objects, reducing average access delays. This effect is clearly shown in 5(b). Average access delay decreases from 108.15 seconds to 72.58 seconds, as the value of α is changed from 0.0 to 1.0, whereas without caching average access delay is around 303 seconds.

5.4 Effect of Segment Size

Segment size measures the number of objects in each segment. The total number of segments in a broadcast cycle decreases as segment size increases, so that the number of inter-segment silence periods decreases. Our inter-segment silences are greater than the broadcast time of a segment, so that the broadcast cycle length decreases with an increase in segment size.

Increasing segment size has both positive and negative effects. Larger segments result in lower average access delay (see Figure 6(a)), but the wireless channel is occupied for a longer period of time by the broadcast server, which may result in longer wait times for inter-client communications. Segment size is a tunable parameter, and can be set for optimum performance.

5.5 Effects of Node Mobility

The neighborhood set of a mobile node is always changing, so that the prime for any object will also change. Thus, the probability of finding an object in the neighborhood of a mobile node decreases with mobility, increasing average access delay. Discussions in [15, 14] have shown that users of a wireless LAN can be safely assumed to be stationary. As Figure 6(b) shows, the performance of our method remains surprisingly stable over a large range of node velocities.

5.6 Effect of Number of Objects

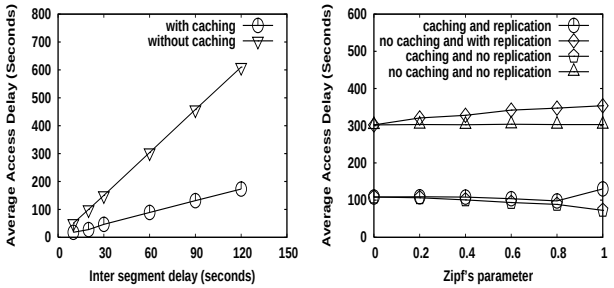
Increasing the total number of broadcasted objects increases the number of segments, and hence the total broadcast cycle length. This effect is shown in Figure 7(a). However, the average access latency can be reduced by increasing the segment size, as Figure 6(a) shows.

5.7 Effect of Node Density

The number of nodes in each 1-hop neighborhood increases with node density. Consequently, each node is prime for fewer objects, and must cache fewer prime objects. It can use rest of the cache space to cache other popular non-prime objects. This leads to an improvement in performance as is evident from 7(b).

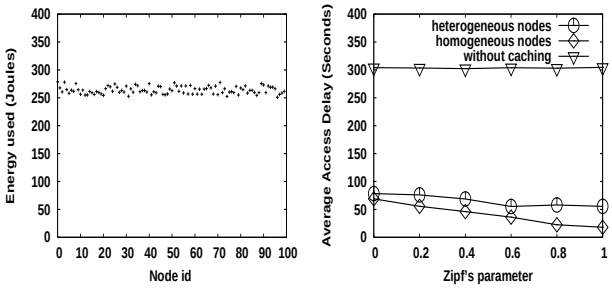
5.8 Effect of Inter-Segment Delay

The broadcast cycle length increases as inter segment delay increases, so that average access latency increases. How-



(a) Intersegment delay (b) Zipf parameter

Figure 8: Effects of intersegment delay and object replication.



(a) Energy Usage (b) Heterogeneous Network

Figure 9: Energy usage and effect of node heterogeneity.

ever, longer the inter segment delays allow mobile devices greater flexibility to communicate during the silences. Since clients can obtain objects from peers, performance degrades only moderately with our mechanism (see Figure 8(a)) as inter-segment delay increases.

5.9 Broadcast With Object Replication

Servers can reduce access latencies by broadcasting an object several times in each cycle, in proportion to its popularity [21]. We assume that the server knows object popularities, and creates the broadcast schedule as in [21]. We have compared the replicated broadcast scheme without caching and our caching scheme without replication. Figure 8(b) shows that our caching scheme outperforms object replication in broadcasts. We can achieve even better results by combining caching with replication.

5.10 Energy Usage of Mobile Nodes

Our hash-based caching approach distributes caching load uniformly among all nodes. This claim is confirmed by experiments. Figure 9(a) shows the energy used by each mobile node at the end of simulation, as determined by the `EnergyModel` function of ns-2 [1] in this set of experiments. Transmission and receive power for each 512-byte packet has been set to 0.6W and 0.3W respectively. For the idle and sleep modes, power usage of mobile nodes has been set to 0.05W and 0.01W, respectively. Our experiments demonstrate that loads, and hence object requests were uniformly distributed among peer mobile nodes and thus leading to similar power usage at all mobile nodes.

5.11 Effects of Heterogeneity

In practice, mobile nodes are likely to be heterogeneous in power and range. Devices like PDAs are likley to rank at the low end of this spectrum, while laptop computers are likely to be at the high end. To understand the effectiveness of our caching scheme in a heterogeneous network, we conducted two set of experiments. In the first set, the transmission and reception radius for a mobile node was randomly assigned a value in the range 100–600 m. In the second set, the transmission and reception radius of nodes was 250 m. We kept the nodes stationary to isolate the effects of node heterogeneity from the effects of mobility. We find moderate increases in average access delay in case of heterogeneous network (see Figure 9(b)). We can attibute this to two reasons. First, lower-capacity nodes have fewer nodes in their 1-hop neighborhood, increasing their dependence on the broadcast server and on peer nodes, which are farther away. Second, higher-end nodes have larger transmission radius, leading to higher interference with the communications for nodes in their 1-hop neighborhood.

6 Conclusion and Future Work

We have argued that the standard approach of continuous broadcasts is unsuitable for typical wireless environments. The server monopolizes the medium, preventing communication among clients. We have addressed this issue by advocating the use of segmented broadcasts. Wireless devices can communicate among themselves in the silence period between two consecutive broadcast segments.

We counteract increases in access times due to these silences using a novel cooperative caching scheme that allows clients to obtain objects from peers, rather than from server broadcasts alone. Our approach uses limited cache space at clients effectively, and uses a hash-based mapping function, making it completely decentralized. Earlier approaches have required inter-cache cooperation, which is an unreasonable requirement in wireless broadcast schemes. Our experimental results clearly demonstrate the efficiency of our approach.

A standard approach in broadcasting is to replicate each object in proportion to its popularity. Although this scheme reduces the average object access delay, we have been able to achieve similar performance by our caching approach, even without the use of replication. In our current model, objects are assumed to be accessed independently of each other. The study of the effects of correlated

data will be part of future work.

REFERENCES

[1] Ns. http://www.isi.edu/nsnam/ns/.

[2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: data management for asymmetric communication environments. *SIGMOD Rec.*, 24(2):199–210, 1995.

[3] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[4] J. Cai, K.-L. Tan, and B. C. Ooi. On incremental cache coherency schemes in mobile computing environments. In *Proceedings of the Thirteenth International Conference on Data Engineering*, pages 114–123. IEEE Computer Society, 1997.

[5] G. Cao, L. Yin, and C. R. Das. Cooperative cache-based data access in ad hoc networks. *IEEE Computer*, 37:32–39, 2004.

[6] S. Chen, B. Shen, S. Wee, and X. Zhang. Adaptive and Lazy Segmentation Based Proxy Caching for Streaming Media Delivery. In *Proceedings of the 13th international workshop on Networks and operating systems support for digital audio and video*, number 1-58113-694-3, pages 22–31, June 2003.

[7] C.-Y. Chow, H. V. Leong, and A. Chan. Peer-to-peer cooperative caching in mobile environments. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*. IEEE Computer Society, 2004.

[8] S. Hameed and N. H. Vaidya. Efficient algorithms for scheduling data broadcast. *Wirel. Netw.*, 5(3):183–193, 1999.

[9] T. Hara. Cooperative caching by mobile clients in push-based information systems. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 186–193. ACM Press, 2002.

[10] T. Imielinski, S. Viswanathan, and B. Badrinath. Power efficient filtering of data on air. In *International Conference on Extending Database Technology*, pages 245–248, 1994.

[11] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficient indexing on air. In *Proceedings of the ACM SIGMOD Conference*, pages 25–36, 1994.

[12] K.-F. Jea and M.-H. Chen. A data broadcast scheme based on prediction for the wireless environment. In *Proceedings of the Ninth International Conference on Parallel and Distributed Systems (ICPADS)*, 2002.

[13] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web Caching with Consistent Hashing. In *Proceedings of 8th International World Wide Web Conference*, May 1999.

[14] D. Kotz and K. Essien. Analysis of a campus-wide wireless network. In *Proceedings of International Conference on Mobile Computing and Networking(MOBICOM)*, pages 107–118, 2002.

[15] P. Nuggehalli, V. Srinivasan, and C.-F. Chiasserini. Energy-efficient caching strategies in ad hoc wireless networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 25–34, 2003.

[16] W.-C. Peng and M.-S. Chen. Allocation of Shared Data based on Mobile User Movement. In *Proceedings of the Third International Conference on Mobile Data Management (MDM'02)*, 2002.

[17] N. Shivakumar and S. Venkatasubramanian. Efficient indexing for broadcast based wireless systems. In *Mobile Network and Applications*, pages 433–446, 1996.

[18] C.-J. Su and L. Tassiulas. Joint Broadcast Scheduling and User's Cache Management for Efficient Information Delivery. In *Proceedings of International Conference on Mobile Computing and Networking(MOBICOM)*, 1998.

[19] J. B. Tchakarov and N. H. Vaidya. Efficient content locations in wireless ad hoc networks. In *Proceedings of the IEEE International Conference on Mobile Data Management (MDM'04)*. IEEE Computer Society, 2004.

[20] D. G. Thahler and C. V. Ravishankar. Using Name-Based Mapping to Increase Hit Rates. *IEEE/ACM Transactions on Networking*, 6:1–13, February 1998.

[21] W. Wang and C. V. Ravishankar. Adaptive data broadcasting in asymmetric communication environments. In *Proceedings of 8th International Database Engineering and Applications Symposium (IDEAS)*, pages 27–36. IEEE Computer Society, 2004.

[22] K. L. Wu, P. S. Yu, and J. L. Wolf. Segment-Based Procy Caching of Multimedia Streams. In *Proceedings of the tenth internation conference on World Wide Web*, number 1-58113-348-0, pages 36–44, May 2001.

[23] B. Xu, O. Wolfson, and S. Chamberlain. Cost based data dissemination in broadcast networks with disconnection. In J. V. den Bussche and V. Vianu, editors, *Database Theory - ICDT 2001, 8th International*

Conference, London, UK, January 4-6, 2001, Proceedings, volume 1973 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2001.

[24] B. Xu, O. Wolfson, S. Chamberlain, and N. Rishe. Cost based data dissemination in satellite networks. *Mob. Netw. Appl.*, 7(1):49–66, 2002.

[25] E. Yajima, T. Hara, M. Tsukamoto, and S. Nishio. Scheduling and Caching Strategies for Correlated Data in Push-based Information Systems. volume 9, pages 22–28. ACM Press, 2001.

[26] W. G. Yee, S. B. Navathe, E. Omiecinski, and C. Jermaine. Efficient Data Allocation over Multiple Channels at Broadcast Servers. In *IEEE Transactions on Computers*, October 2002.

[27] G. K. Zipf. *Human Behavior and the Principles of Least Effort*. Addison-Wesley, Cambridge, MA, 1949.