Middleware for Scalable Data Dissemination*

Panos K. Chrysanthis† Vincenzo Liberatore‡ Kirk Pruhs†

1 Introduction

A major problem on the Internet is the scalable dissemination of information. This problem is particularly acute exactly at the time when the scalability of data delivery is most important, e.g., election results on the night of the 2000 United States presidential election, and news during 9/11/2001. The current unicast pull framework simply does not scale up to these types of workloads. One proposed solution to this scalability problem is to use multicast communication. Multicast introduces many non-trivial data management problems, which are common to various data dissemination applications. Multicast data management issues can be addressed by a middleware such that

- The middleware would free application developers from the need of implementing in each application a common set of data
- The middleware would free application developers from the details of the underlying multicast transport, management algorithms,
- The middleware would implement and unify high-performance state-of-the-art data management methods and algorithms while hiding their complexity from the application developer.

The chapter describes the design, architecture, implementation, and research issues of such a middleware. Particular emphasis will be placed on previous and current research to improve the performance of the middleware implementation. Current applications of the middleware include the scalable dissemination of Web contents and a healthcare alert system, and are described in this chapter.

The middleware addresses the following cored data management issues in multicast-enabled data dissemination applications:

Document selection. Select documents for which broadcast is appropriate from those documents for which different dissemination methods are preferable.

Scheduling. Decide how frequently and in which order documents are multicast.

Consistency. Support currency and transactional semantics for updated contents.

Cache Replacement. Broadcast data can be cached at the client site, and local caches are managed by replacement policies tailored to the broadcast environment.

*This work has been supported in part under NSF grants ANI-0123929, ANI-0123705 and CCR-0098752.

†Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260. {panos, kirk}@cs.pitt.edu

‡Division of Computer Science, Case Western Reserve University, 10900 Euclid Ave., Cleveland, Ohio 44106-7071. vincenzo.liberatore@cwru.edu

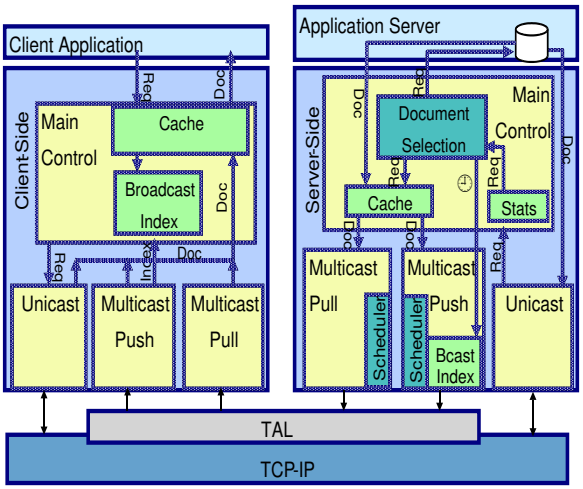


Figure 1: Middleware data dissemination architecture.

Hybrid schemes. Support client-initiated requests to complement the multicast approach.

Indexing. Reduce client waiting and tuning time by advertising transmission times.

The middleware architecture is built from individual components that can be selected or replaced depending on the underlying multicast transport mechanism or on the application needs. A prototype will shortly be made available at `http://dora.cwru.edu/mware/` and `http://db.cs.pitt.edu/mware/`.

The chapter is organized as follows. Section 2 overviews the middleware components, their relationship, and gives an example of the middleware operations. Section 3 reviews previous work on data dissemination systems. Section 4 discusses each building block within the middleware in terms of implementation, state-of-the-art research, and open research problems. Section 5 discusses the integration of the various data management operations. Section 6 describes an application of middleware for data dissemination. Section 7 concludes the chapter.

2 Architecture Overview

The outline of a possible configuration of the middleware and its relationship with the application and transport layers is shown in Figure 1. The configuration demonstrates the main building blocks, each tied to a particular data management function in multicast environments (document selection, scheduling, consistency, caching, and indexing). The middleware has server-side and client-side components; most of the server-side building blocks have a corresponding module on the clients and vice versa. Sections 4 and Section 5 will describe in details each one of these components and their integration. The rest of this section gives a motivating example and demonstrates the purpose of each one of the middleware building blocks.

The application is a highly scalable Web server and, in the example, we assume that the underlying transport is IP multicast. The following discussion closely follows the original work of Almeroth *et al.* [8]. The objective of the Web server application is to scale to a large client population, and scalability is accomplished by using the data dissemination middleware. The configuration in Figure 1 enables hybrid dissemination schemes, whereby the server can disseminate data by choosing any combination of the following three schemes: *multicast push*, *multicast pull*, and *unicast push*. The end-user should not perceive that Web resources are downloaded with a

variety of methods, as the browser and the middleware shield the user from the details of the hybrid dissemination protocol.

Multicast Push. In multicast push the server repeatedly sends information to the clients without explicit client requests. (For example, television is a classic multicast push system.) Multicast push is an ideal fit for asymmetric communication links, such as satellites and base station methods, where there is little or no bandwidth from the client to the server. For the same reason, multicast push is also ideal to achieve maximal scalability of Internet hot spots. Hence, generally multicast push should be restricted to hot resources.

Multicast Pull. In multicast pull, the clients make explicit requests for resources, and the server broadcasts the responses to all members of the multicast group. If the number of pending requests for a document is over some threshold, then the server transmits the document on the multicast pull channel, instead of individually transmitting the document to each of the requesting clients. One would expect that this possibility of aggregation would improve user perceived performance for the same reason that proxy caches improve user perceived performance, that is, it is common for different users to make requests to the same resource. Multicast pull is a good fit for "warm resources" for which repetitive multicast push cannot be justified, while there is an advantage in aggregating concurrent client requests [8]. Further, multicast pull is particularly helpful when some documents quickly become hot [17].

Unicast Pull. Unicast pull is the traditional method to disseminate data and is reserved for cold documents.

We now explain how the middleware client interacts with these three dissemination methods. The middleware client is invoked when the application client requests a document. The client middleware first checks to see if the document is cached locally, and if so, the client middleware returns the document to the application client. Otherwise, the middleware client next checks the latest copy of the multicast push index, which it stores locally, to check whether the document is being broadcast on the multicast push channel. Each entry of the multicast push index is associated with a flag that is used to indicate which of the documents in the current broadcast will be dropped from the next broadcast. If the document is listed in the index to be broadcast, then the client middleware waits for the document on the multicast push channel. With some small probability the client makes an explicit request to the server so that the server can still maintain estimates of popularity for those items on the multicast push channel. When the middleware client learns that the document it not on the multicast push channel, it opens a TCP connection to the server and explicitly requests the document using HTTP. After making this request, the client needs to listen for a response on all of the channels. The document may be returned on the multicast push channel if the document was recently became hot. The document may be returned on the multicast pull channel if several requests for this document arrived nearly simultaneously at the server. And the document may also be returned over the TCP connection that the client has with the server. The client might also observe the TCP connection being closed without the document being returned in the case that the server has determined that the document will not be served to this client via unicast.

The hybrid data dissemination scheme is supported by additional components.

Document Selection. In the Web server application, the *document selection* unit periodically gathers statistics on document popularity. Once statistics have been collected, the server partitions the resources into *hot*, *warm*, and *cold* documents. When a client wishes to request a Web document, it either downloads it from a multicast group or it requests the document explicitly. In the former case, the client needs to find on which multicast address the server transmits hot resources. Multicast address determination can be accomplished with a variety of schemes including an explicit http request followed by redirection to the appropriate multicast address, hashing the URI to a multicast group [8], using a well-known multicast address paired to the IP address of the origin server in single-source multicast [38], or application-level discovery in the case of end-to-end multicast.

Bandwidth Allocation. This component determines what fraction of the server bandwidth is devoted to servicing documents via the multicast push channel, and what fraction of the server bandwidth is devoted to servicing documents through the pull channels. This component is closely related to the document selection component. We use an adaptation of an algorithm proposed by Azar et. al. [11] for the case that the server only has a multicast push channel and unicast. In [11] it is shown that if one wants to minimize the server bandwidth required to obtain a particular delay L for all the documents, then the multicast push channel should consist of those documents with probability of access greater than $1/(\lambda L)$, where λ is the arrival rate of document requests. By binary search, we can then find the minimum delay L obtainable by our server's fixed known bandwidth.

Indexing. The server also broadcasts an *index* of sorted URIs or URI digests which quickly allows the client to determine whether the requested resource is in the hot broadcast set [41, 51, 78, 6]. On the whole, the client determines the multicast group, downloads the appropriate portions of the index, and determines whether the resource is upcoming along the cyclic broadcast. If the request is not in the hot broadcast set, the client makes an explicit request to the server, and simultaneously starts to listen to the warm multicast group if one is available. If the page is cold, the requested resource is returned on the same connection.

Multicast Pull Scheduling. The *multicast pull scheduling* component resolves contention among client request for the use of the warm multicast channel and establishes the order in which pages are sent over that channel.

Multicast Push Scheduling. In multicast push, the server periodically broadcasts hot resources to the clients. The server chunks hot resources into nearly equal-size pages that fit into one datagram and then cyclically sends them on the specified multicast group along with index pages. The frequency and order in which pages are broadcast is determined by the *multicast push scheduling* component. Pages are then injected at a specified rate that is statically determined from measurements of network characteristics [8]. Alternatively, different connectivity can be accommodated with a variety of methods: the multicast can be replicated across multiple layers [16, 21, 81], it can be supported by router-assisted congestion control [61], or it can use application-level schemes in end-to-end multicast [23]. Client applications can recover from packet loss by listening to consecutive repetitions of the broadcast [8] or pages can be encoded redundantly with a variety of schemes that allow the message to be reconstructed [21].

Caching and consistency. Upon receipt of the desired pages, the client buffers them to reconstruct the original resource and can *cache* resources to satisfy future request [50, 57]. The set of hot pages is cyclically multicast, and so received pages are current in that they cannot be more than one cycle out-of-date. Furthermore, certain types of *consistency* semantics can be guaranteed by transmitting additional information along with the control pages [73, 66].

Transport Adaptation Layer. The TAL (Transport Adaptation Layer) renders middleware and applications independent of the particular choice of a multicast transport protocol.

3 Background and Historical Notes

3.1 Multicast Data Dissemination

An early example of scalable data dissemination is the teletext system, which uses the spare bandwidth allocated in the European television channels to implement multicast-push data dissemination. Television-based teletext systems have been deployed since the early 80's by most national television companies in Western Europe. Teletext has since attained nationwide diffusion and reaches most households. It provides a continuous information source and has deeply influenced the lifestyle of the countries where it is operational [35, 74].

In the context of data networks, early examples of multicast push include a high throughput multiprocessors database systems over high bandwidth networks [20] and a community information system over wireless [35]. Multicast push is a good fit for *infostations*, a critical component in a wireless information system where connectivity is spotty [43]. Early results and directions in the field are discussed in [32]. The DBIS-Toolkit [9] introduces a gateway for data delivery. The toolkit provides a set of components to adapt various data dissemination methods with each other. It builds an overlay network where different logical links used either multicast or unicast, push or pull, and synchronous or asynchronous communication.

3.2 Multicast

In *broadcast* communication, packets are delivered to all the hosts in a network. In *multicast*, packets are delivered to a specified set of hosts. In many types of networks, broadcasting is available natively at the physical or at the data link layer. For example, broadcasting is the basic communication mode in shared media, such as wireless and optical. In these networks, broadcast is the foundation for other communication methods (multicast and unicast), which are implemented by selectively discarding frames at the receiver site if that frame was not directed to that host. Since broadcasting is available natively, it is the natural methods to disseminate data on such media. Broadcast data dissemination can be used over wireless links, including satellites [3] and base stations schemes [43], and optical networks.

Broadcast does not scale to multi-hop networks and, consequently, is only used for special applications (e.g., DHCP) in these networks. The lack of a broadcast primitive complicates the implementation of multicast as well. In the context of the Internet, multicast is supported by the network layer [27, 38] but is seldom enabled for reasons including security, performance, scalability, difficulty of management and accounting [23]. As a result, multicast is more commonly supported at the application layer (*overlay multicast*) through protocols such as HyperCast [60], OverCast [44], Narada [23], or Yoid. Multicast protocols, whether based on IP-multicast or on overlays, differ in the type of functionality that they offer. For example, reliable multicast [37] ensures error recovery

and single-source multicast [38] allows for the added scalability of IP multicast. Additionally, a peer-to-peer system such as Chord [24] can support some multicast-like functionality.

In unicast communication, the sender typically regulates the rate at which it injects packets into the network so as to avoid overwhelming the receiver (*flow control*) or intermediate network nodes (*congestion control*) while simultaneously achieving maximum throughput [64]. In a multicast environment, flow and congestion control are significantly complicated by the heterogeneity of the receiver population. In particular, each receiver can have a receive buffer of different size and different available end-to-end bandwidth. A highly scalable solution to flow and congestion control is *layered multicast* [63]. In a layered multicast scheme, data is multicast simultaneously on L different channels (*layers*). In the most common scheme, the transmission rate r_l at layer l is $r_0 = r_1 = 1$ and $r_l = 2r_{l-1}$ for $1 < l < L$ (basically, the rates increase by a factor of 2 from one layer to the next). A receiver can subscribe to a prefix of layers $0, 1, \ldots, l$, thereby selecting to receive data at a rate which is at least $1/2$ of the bandwidth available from the source to the receiver.

4 Middleware Components

In this section, we discuss all the components of the middleware. We will describe known algorithms for the implementation of the components and their analytical and experimental evaluation. We will also discuss open research issues pertaining to each one of the components. Figure 1 gives a broad architectural overview of how the following components can fit together in a typical configuration of the middleware.

4.1 Transport Adaptation Layer

A first middleware component is the Transport Adaptation Layer (TAL). Its objective is to enable the middleware to interact with different types of multicast transport within a uniform interface. Different multicast protocols often present different API's and different capabilities. It is unlikely that a single multicast mechanism would be able to satisfy the requirements of all applications [37], and so the middleware must be able to interact with various underlying multicast transport protocols. As a result, the TAL allows us to write the middleware with a unique multicast API while retaining the flexibility as to the exact multicast transport. The TAL is a lowest common layer within the middleware (Figure 1) and provides transport-like functionality to all other communicating middleware modules. It possesses a server components (mostly used to stream pages out on a multicast channel) and a client component (mostly used to receive pages from such stream).

The Java Reliable Multicast (JRM) library [71] is an existing implementation that contains a TAL-like interface, the *Multicast Transport API (MTAPI)*. The MTAPI supports multiple underlying multicast protocols and allows for new protocols to be seamlessly added. It should be noted that the TAL's objective is not to restrict a given application to a set of specific multicast protocols. An additional functionality of TAL is to integrate various multicast packages so that clients supporting different multicast protocols can communicate with each other through the middleware.

An applications follows the following steps to employ the TAL. Before activating the middleware, the application will call the channel management API (e.g., in JRM) to:

- Choose the most appropriate multicast transport among those that are available.
- Interface with transport layer functions that resolve channel management issues, including the creation, configuration, destruction, and, possibly, address allocation, of multicast channels

upon requests.

- Interface with transport layer functions that support session management, including session advertising, discovery, join, and event notification.

The application can then invoke a security API (as in JRM) to establish data confidentiality and integrity as well as message, sender, and receiver authentication. At this stage, the application has a properly configured multicast channel, which will be passed to the data management middleware along with a description of the target data set. Additionally, if the application also desires a multicast pull channel for the warm pages, it creates and configures a second multicast channel through analogous invocations of the channel management and security API's and passes it to the middleware. Other middleware modules use the TAL (e.g., MTAPI) to:

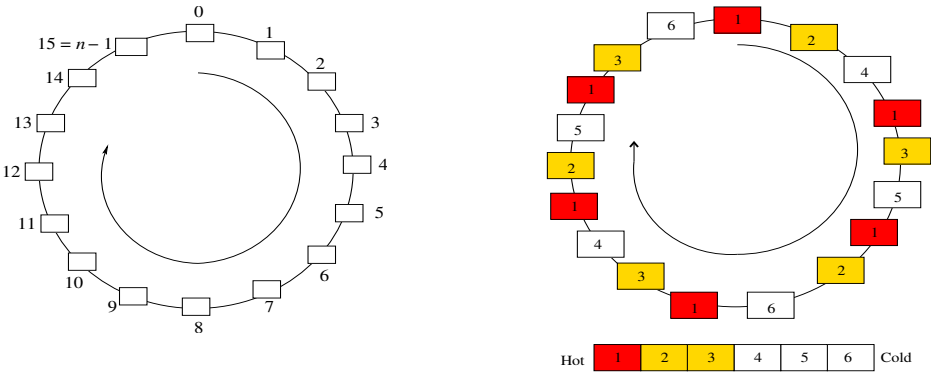
- Send and receive data from multicast groups.
- Obtain aggregate information from clients if the multicast transport layer supports cumulative feedback from the clients.

The TAL is a thin layer that does *not* implement features that are missing or are inappropriate for the underlying transport. The purpose of the TAL is to provide a common interface to existing protocols and not to replace features that are not implemented in the given protocols. For example, the TAL does not provide any additional security, but it simply interfaces with existing security modules in the underlying multicast layer.

While the transport adaptation layer aims at supporting diverse transport modules, the nature of the middleware and of the target applications that will run on it impose certain constraints on the types of transport layers that can be supported. The most important one is that data dissemination needs to scale to a very large receiver group. Consequently, the transport must avoid the problem of ACK/NACK implosion, through, for example, ACK aggregation [37] or NACK suppression [31, 36]. Alternatively, a reliable multicast transport can adopt open-loop reliability solutions such as cyclical broadcast [8] or error-correcting codes [21]; open-loop solution are an especially good fit for asymmetric communication environments such as satellites or heavily loaded servers. On the other hand, the middleware does not necessarily need a multicast transport that provides Quality-of-Service guarantees, that accommodates multiple interacting senders, that supports intermittent data flows, nor that provides secure delivery. Receivers can join the multicast at different start times, but will not receive copies of data sent before their start time.

4.2 Document Selection

As discussed in Section 2, the middleware operates according to a hybrid scheme whereby multicast push, multicast pull, and unicast are used depending on whether individual pages are hot, warm, or cold. The server must partition the set of pages into these three groups, and such operation is accomplished by the document selection module. The document selection is one of the primary entry points to the middleware at the server side: the application dynamically specifies the target document set to the document selection unit through an appropriate API and the unit will autonomously partition such data set for use of downstream middleware components (Figure 1). The document selection unit uses statistics of document popularity collected within the main control module, caches documents in main memory if possible, and requests to the application documents that are not cached. Document selection is a server-only module: the client must conform to the selection choices made by the server.



(a) Flat schedule

(b) Skewed schedule

Figure 2: The figure at left is an example of a flat broadcast program. Pages are numbered from 0 to $n-1$, and are cyclically transmitted by the server in that order. The figure at right is a skewed broadcast schedule that is obtained by multiplexing on the same physical channel the logical channels $\{1\}, \{2,3\}, \{4,5,6\}$.

The implementation of the document selection unit ultimately depends on the following consideration: pages should be classified as hot, warm, or cold so that overall middleware performance is improved [75]. Save this general consideration, however, the document selection unit can be implemented in various ways. Document selection can be accomplished by having client request all documents, including those in the multicast push broadcast [8]. The advantage of this solution is that the server obtains precise estimates of document popularity, at the price of forcing client requests even when such requests are not needed. As a result, client operations could be delayed and dissemination does not achieve maximal scalability. Another solution is to temporarily remove documents from the broadcast cycle and count the number of ensuing requests. Then, if the server receives a large number of requests, it assumes that the document is still hot and it is reinserted in the hot resource set [75]. In addition to these established methods, alternatives include collecting aggregate statistics from multicast feedback, as in single-source multicast, and randomly sampling the client population [17].

4.3 Multicast Push Scheduling

In multicast push, a server cyclically multicasts a set of hot pages. The objective of the multicast push scheduling component is to decide the frequency and order in which those pages are disseminated. The component obtains the multicast page set from the main control unit (so that no further decision as to the membership in that set is made downstream of the document selection component, Figure 1). Multicast push scheduling also interacts with the broadcast index component (Section 4.5). The multicast push scheduling components is present only at the server side.

The implementation of multicast push scheduling can proceed according to different algorithms. The simplest broadcast strategy is to adopt a *flat broadcast schedule*, whereby each page is transmitted once every n ticks. A sample flat schedule is depicted in figure 2(a). Non-flat schemes are desirable when some pages are more popular than others, in which case, the most popular pages should be devoted a larger fraction of available bandwidth. A simple way to differentiate pages is

through *frequency multiplexing*, where the data source partitions the data set across several physical channels according to their popularity [54, 69]. Differentiated treatment arises from aggregating a smaller amount of hotter data on one channel and a larger amount of cooler data on another channel. Since channel bandwidth is the same, the fewer hotter pages receive a proportionally larger amount of bandwidth than cooler pages. An alternative is *time-division multiplexing*, whereby pages are partitioned into a set of *logical channels*, and the logical channels alternate over the same physical channel [3]. The broadcast schedule is flat within a single logical channel, but hotter channels contain less pages or are scheduled for broadcast more frequently than cooler channels. Thus, hotter pages are transmitted more often than cooler ones. Figure 2(b) gives an example of a broadcast schedule that is the time-multiplexed combination of three logical channels, each containing a different number of pages. Time-division multiplexing is potentially more flexible than frequency multiplexing in that it allows for a finer bandwidth partition. In particular, a logical channel can contain only one page, which results in a fine per-page transmission schedule. When the broadcast is scheduled on a per-page basis, pages are broadcast on the same physical channel with frequency determined by their popularity. A family of scheduling algorithms for time-multiplexed broadcast assumes that the data source has estimates of the stationary probabilities with which clients need pages. Estimations of these probabilities would be collected by the document selection module as part of the classification of page popularity. The objective of scheduling is to minimize the expected waiting time to download a page. The square-root law asserts that page i should be scheduled with frequency proportional to $\sqrt{p_i}$ [12], where p_i is the stationary probability that i is requested by clients. The square-root law results in transmission frequencies that are in general not integral, and thus the law can be interpreted as a linear relaxation of an integer programming problem [12]. The problem of finding an optimal integral cyclic schedule is NP-hard if pages have a broadcast cost [12], but not MAX-SNP-hard [49], and it can be solved in polynomial time if the server broadcasts only two pages [14]. A simple and practical 2-approximation algorithm is expressed by the *MAD* (Mean Aggregate Delay) rule [76, 12]. The MAD algorithm maintains a value s_i associated with each page i. The quantity s_i is the number of broadcast ticks since the last time page i was broadcast. The MAD algorithm broadcasts a page i with the maximum value of $(s_i + 1)^2 p_i$. In particular, when all p_i's are equal, MAD generates a flat broadcast. MAD is a 2-approximation of the optimal schedule, and guarantees a cyclical schedule. Extensions include algorithms for the cases when broadcast pages have different sizes [48, 72] and when client objectives are described by polynomial utility functions [13].

The stationary access probabilities p_i do not express dependencies between data items. Consider the following elementary example. Page B is an embedded image in page A (and only in page A). Page A is not accessed very frequently, but when it is accessed, page B is almost certainly accessed as well. In this scenario, the stationary access probability p_B of page B is small, but the value of p_B is not fully expressive of the true access pattern to B. The problem of exploiting page access dependencies can be modeled as a graph optimization problem, for which we have devised an $\widetilde{O}(\sqrt{n})$-approximation algorithm [59]. We have also proposed a sequence of simpler heuristics that exploit page access dependencies. We measured the resulting client-perceived delay on multiple Web server traces, and observed a speed-up over previous methods ranging from 8% to 91% [58].

4.4 Multicast Pull Scheduling

In multicast pull, clients make explicit requests for resources and the server multicasts its responses. If several clients ask the same resource at approximately the same time, the server can aggregate those requests and multicast the corresponding resource only once. The multicast pull scheduling

component resolves contention among client request for the use of the warm multicast channel and establishes the order in which pages are sent over that channel. The multicast pull scheduling component operates only at the server. There are many reasonable objective functions to measure the performance of a server, but by far the mostly commonly studied measure is average user perceived latency, or equivalently average flow/response time, which measures how long the average request waits to be satisfied. In traditional unicast pull dissemination it is well known that the algorithm Shortest Remaining Processing Time (SRPT) optimizes average user perceived latency.

The most obvious reason that the situation is trickier for the server in multicast pull data dissemination, than for unicast pull data dissemination, since the server needs to balance the conflicting demands of servicing shorter files and of serving more popular files. To see that the situation can even be more insidious consider the case that all of the documents are of unit size. Then the obvious algorithm is Most Requests First (MRF) which always transmit the document with the most outstanding requests. In fact, one might even initially suspect that MRF is optimal in the case of unit sized documents. However, it was shown in [46] that MRF can perform arbitrarily badly compared to optimal. The basic idea of this lower bound is that it can be a bad idea to immediately broadcast the most popular document if more requests will arrive immediately after this broadcast. Thus one sees that the multicast pull server has to also be concerned about how to best aggregate requests over time.

The standard redress in situations where a limited algorithm can not produce optimal solutions is to seek algorithms that produce solutions with bounded relative error (i.e., competitive analysis). However, it was shown in [46, 30] that no algorithm can achieve even bounded relative error. As is commonly the case, the input distributions for which it is impossible to guarantee bounded relative error are inputs where the system load is near peak capacity. This makes intuitive sense as a server at near peak capacity has insufficient residual processing power to recover from even small mistakes in scheduling. In [45], we suggested that one should seek algorithms that are guaranteed to perform well unless the system is practically starved. For example, we showed in [45] that the algorithms Shortest Elapsed Time First, and the Multi-level Feedback Queue used by Unix and Windows NT, have bounded relative error if the server load is bounded an arbitrarily small amount away from peak capacity.

Our method to analyze scheduling problems has come to be known as *resource augmentation analysis*, and has been widely adopted as an analysis technique for a wide array of online and offline scheduling problems. In our context, an *s-speed c-approximation* online algorithm A has the property that for all inputs, the average user perceived latency of the schedule that A produces with a speed s server, denoted by A_s, is at most c times Opt_c, the optimal average user perceived latency for a speed 1 server. The notation and terminology are from [65]. Intuitively, Opt_1 is approximately equal to $\Theta(Opt_{1+\epsilon})$ unless the system load is near peak capacity. So one way to interpret an s-speed c-approximation algorithm is that is it is $O(c)$-competitive if $Opt_1 = \Theta(Opt_s)$, or alternatively, if the system load is at most $1/s$.

We have shown in [30] that the algorithm Equi-partition, that broadcasts each file at a rate proportional to the number of outstanding requests for the file, is an $O(1)$-speed $O(1)$-approximation algorithm, that is, the average user perceived latencies produce by Equi-partion have bounded relative error if the server load is below some constant threshold. This work also highlights the surprisingly close relationship between multicast pull scheduling and scheduling computational jobs, with varying speed-up curves, on a multiprocessor. Another way of viewing Equi-partition is that the bandwidth is distributed evenly between all requests. Hence, Equi-partition can be implement by servicing the requests in a round robin fashion.

Researchers, including us, have also considered the special case where the data items are all of

approximately the same size [7, 46]. This would be an appropriate job environment for a name server communicating IP address, or any server where all data items are small. It seems that the right algorithm in this setting is Longest Wait First (LWF), which was proposed in proposed in [29]. The algorithm LWF maintains a counter for each data item that is the sum over all unsatisfied requests for that page, of the elapsed time since that request. The algorithm LWF then always broadcasts the page with highest counter. LWF can be implemented in logarithmic time per broadcast using the data structure given in [47]. We show in [62] that LWF is an $O(1)$-speed $O(1)$-approximation algorithm.

An open research problem is to seek an algorithm that has bounded relative error when the load is arbitrarily close to the peak capacity of the server. That is, we seek an $(1+\epsilon)$-speed $O(1)$-competitive algorithms. Such an algorithm is called *almost fully scalable* [70].

Another interesting dimension to the multicast push scheduling is the exploitation of the semantics of data and applications to develop highly efficient, application specific algorithms. In Section 6, we briefly discuss two such algorithms developed in the context of decision-support applications.

4.5 Multicast Indexing

In multicast data dissemination, users monitor the multicast/broadcast channel and retrieve documents as they arrive on the channel. This kind of access is sequential, as it is in tape drives. On the other hand, the middleware combines one multicast push channel and one multicast pull channel, and so it must support effective tuning into multiple multicast channels. To achieve effective tuning, the client needs some form of directory information to be broadcasted along with data/documents, making the broadcast self-descriptive. This directory identifies the data items on the broadcast by some key value, or URL, and gives the time step of the actual broadcast. Further, such a directory not only facilitates effective search across multiple channels but it also supports an energy efficient way to access data. The power consumption is a key issue for both hand-held and mobile devices given their dependency on small batteries, but also for all other computer products given the negative effects of heat. Heat adversely effects the reliability of the digital circuits and increases costs for cooling especially in servers [77]. In order to access the desired data, a client has to be in *active mode*, waiting for the data to appear on the multicast. New architectures are capable of switching from *active mode* to *doze mode* which requires much less energy.

The objective of the multicast indexing component is to introduce a page directory in the multicast push channel for the purposes described above. The component is invoked after scheduling and feeds directly into the TAL to multicast the hot data set and the corresponding index. Multicast indexing has component both at the server and on the clients. In multicast push data dissemination, several implementation of multicast indexing have been proposed to encode a directory structure. These include incorporating hashing in broadcasts [42], using signature techniques [53] and broadcasting *index* information along with data [41, 25, 26, 78, 39]. Of these, broadcast indexing is the simplest and most effective in terms of space utilization. The efficiency of accessing data on a multicast can be characterized by two parameters.

- *Tuning Time:* The amount of time spent by a user in active mode (listening to channel) and
- *Access Time:* The total time that elapses from the moment a client requests data identified by ordering key, to the time the client reads that data on channel.

Ideally, we would like to reduce both tuning time and access time. However, it is generally not possible to simultaneously optimize both tuning time and access time. Optimizing the tuning time requires additional information to be broadcast. On the other hand, the best access time is

achieved when only data are broadcast and without any indexing. Clearly, this is the worst case for tuning time.

We have developed a new indexing scheme, called *Constant-size I-node Distributed Indexing* (CI) [6], that performs much better with respect to tuning time and access time for broadcast sizes in practical applications. This new scheme minimizes the amount of coding required for constructing an index in order to correctly locate the required data on the broadcast, thus decreasing the size of the index and consequently access time as well. Our detailed simulation results indicate that CI, which is a variant of the previously best performing *Distributed Indexing* (DI) [41, 78], outperforms it for broadcast sizes of 12,000 or fewer data items, reducing access time up to 25%, tuning time by 15% and saving energy up to 40%. Our experimental results on 1 to 5 channels also reveal that there is a tradeoff between the various existing indexing schemes in terms of tuning and access time and that the performance of different schemes is dependent on the size of the broadcast [5].

Besides optimizing existing schemes, one open research goal is to develop a mixed-adaptive indexing scheme that is essentially optimal over all broadcast sizes.

4.6 Data Consistency and Currency

As multicast-based data dissemination continues to evolve, more and more sophisticated client applications will require reading current and consistent data despite updates at the server. For this reason, several protocols have been recently proposed [73, 15, 67, 66, 52, 28] with the goal of achieving consistency and currency in broadcast environments beyond local cache consistency. The objective of the data consistency components are to implement such protocols for consistency and currency. All these protocols assume that the server is stateless and does not therefore maintain any client-specific control information. To get semantic and temporal related information, clients do not contact the server directly, instead concurrency control information, such as invalidation reports, is broadcast along with the data. This is in line with the multicast push paradigm to enhance the server scalability to millions of clients. When the scheduling algorithm selects a page to be multicast, there can be in general different versions of that page that can be disseminated. The two obvious choices are

- *Immediate-value broadcast:* The value that is placed on the broadcast channel at time t for an item x is the most recent value of x (that is the value of x produced by all transactions committed at the server by t).
- *Periodic-update broadcast:* Updates at the server are not reflected on the broadcast content immediately, but at the beginning of intervals called *broadcast currency intervals* or *bc-intervals* for short. In particular, the value of item x that the server places on the broadcast at time t is the value of x produced by all transactions committed at the server by the beginning of the current bc-interval. Note that this may not be the value of x at the server at the time x is placed in the broadcast medium if in the interim x has been updated by the server.

In the case of periodic-update broadcast, often the bc-interval is selected to coincide with the broadcast cycle, so that the value broadcast for each item during the cycle is the value of the item at the server at the beginning of the cycle. In this case, clients reading all their data, for example, components of a complex document, within a bc-interval are ensured to be both consistent and current with respect to the beginning of the broadcast cycle. However, different components that are read from different broadcast cycles might not be mutually consistent even if current, and hence when they are combined by the clients, the resulting document may be one that had never existed

in the server. The same problem exists also in the case of immediate-value broadcast – consider immediate-value broadcast as a periodic-update broadcast with a bc-interval of zero duration.

Our investigation is directed towards the development of a general framework for correctness in broadcast-based data dissemination environments [68]. We have introduced the notion of the *currency interval* of an item in the readset of a transaction as the time interval during which the value of the item is valid. Based on the currency intervals of the items in the readset, we developed the notion of temporal spread of the readset and two notions of currency (snapshot and oldest-value) through which we characterize the temporal coherency of a transaction. Further, in order to better combine currency and consistency notions, we have developed a taxonomy of the existing consistency guarantees [18, 33, 82] and show that there are three versions for each definition of consistency. The first (Ci) is the strongest one and requires serializability of *all* read-only client transactions with server transactions. This means that there is a global serialization order including all read-only client transactions and (a subset of) server transactions. The second version (Ci-S) requires serializability of some *subset* of client read-only transactions with the server transactions. This subset may for example consist of all transactions at a given client site. The last version (Ci-I) requires serializability of each read-only client transaction *individually* with the server transactions.

An open research problem is to investigate efficient ways to disseminate any control information. For example, consistency information can utilize the broadcast indexing structures. Another open research issue is to expand our theoretical framework to include the case of a cache being maintained at the clients. The goal is to develop a model that will provide the necessary tools for arguing about the temporal and semantic coherency provided by the various protocols to client transactions. In addition, it will provide the basis for new protocols to be advanced.

4.7 Client Cache

Clients have the option of caching the multicast pages locally. As a result, clients can find requested pages either in their local cache or by listening to the server multicast. The client caching module provides buffer space and a cache management functionality to the client. The module is the main entry point of the client application into the middleware: a page is always requested to the cache module first, and it passed on to the remaining client middleware components only in the case of a cache miss. The implementation of the client cache module takes into account the two primary factors of *consistency* and *replacement*. If the client caches a page, that page could contain stale values or its usage can lead to an inconsistent view of the original data set. Thus, appropriate strategies are needed for currency and consistency, which were discussed in Section 4.6. Moreover, if the client has a cache of limited size, it must perform *page placement and replacement*, that is, it must dynamically decide which pages are stored in the cache and which pages should be removed from the cache. Traditional policies for cache replacement include for example LRU (Least-Recently Used) and similar policies originally developed in the context of virtual memory systems [79]. The gist of LRU is that past accesses should predict future accesses, and so it should incur few page misses. In multicast push environments, caching aims at reducing the time spent waiting during the broadcast. By contrast, minimizing (say) the number of page faults in isolation might not bring any performance improvement. Consequently, a policy such as LRU can lead to suboptimal performance in multicast push environments. In the context of Web caching, a number of algorithms, such as Landlord/Greedy-Dual-Size [22, 83], take into account both the number of cache misses and the time spent to download a Web object. Such policies can be suboptimal in a multicast push environment for two reasons. First, Web caching policies are based on average estimates of resource download times, whereas in the multicast environment, the download time varies periodically depending on

the server schedule. Second, *prefetching* can sometimes be executed without clients waiting on the broadcast [4], and can thus be used to complement a pure replacement strategy.

If stationary access probabilities exist and are known, several policies strike a balance between page popularity and access times [3, 4, 80]. For example, the PT algorithm maintains two values for each broadcast page i: p_i, the probability that page i will be requested, and t_i, the waiting time needed to load i once the current request has been satisfied. Then, PT keeps in the cache the set of pages with the largest values of $p_i t_i$ [4]. In certain applications, such as Web resource dissemination, hot pages have nearly the same stationary access probability [10] or such probabilities do not fully capture the underlying access process [56], so that it is desirable to have page replacement and prefetching algorithms that work with no probabilistic assumptions. Algorithms of this type are said to be *online algorithms* and are almost always evaluated through *competitive analysis* (Section 4.4, [19]). We have devised an algorithm (the *Gray algorithm*) that is online and is provably optimal in terms of competitive ratios [50]. We have subjected the Gray algorithm to extensive empirical investigation on both synthetic and Web traces, where it outperformed previous online strategies [57].

5 Building Block Integration

5.1 Integration

The interaction of the middleware building blocks should be carefully tuned to achieve optimal performance. Multicast data management components have been often designed, analyzed, and empirically evaluated in isolation. Important exceptions include [57], which addresses the interaction of caching and scheduling, and [40], which focuses on wireless environments. The integration of building blocks is still riddled with many additional open research issues. First, the scheduling of the multicast push determines the frequency and order in which data items are broadcast, and so it affects the appropriateness of an indexing scheme. Analogously, cache replacement strategies interact with consistency policies at the client side. Another issue arises from the interaction of dynamic document selection with scheduling and indexing.

Another type of interaction exists between the middleware and the underlying transport. For example, certain scheduling, indexing, and caching algorithms assume that the broadcast data is reliably delivered to all recipients in the order in which it is broadcast. Packet drops can potentially cause performance degradation, even when open-loop reliability solutions are adopted. Certain protocols, such as IP multicast, are intrinsically unreliable. Furthermore, packet losses can occur as a side effect of congestion control algorithms: intermediate components, such as router in router-assisted congestion control [61] or intermediate hosts in end-to-end multicast and in multiple-group sender-controlled congestion control [37] deliberately drop packets to avoid overrunning the downlink. Another issue arises when a layered multicast scheme is chosen (Section 3.2). Certain scheduling and indexing algorithms are optimized for the case when all receiving clients obtain the multicast pages in the same order. However, layering can change the order in which clients receive the multicast pages depending on how many layers have been subscribed to by clients. For example, a client that has subscribed several layers receives broadcast pages in a shuffled sequence as compared to a client that subscribed to one multicast layer [16]. However, indexing relies on the existence of a certain fixed ordering of the broadcast cycle that is common to all clients. Thus, layering can lead to suboptimal performance of indexing and scheduling. Analogously, scheduling establishes an ordering of the broadcast pages which can be scrambled by a subsequent layered scheme. The interaction between push scheduling and multicast congestion control is discussed

next in more details.

5.2 Scheduling for Layered Multicast

Layered multicast leads to a different version of the multicast push scheduling problem where the contents are scheduled on multiple channels and the bandwidth on channel l is given by the rates r_l. Unlike previous multichannel models (e.g., [12]), all contents are multicast on layer 0 (to guarantee that all receivers can eventually get all the contents) and receivers do not necessarily listen to all channels. Because of these added restrictions, it can easily be shown that the previously known lower-bounds on the average waiting time hold also for the layered multicast setting. Furthermore, previous algorithms for the multichannel problem can be made to run within one layer, which, in conjunction with the previous lower bound, proves that every c-approximation algorithm for the multichannel problem becomes a $2c$-approximation algorithm for the layered multicast problem. In particular, the algorithm of [49] gives a $(2+\epsilon)$-approximation algorithm that runs in polynomial time for any fixed $\epsilon > 0$ [55]. The main open questions in this area are to find better approximation algorithms and to empirically verify their performance.

6 Application: Real-Time Outbreak and Disease Surveillance

Section 2 describes the use of a data dissemination middleware for scalable Web contents delivery. This section discusses an additional application: the RODS (Real-Time Outbreak and Disease Surveillance) system. RODS is a healthcare alert system developed by the Center for Biomedical Informatics at the University of Pittsburgh [1]. The RODS system has been deployed since 1999 in Western Pennsylvania and since December 2001 in Utah for the Winter Olympic Games [34]. The core of RODS is the health-system-resident component (HSRC) whose function is data merging, data regularization, privacy protection, and communication with the regional system. Typically, users or applications request consolidated and summarized information as in OLAP (on-line analytical processing) business applications. For example, queries often involve joins over several large tables to perform a statistical analysis, e.g., computing daily percentage of patients with a particular prodrome in a region for one month period. Also, currently RODS displays spatial-temporal plots of patients presenting with seven key prodromes through a Web interface.

RODS can use a data dissemination middleware to support the collection and monitoring of the large volume of data needed for the assessment of disease outbreaks, as well as the dissemination of critical information to a large number of health officials when outbreaks of diseases are detected.

We have implemented a prototype similar to RODS that utilizes the middleware to disseminate data for decision making especially during emergencies and periods of crisis. In this prototype, the schema for the database was adapted from the RODS Laboratory. The database, implemented on one of the department's Oracle servers, was populated with randomly distributed data that would mimic the type of data that is collected in the real system. In particular, cases of disease and prodromes populate the database. In the prototype, the data was restricted to each of the sixty-seven counties in Pennsylvania.

The server-side modules were written in Java 2 using the JDBC to Oracle conduits. The application server queries the database for the counts of each distinct disease and prodrome grouped by different attributes, for example by county, and passes them to the middleware. The middleware then transmits these counts in XML messages on the appropriate multicast channel.

The client-side modules were written in Java 2 as well. The client application accepts user input via a graphical user interface that displays the map of Pennsylvania broken down by county. When

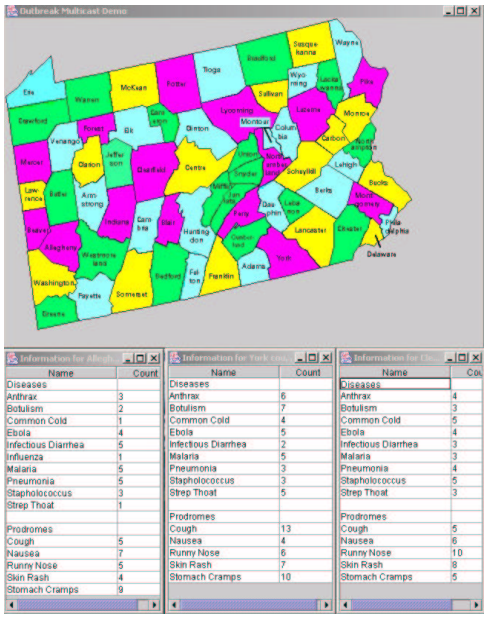


Figure 3: A snapshot of the emulated RODS systems.

a user clicks on a county, the counts of disease and prodromes in that county are requested via the middleware (see Figure 3). If the requested data is not locally cached, then it is fetched from the multicast channel. When these counts are returned by the middleware, they are displayed on a separate window in a new table.

In the special context of RODS, we have investigated multicast push scheduling schemes that exploit the semantics of the OLAP data. More specifically, every requested document is a summary table and summary tables have the interesting property that one summary table can be derived from one or more other summary tables. This means that a table requested by a client may *subsume* the table requested by another client. We proposed two new, heuristic scheduling algorithms that use this property to both maximize the aggregated data sharing between clients and reduce the broadcast length compared to the already existing techniques. The first algorithm, called *Summary Tables On-Demand Broadcast Scheduler* STOBS, is based on the *RxW* algorithm [2] and the second one, called *Subsumption-Based Scheduler* (SBS), is based on the *Longest Total Stretch First* (LTSF) algorithm [2]. Further, they differ on the used criterion for aggregate requests. Otherwise, both STOBS and SBS are non-preemptive and consider the varying sizes of the summary tables. The effectiveness of the algorithms with respect to access time and fairness were evaluated using simulation. Both SBS and STOBS are applicable in the context of any OLAP environment both wired and wireless.

7 Conclusions

Multicast is a promising method to solve the scalability problem in the Internet. In this chapter, we have discussed a middleware that unifies and extends support for data management in multicast data dissemination. The chapter describes the overall objectives and architecture of the middleware, the function and interaction of its components, and the implementation of each component. Particular emphasis has gone into improving the performance of the middleware through the adoption of state-of-the-art algorithms, and research issues in this area have been surveyed.

The chapter has also described two applications of the middleware that ensure the scalability of bulk data dissemination. The first application is a *scalable Web server*, which exploits the middleware hybrid dissemination scheme to improve the performance of a Web hot spot. The second application is the *Real-Time Outbreak and Disease Surveillance* (RODS) system. RODS is a scalable healthcare alert system that disseminates information about prodromes and diseases over a wide geographical area. It can be used to scalable and efficiently alert health operator of disease outbreaks so that appropriate responses can be pursued in real-time.

References

[1] http://www.health.pitt.edu/rods.

[2] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proceedings of Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 1998.

[3] Swarup Acharya, Rafael Alonso, Michael Franklin, and Stanley Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *Proceedings of the 1995 ACM SIGMOD Conference International Conference on Management of Data*, pages 199–210, 1995.

[4] Swarup Acharya, Michael Franklin, and Stanley Zdonik. Prefetching from a broadcast disk. In *Proceedings of the International Conference on Data Engineering*, 1996.

[5] R. Agrawal and P. K. Chrysanthis. Accessing broadcast data on multiple channels in mobile environments more efficiently. Computer Science Technical Report TR-01-05, University of Pittsburgh, February 2001.

[6] R. Agrawal and P. K. Chrysanthis. Efficient data dissemination to mobile clients in e-commerce applications. In *Proceedings of the Third IEEE Int'l Workshop on Electronic Commerce and Web-based Information Systems*, June 2001.

[7] Demet Aksoy and Michael Franklin. RxW: A scheduling approach for large-scale on-demand data broadcast. In *Proceedings of the Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1998)*, 1998.

[8] K. C. Almeroth, M. H. Ammar, and Z. Fei. Scalable delivery of Web pages using cyclic best-effort (UDP) multicast. In *Proceedings of the Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1998)*, 1998.

[9] Mehmet Altinel, Demet Aksoy, Thomas Baby, Michael J. Franklin, William Shapiro, and Stanley B. Zdonik. Dbis-toolkit: Adaptable middleware for large scale data delivery. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 544–546. ACM Press, 1999.

[10] Martin Arlitt and Tai Jin. Workload characterization of the 1998 World Cup Web site. Technical Report HPL-1999-35R1, HP Laboratories, September 1999.

[11] Yossi Azar, Meir Feder, Eyal Lubetzky, Doron Rajwan, and Nadav Shulman. The multicast bandwidth advantage in serving a web site. In *Networked Group Communication*, pages 88–99, 2001.

[12] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. In *Proceedings of the Ninth ACM-SIAM Symposium on Discrete Algorithms*, pages 11–20, 1998.

[13] Amotz Bar-Noy, Boaz Patt-Shamir, and Igor Ziper. Broadcast disks with polynomial cost functions. In *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, 2000.

[14] Amotz Bar-Noy and Yaron Shilo. Optimal broadcasting of two files over an asymmetric channel. *Journal of Parallel and Distributed Computing*, 60(4):474–493, April 2000.

[15] D. Barbará. Certification reports: Supporting transactions in wireless systems. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, June 1997.

[16] S. Battacharyya, J. F. Kurose, D. Towsley, and R. Nagarajan. Efficient rate controlled bulk data transfer using multiple multicast groups. In *Proceedings of the Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1998)*, 1998.

[17] Jonathan Beaver, Kirk Pruhs, Panos Chrysanthis, and Vincenzo Liberatore. The Multicast Pull Advantage. Computer science tehnical report, University of Pittsburgh, September 2003.

[18] P. M. Bober and M. J. Carey. Multiversion query locking. In *Proceedings of the 1992 SIGMOD Conference*, pages 497–510, May 1992.

[19] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, New York, 1998.

[20] T. G. Bowen, G. Gopal, G. Herman, T. Hickey, K. C. Lee, W. H. Mansfield, J. Raitz, and A. Weinrib. The Datacycle architecture. *Communications of the ACM*, 35(12):71–81, December 1992.

[21] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. In *Proc. Sigcomm*, 1998.

[22] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 193–206, 1997.

[23] Yang-hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proceedings ACM SIGMETRICS '2000*, pages 1–12, 2000.

[24] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[25] A. Datta, A. Celik, J. Kim, D. VanderMeer, and V. Kumar. Adaptive Broadcast Protocols to Support Efficient and Energy Conserving Retrieval from Databases in Mobile Computing Environments. In *Proceedings of the IEEE Int'l Conference on Data Engineering*, pages 124–133, March 1997.

[26] A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar. Broadcast protocols to support efficient retrieval from databases by mobile users. *ACM TODS*, 24(1), 1999.

[27] S. Deering. Multicast routing in internetworks and extended lans. In *Proc. Sigcomm*, pages 55–64, 1988.

[28] Pavan Deolasee, Amol Katkar, Ankur Panchbudhe, Krithi Ramamritham, and Prashant Shenoy. Dissemination of dynamic data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 599. ACM Press, 2001.

[29] H. Dykeman, M. Ammar, and J. Wong. Scheduling algorithms for videotext under broadcast delivery. In *Proceedings of the IEEE International Conference on Communications*, 1986.

[30] Jeff Edmonds and Kirk Pruhs. Multicast pull scheduling: when fairness is fine. *Algorithmica*, 36:315–330, 2003.

[31] S. Floyd, V. Jacobson, and S. McCanne. A reliable multicast framework for light-weight sessions and application level framing. In *Proc ACM SIGCOMM*, pages 342–356, 1995.

[32] Michael Franklin and Stanley Zdonik. A framework for scalable dissemination-based systems. In *Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications*, pages 94–105, 1997.

[33] H. Garcia-Molina and G. Wiederhold. Read-Only Transactions in a Distributed Database. *ACM TODS*, 7(2):209–234, 1982.

[34] Per H. Gesteland, Reed M. Gardner, Fu-Chiang Tsui, Jeremy U. Espino, Robert T. Rolfs, Brent C. James, Wendy W. Chapman, Andrew W. Moore, and Michael M. Wagner. Automated syndromic surveillance for the 2002 winter olympics. August 2003.

[35] David K. Gifford. Polychannel systems for mass digital communications. *Communications of the ACM*, 33(2):141–151, February 1990.

[36] M. Handley and J. Crowcroft. Network text editor (nte) a scalable shared text editor for mbone. In *Proc ACM SIGCOMM*, pages 197–208, 1997.

[37] M. Handley, S. Floyd, B. Whetten, R. Kermode, L. Vicisano, and M. Luby. The reliable multicast design space for bulk data transfer. RFC 2887, 2000.

[38] Hugh W. Holbrook and David R. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proc. Sigcomm*, 1999.

[39] Q. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *Proceedings of the 16th International Conference on Data Engineering*, pages 157–166, February 2000.

[40] Qinglong Hu, Wang-Chien Lee, and Dik Lun Lee. Performance evaluation of a wireless hierarchical data dissemination system. In *Proc. MobiCom*, 1999.

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

[42] T. Imielinski, S. Viswanathan, and B.R. Badrinath. Power efficient filtering of data on air. In *Proc. of the Int'l Conference on Extending Database Technology*, 1994.

[43] Tomasz Imielinski and B. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, 37(10):18–28, October 1994.

[44] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Fran s Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *OSDI*, pages 197–212, 2000.

[45] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[46] B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai. Scheduling broadcasts in wireless networks. In *European Symposium on Algorithms (ESA)*, 2000.

[47] Haim Kaplan, Robert Tarjan, and Kostas Tsiotsiouliklis. Faster kinetic heaps and their use in broadcast scheduling. In *Proceedings of the ACM/SIAM Symposium on Discrete Algorithms*, 2001.

[48] Claire Kenyon and Nicolas Schabanel. The data broadcast problem with non-uniform transmission times. In *Proceedings of the Tenth ACM-SIAM Symposium on Discrete Algorithms*, pages 547–556, 1999.

[49] Claire Kenyon, Nicolas Schabanel, and Neal Young. Polynomial-time approximation scheme for data broadcast. In *Proceedings of the Thirtisecond ACM Symposium on the Theory of Computing*, 2000.

[50] Sanjeev Khanna and Vincenzo Liberatore. On broadcast disk paging. *SIAM Journal on Computing*, 29(5):1683–1702, 2000.

[51] Sanjeev Khanna and Shiyu Zhou. On indexed data broadcast. In *Proceedings of the Thirtieth ACM Symposium on the Theory of Computing*, pages 463–472, 1998.

[52] V. C. S. Lee, S. H. Son, and K. Lam. On the performance of transaction processing in broadcast environments. In *Proceedings of the Int'l Conference on Mobile Data Access (MDA'99)*, January 1999.

[53] W.C. Lee and D. L. Lee. Signature caching techniques for information filtering in mobile en vironments. *Wireless Networks*, pages 57–67, 1999.

[54] H.V. Leong and A. Si. Data broadcasting strategies over multiple unreliable wireless chan nels. In *Proceedings of the ACM Int'l Conference on Information and Knowledge Management*, 1995.

[55] Wei Li, Wenhui Zhang, and Vincenzo Liberatore. Dissemination scheduling in layered multicast environments. (In preparation).

[56] Vincenzo Liberatore. Empirical investigation of the Markov reference model. In *Proceedings of the Tenth ACM-SIAM Symposium on Discrete Algorithms*, pages 653–662, 1999.

[57] Vincenzo Liberatore. Caching and scheduling for broadcast disk systems. In *Proceedings of the 2nd Workshop on Algorithm Engineering and Experiments (ALENEX 00)*, pages 15–28, 2000.

[58] Vincenzo Liberatore. Broadcast scheduling for set requests. In *DIMACS Workshop on Resource Management and Scheduling in Next Generation Networks*, 2001.

[59] Vincenzo Liberatore. Circular arrangements. In *29-th International Colloquium on Automata, Languages, and Programming (ICALP), LNCS 2380*, pages 1054–1065, 2002.

[60] J. Liebeherr and B. S. Sethi. A scalable control topology for multicast communications. In *Proc. IEEE Infocom*, 1998.

[61] M. Luby, L. Vicisano, and T. Speakman. Heterogeneous multicast congestion control based on router packet filtering. In *RMT working group*, 1999.

[62] A maiden analysis of Longest Wait First. Scheduling in the dark. In *ACM/SIAM Symposium on Discrete Algorithms*, 2003.

[63] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proc. ACM SIGCOMM*, pages 117–130, 1996.

[64] Larry L. Peterson and Bruce S. Davie. *Computer Networks.* Morgan Kaufmann, 2000.

[65] Cynthia Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the ACM Symposium on Theory of Computing*, 1997.

[66] E. Pitoura and P. K. Chrysanthis. Exploiting versions for handling updates in broadcast disks. In *Proceedings of the 25th Int'l Conference on Very Large Data Bases*, pages 114–125, September 1999.

[67] E. Pitoura and P. K. Chrysanthis. Scalable processing of read-only transactions in broadcast push. In *Proceedings of the 19th IEEE Int'l Conference on Distributed Computing Systems*, pages 432–441, June 1999.

[68] E. Pitoura, P. K. Chrysanthis, and K. Ramamritham. Characterizing the temporal and semantic coherency of broadcast-based data dissemination. In *Proceedings of the International Conference on Database Theory*, pages 410–424, January 2003.

[69] K. Prabhakara, K. A. Hua, and J. Oh. Multi-level multi-channel air cache design for broadcasting in a mobile environment. In *Proceedings of the IEEE Int'l Conference on Data Engineering*, pages 167–176, March 2000.

[70] Kirk Pruhs, Eric Torng, and Jiri Sgall. Online scheduling. 2003.

[71] Phil Rosenzweig, Miriam Kadansky, and Steve Hanna. The Java reliable multicast service: A reliable multicast library. Technical Report SMLI TR-98-68, Sun Microsystems, 1998.

[72] Nicolas Schabanel. The data broadcast problem with preemption. In *LNCS 1770 Proc. of the 17th International Symposium on Theoretical Aspects of Computer Science (STACS 2000)*, pages 181–192, 2000.

[73] Jayavel Shanmugasundaram, Arvind Nithrakashyap, Rajendran Sivasankaran, and Krithi Ramamritham. Efficient concurrency control for broadcast environments. In *ACM SIGMOD International Conference on Management of Data*, 1999.

[74] Efrem Sigel. *Videotext: The Coming Revolution in Home/Office Information Retrieval.* Knowledge Industry Publications, White Plains, 1980.

[75] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *Proceedings 23rd International Conference on Very Large DataBases*, pages 326–335, 1997.

[76] C. J. Su and L. Tassiulas. Broadcast scheduling for information distribution. In *Proceedings of the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1997)*, 1997.

[77] Mudge T. Power: A first class design constraint. *Computer*, 34(4):52–57, April 2001.

[78] S. Viswanathan T. Imielinski and B.R. Badrinath. Data on air: Organization and access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):353–372, 1997.

[79] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, Englewood Cliffs, NJ, 1992.

[80] L. Tassiulas and C. J. Su. Optimal memory management strategies for a mobile user in a broadcast data delivery system. *IEEE Journal on Selected Areas in Communications*, 1997.

[81] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proceedings of the Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1998)*, 1998.

[82] W. E. Weihl. Distributed Version Management for Read-Only Actions. *ACM Transactions on Software Engineering*, 13(1):56–64, 1987.

[83] Neal Young. On-line file caching. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–91, 1998.