Middleware Support for Multicast-based Data Dissemination: A Working Reality[*]

Panos K. Chrysanthis[†] Vincenzo Liberatore[‡] Kirk Pruhs[§]

Abstract

Multicasting is an effective method to guarantee scalability of data transfer. Multicast applications range from the relief of Internet hot spots to healthcare alert systems. Much research has focused on isolated data management issues that arise in a multicast environment, including our previous work on caching, scheduling, indexing, hybrid schemes, and consistency maintenance. This paper discusses the integration of these research contributions and the transition to a working software distribution that provides the middleware support of a data management layer to applications. Our middleware is flexible, can be shared across applications, and operates on top of existing and upcoming implementations of multicast protocols. The middleware benefits distributed applications with a uniform, efficient, scalable, and state-of-the-art support for critical data management functionality.

1 Introduction

In a multicast environment, a single source sends data items, which are then replicated within the network infrastructure to reach a large client population. Therefore, multicast is an effective method to guarantee scalability. We believe that a fundamental question in middleware technology is how to support multicast environments. In this paper, we describe a unified middleware platform for multicast-based data dissemination, report on the current implementation, discuss our preliminary evaluation of this technology, and describe the associated research issues.

A major objective is to transparently provide applications with data management services such as caching, scheduling, and consistency maintenance. In turn, those services are enabled by an underlying multicast transport. The middleware frees application developers from the details of the underlying multicast transport and from the need of implement in each application a common set of data management algorithms. Furthermore, the middleware unifies and extends state-of-the-art data management methods and algorithms into one software distribution. Its flexible and extensible architecture is built from individual components that can be selected or replaced depending on the underlying multicast transport or on the application needs. As a result, established and novel techniques from data management will be more generally available to developers of scalable applications in multicast environments.

[*]This work has been supported in part under NSF grants ANI-0123929 and ANI-0123705.

[†]Dept. of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: `panos@cs.pitt.edu`

[‡]Electrical Engineering and Computer Science Department, Case Western Reserve University, 10900 Euclid Av., Cleveland, Ohio 44106. E-mail: `vxl11@po.cwru.edu`. URL: `http://vorlon.cwru.edu/~vxl11/`.

[§]Dept. of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: `kirk@cs.pitt.edu`

1.1 Multicasting

Broadcasting and multicasting are the natural methods to propagate information in media such as shared Ethernet [43], wireless links, including satellites [43] and short-range wireless [43], and optical networks, ranging from cable modems to high-throughput database systems [11]. In all those media, broadcasting is the primary mode of operation for the physical layer [43]. Broadcasting is used by several companies, including Hughes Networks and PanAmSat, to perform content delivery and to support Web browsing with satellites for home-based dishes or for access providers.

In wired networks, multicasting is an effective method to guarantee scalability of bulk data transfer. Multicasting has applications ranging from the support of Content Delivery Networks to the relief of Internet hot spots, such as during the last national elections [47]. Millions of clients attempted to access news sites during the evening of Election Day, overwhelming these servers, and resulting in long delays to retrieve documents. While the servers were contacted by millions of clients, most clients were interested in nearly the same data. In this scenario, a single server could have served all such requests by multicasting the hot data items to the interested clients. A substantial amount of research has been devoted to the design and implementation of multicast protocols in wired IP networks, including IP multicast [17, 26], reliable multicast [24, 58], and end-to-end multicast [13]. Those approaches will almost surely result in the diffusion of multicast in the Internet, and, in the case of application level solutions, they can be efficiently implemented regardless of the willingness and support of service providers or of the backbone infrastructure.

1.2 Data Management Issues in Multicasting

Multicast communication raises many data management issues and problems that either do not arise in unicast communication, or that obviously require different solutions than the standard methods used in unicast settings. Some of the issues are:

Document selection. What is the appropriate dissemination method for each document?

Scheduling. How frequently and in which order should documents be multicast?

Consistency. How should the system support currency and consistency for updated contents?

Cache Replacement. How to best manage client-side caches?

Indexing. How to best use indexes to reduce client waiting and tuning time?

1.3 Middleware for Broadcast Data Dissemination

The outline of our architecture is shown in Figure 1 (the transport layer is any one of the protocols that is available independently of the middleware, and the objective of the transport adaptation layer is to enable the middleware to interact with different types of multicast transport within a uniform interface). The approach

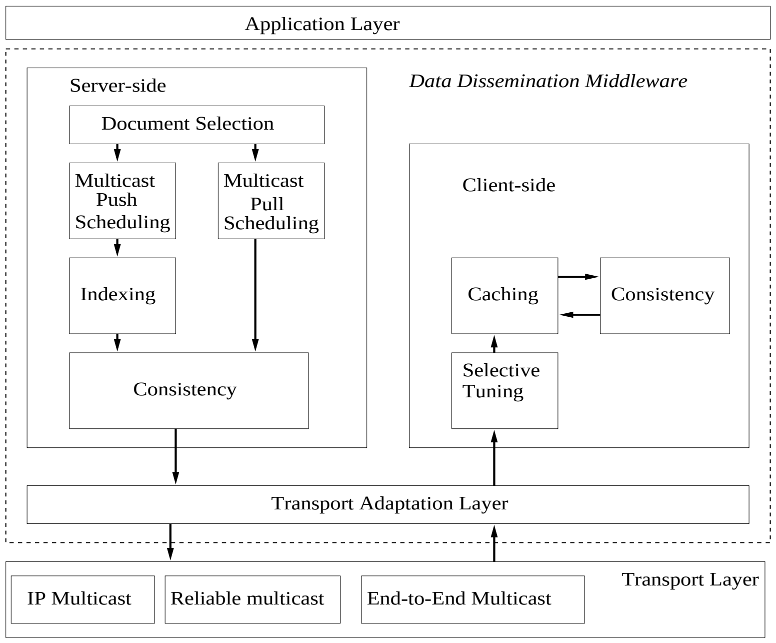

Figure 1: Our middleware data dissemination architecture, and its relationship with the application and transport layers.

has the benefit of unifying several algorithms and techniques from data management, such as caching or scheduling into one software distribution. As a result, established and new methods from data management will be more generally available to application developers. An integrated approach has highlighted gaps in the state of the art and led to new research problems and solutions. In particular, the middleware has exposed the performance and functional trade-off between existing data management algorithms and existing or proposed multicast protocols.

The two motivating example applications of our middleware are a scalable Web server and a healthcare alert system. In the next section, we elaborate on them in an integrated fashion. In Section 3, we discuss the middleware building blocks. In Section 4, we discuss the transport adaptation layer of our architecture. In Section 5, we outline multicast push scheduling for layered environments.

2 Motivating Application: Healthcare Alert System

In this section, we give a complete example of a scalable application that can exploit the multicast middleware. It is a healthcare alert system, called RODS[1] (Real-time Outbreak and Disease Surveillance) that has been developed by the Center for Biomedical Informatics at the University of Pittsburgh [20, 56].

The RODS system is a public health surveillance system deployed since 1999 in Western Pennsylvania and since December 2001 in Utah for the Winter Olympic Games. The core of RODS is the health-system-resident component (HSRC) whose function is data merging, data regularization, privacy protection, and

[1]RODS URL: www.health.pitt.edu/rods.

communication with the regional system. HSRC receives HL-7 ADT (admission, discharge and transfer) messages from over 45 hospitals and clinics within Utah State and Western Pennsylvania, operating under a trusted broken arrangement. These large data volume of raw data (about ten thousands records per day) are stored in a database and from there are disseminated for further analysis.

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 web interface.

RODS can use the multicast middleware to support the collection and monitoring of data needed for the assessment of disease outbreaks as well as the dissemination of critical information to a large number of health officials and other authorized personnel when outbreaks of diseases are detected[2]. Interestingly, RODS has the functionality of a Subscription/Publisher server and exhibits all the requirements of a highly scalable Web server as described in the work of Almeroth *et al.* [5]. The objective of the Web server application is to scale to a large user (client) population, and scalability will be accomplished by using the multicast-based middleware.

In the middleware, the server can disseminate data by choosing any combination of the following three schemes: *multicast push*, *multicast pull*, and *unicast 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. In multicast pull, the clients make explicit requests for resources, and the server broadcasts the responses to all members of the multicast group. If multiple clients request the same resource at approximately the same time, the server may aggregate these requests, and only broadcast the resource once. 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 [5]. Traditional unicast pull is reserved for cold documents. 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 multi-tier dissemination protocol.

In the RODS system as well as in a general Web server application, the *document selection* unit periodically gather statistics on document popularity (in other applications, the notion of Web document is

[2]In this paper, we are not elaborating on any security aspects of the system.

replaced by the concept of an application data unit (ADU) [24]). Furthermore, in the RODS system, the document selection can be influenced by epidemiological data. For example, during a winter period when Influenza outbreaks typically occur, prodrome daily data on Viral, Respiratory, Diarrhea, Rash, Encephalitis registrations are useful for detection of Influenza outbreaks. 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 explicit http request followed by redirection to the appropriate multicast address, hashing the URI to a multicast group [5], using a well-known multicast address paired to the IP address of the origin server in single-source multicast [26], or application-level discovery in the case of end-to-end multicast. 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 [28, 35, 54, 3]. 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 has the option of leaving the multicast group (although is not forced to do so), 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. If the page is warm, the clients waits on the warm multicast group until the requested resource is transmitted [5]. 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 [18, 1, 51].

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 [5]. Alternatively, different connectivity can be accommodated with a variety of methods: the multicast can be replicated across multiple layers [9, 12, 55], it can be supported by router-assisted congestion control [40], or it can use application-level schemes in end-to-end multicast [13]. Client applications can recover from packet loss by listening to consecutive repetitions of the broadcast [5] or pages can be encoded redundantly with a variety of schemes that allow the message to be reconstructed [12]. Upon receipt of the desired pages, the client can buffer them to reconstruct the original resource and can *cache* resources to satisfy future request [34, 39]. 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 [49, 44].

3 Multicast Middleware Building Blocks

3.1 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. Multicast pull is appropriate for "warm" documents that many, but not most, clients are interested in. 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 although current web servers use FIFO out of fear of starving jobs. This may perhaps change as it was recently shown in [25, 6] that all jobs, regardless of length, should prefer SRPT to FIFO if the distribution of lengths is heavy tailed, as is the case when job length is proportional to Web resource size [14].

In multicast pull dissemination, it is not too difficult to see that it is impossible for a server to construct online a schedule that minimizes average user perceived latency. 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. Even worse, it is shown in [19] that it is not even possible for a server to construct a schedule with bounded relative error. However, in the input distributions for which it was to be impossible to guarantee bounded relative error, 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 [30], it was suggested that one should seek algorithms that are guaranteed to perform well if the load of the server is not near peak capacity.

To date the only algorithm with a performance guarantee for low loads is Broadcast Equipoise [19]. Broadcast Equipoise broadcasts each file at a rate proportional to the number of outstanding requests for the file. Broadcast Equipoise guarantees average user perceived latency at most a constant times optimal if the load on the server is at most $1/4$.

Ideally what one would like is an algorithm that guarantees good performance with loads up to $1-\epsilon$. One difficulty of obtaining such an algorithm is that the choice of the document to broadcast depends not only on the obvious factors of the size of the document and the number of requests for the document, but also on the age of the requests. This insight was made more formal in [31] where is was shown that also broadcasting the most popular document could produce arbitrarily bad schedules in the case that all documents are the same size; further this result holds for even arbitrarily small loads. One possibility is a generalization of the Longest Wait First (LWF), proposed in [18], that experimentally seems to perform well for unit sized files [4]. 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 [32]. The generalization that we propose is to divide the counter by the file size.

A tight mathematical analysis of Broadcast Equipoise and LWF seems quite difficult. We are currently experimentally evaluating these algorithms. At the same time, we have looked into *flexible* scheduling schemes that exploit the semantics of the requested documents to aggregate requests that goes beyond the exact match of requests of the current scheduling approaches which we accordingly called *strict* [51]. The use of document semantics adds a new optimization dimension to the request aggregation process that allows for further multicast efficiency and scalability.

For example, in the context of RODS as well as a general wireless OLAP environment, every requested document is a summary table. An interesting property of summary tables which we call *derivation dependency*, is that one summary table can be derived from one or more summary tables. This means that a table requested by a client may *subsume* the table requested by another client. We propose two new, heuristic scheduling algorithms that use this derivation dependency 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 [51], is based on the *RxW* algorithm [1] and the second one, called *Subsumption-Based Scheduler* (SBS) [50], is based on the *Longest Total Stretch First* (LTSF) algorithm [1]. Further, they differ on the used criterion for aggregate requests. Otherwise, both STOBS and SBS are non-preemptive and considers the varying sizes of the summary tables. The effectiveness of the algorithms with respect to access time, power consumption and fairness were evaluated using simulation.

3.2 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 [53]. 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.

In multicast push data dissemination, several broadcast organizations have been proposed to encode a directory structure. These include incorporating hashing in broadcasts [29], using signature techniques [37] and broadcasting *index* information along with data [28, 15, 16, 54]. 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. In this project, our goal is to develop indexing schemes which provide the best balance between tuning and access time.

As part of our preliminary work, we have developed a new indexing scheme, called *Constant-size I-node Distributed Indexing* (CI) [3], 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 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) [28, 54], 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 [2].

Given that CI and DI are currently the best performing indexing schemes, we plan to initially implement these two schemes as part of the middleware. Besides optimizing existing schemes, one of our research goals would be to develop a mixed-adaptive indexing scheme that would be essentially optimal over all broadcast sizes.

3.3 Data Consistency and Currency

Presently, consistency and currency in both unicast and multicast data dissemination are similar to local cache consistency which ensures that only the most recent committed value of an item is stored (disseminated). 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, including some of our own [49, 8, 45, 44, 36], with the goal of achieving consistency and currency in broadcast environments beyond local cache consistency.

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 current investigation is directed towards the development of a general framework for correctness in broadcast-based data dissemination environments [46]. As part of our preliminary results, 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 [10, 22, 57] 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.

In the initial version of the middleware, we plan to implement three currency and consistency criteria: the traditional local cache consistency using invalidation reports, update consistency [49] and multiversion update consistency [44]. Our general objective is to investigate efficient ways to disseminate any control information. For example, consistency information can utilize the broadcast indexing structures. It is also our plan to expand our theoretical framework to include the case of a cache being maintained at the clients. Our 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 Transport Adaptation Layer

Several underlying multicast transport protocols are possible, including basic IP multicast [17], single-source multicast [26], reliable multicast [58], and end-to-end multicast [13]. 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 requirement of all applications [58], and so the middleware must be able to interact with various underlying multicast transport protocols. The objective of the Transport Adaptation Layer (TAL) is to enable the middleware to interact with different types of multicast transport within a uniform interface. 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 allows applications to select the most appropriate transport layer and get the benefits of a common multicast middleware. The Java Reliable Multicast (JRM) Protocol [48] 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.

An applications follows the following steps. 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 and passes it to the middleware. The middleware will 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 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 our middleware targets applications that need to scale to a very large receiver group. Consequently, the transport must avoid the problem of ACK/NACK implosion, through, for example, ACK aggregation [24] or NACK suppression [21, 23, 42]. Alternatively, a reliable multicast transport can adopt open-loop reliability solutions such as cyclical broadcast [5] or error-correcting codes [12]; 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.

Finally, it should be noted that our goal is to to restrict a given application to a specific multicast protocol. An additional functionality of TAL is to integrate various multicast packages available so that clients supporting different multicast protocols can communicate with each other through the middleware. Currently, two packages are being considered - JRMS that supports the TRAM and LRMP protocols and *Your Own Internet Distribution* (YOID).

5 Scheduling in Layered Multicast Environments

While the previous sections have focused on each of the individual building blocks in isolation, a second type of issues arises from the interaction of several components. Such interaction should be carefully tuned to achieve optimal performance. In the past, however, multicast data management components have been mostly designed, analyzed, and empirically evaluated in isolation, with the exception of [39] that addresses the interaction of caching and scheduling and of [27] that focuses on wireless environments. We feel that several additional research issues arise from the interaction of the middleware components.

In this section, we will focus on one such issue, namely the interaction between multicast push scheduling and *layered multicast* transport [41]. The purpose of layered multicast is to provide a mechanism for receivers to adapt their data reception rate to available bandwidth. Therefore, layered multicast shares common goals with congestion control, with the significant difference that multicast algorithms should scale to a large number of receivers. In a layered multicast scheme, data is multicast simultaneously on L different channels (*layers*). The transmission rate r_l at layer l is $r_0 = r_1 = 1$ and $r_l = 2r_{l-1}$ for $1 < l \leq L$ (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. 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 each channel is given by the rates r_l. Unlike previous multichannel models (e.g., [7]), 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 [33] gives a $(2 + \epsilon)$-approximation algorithm that runs in polynomial time for any fixed $\epsilon > 0$ [38]. The main open questions in this area are to find better approximation algorithms and to empirically verify their performance.

6 Conclusion

The paper describes software components and research issues in the area of middleware support for data dissemination over multicast channels. We reviewed the overall system design, and the current work on multicast pull scheduling, indexing, data consistency, currency, and scheduling for layered multicast channels. We have developed a number of new heuristic scheduling algorithms for selecting the data items to be broadcast at a given point in time and show using simulation that the new methods outperform existing ones.

We have also identified new ways to organize data on a broadcast and proposed appropriate caching techniques that ameliorate the negative effects due to any incompatibilities between a broadcast organization and a client data access behavior [52]. Due to space limitation, we have not elaborated on these here.

Several components of our proposed middleware have been implemented (multicast push scheduling, caching, and transport adaptation). These components are currently used to facilitate the development of a prototype application based on the RODS system. Our goal is to evaluate the effectiveness of our middleware in supporting the dissemination of critical information when outbreaks of diseases are detected.

The goals of this project stem from our long-term vision to provide fast and reliable information access to the masses, taking into account the individual user's specific needs. Every attempt is being made to develop theoretical frameworks and then transfer them into pragmatic environments so that their impact is far-reaching in the commercial as well as in the public health and homeland security sectors.

Acknowledgments: This work is a collaborative research which was benefited from the contribution of our students working on different aspect of the middleware: W. Li and W. Zhang of CWRU, and M. Sharaf, V. Penkrot, S. Roychowdhury and S. Vinchurkar of Pitt.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] N. Bansal and M. Harchol-Balter. Analysis of srpt scheduling: Investigating unfairness. In *Sigmetrics*, 2001.

[7] 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.

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

[9] 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.

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

[11] 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.

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

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

[14] H. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835 – 846, 1997.

[15] 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.

[16] 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.

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

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

[19] Jeff Edmonds and Kirk Pruhs. Broadcast scheduling: When fairness is fine. In *ACM/SIAM SODA*, 2002.

[20] M.M. Wagner *et al.* The emerging science of very early detection of disease outbreaks. *Journal of Public Health Management Practice*, 7(6):50–58, 2001.

[21] 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.

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

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

[24] 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.

[25] M. Harchol-Balter, N. Bansal, B. Schroeder, and M. Agrawa. Implementation of srpt scheduling in web servers. Technical report, CMU, 2001. Submitted to Sigcom 2001.

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

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

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

[29] 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.

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

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

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

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

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

[39] 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.

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

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

[42] C. Papadopoulos, G. Parulkar, and G. Varghese. An error control scheme for large-scale multicast applications. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, 1998.

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

[44] 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.

[45] 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.

[46] E. Pitoura, P. K. Chrysanthis, and K. Ramamritham. Characterizing the semantic and temporal coherency of broadcast-based data dissemination. Technical report, University of Ioannina, November 2000.

[47] Akamai Press Release. CNN.com uses Akamai's services on election day, serves record-breaking traffic, 2000.

[48] 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.

[49] 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.

[50] M. A. Sharaf and P. K. Chrysanthis. Semantic-based Delivery of OLAP Summary Tables in Wireless Environments. Technical Report, University of Pittsburgh (Submitted for publication).

[51] M. A. Sharaf and P. K. Chrysanthis. On-demand broadcast: New challenges and scheduling algorithms. In *Proceedings of the 1st Hellenic Conference on the Management of Data*, 2002.

[52] 0. Shigiltchoff, P. K. Chrysanthis, and E. Pitoura. Multiversion data broadcast organizations. In *Proceedings of the 6th East-European Conference on Advances in Databases and Information Systems*, 2002.

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

[54] 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.

[55] 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.

[56] M.M. Wagner, J. Espino, F-C. Tsui, T.A. Wilson, T.L. Tsai, L.H. Harrison, and W.A. Pasculle. The Role of Clinical Event Monitors in Public Health Survillance. *Journal of the American Medical Informatics Association*, 2002.

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

[58] B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, and M. Luby. Reliable multicast building blocks for one-to-many bulk-data transfer. RFC 3048, 2001.