An Optimized Multicast-based Data Dissemination Middleware: A Demonstration[*]

W. Li, W. Zhang, V. Liberatore
Electrical Engineering and Computer Science Dept.
Case Western Reserve University
10900 Euclid Ave., Cleveland, Ohio 44106
{wxl33, wxz24, vx111}@po.cwru.edu

V. Penkrot, J. Beaver, M. A. Sharaf, S. Roychowdhury, P. K. Chrysanthis, K. Pruhs
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
{vince, beaver, msharaf, src, panos, kirk}@cs.pitt.edu

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. However, allowing multicast communication introduces many non-trivial data management problems, such as caching, consistency, and scheduling. We have been building a middleware that 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 mechanism or on the application needs. Particular care has gone into the design of the algorithms to optimize the user-perceived level of service.

2 Proposed Middleware

Data Dissemination: In the proposed middleware, an application server can disseminate data by choosing any combination of the following three schemes: *multicast push*, *multicast pull*, and *unicast pull*. The outline of a possible configuration of our middleware and its relationship with the application and transport layers is shown in Figure 1.

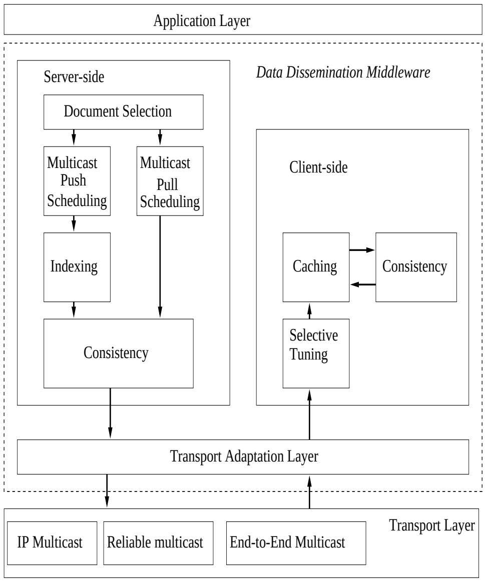


Figure 1. Proposed middleware data dissemination architecture.

When an application that utilizes all three data dissemination schemes, invokes the middleware, the following sequence of modules is typically invoked. The *document selection* unit periodically gathers statistics on application data units (ADUs) popularity. Once statistics have been collected, the server partitions ADUs into *hot*, *warm*, and

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

cold documents to be disseminated using multicast-push, multicast-pull and unicast-pull, respectively.

The server broadcasts an index of sorted URIs or URI digests on the multicast push channel. This quickly allows the client to determine whether a needed ADU is in the current hot broadcast set. If the ADU is not in the hot broadcast set, the client makes an explicit request to the server and simultaneously starts to listen to the multicast pull channel. If the ADU is cold, the requested ADU is returned on the unicast-pull connection. Otherwise, the client waits on the multicast pull channel until the requested ADU is transmitted.

The frequency and order in which hot ADUs are broadcast is determined by the *multicast push scheduling* component. We have implemented the flat and MAD algorithms [3] for the multicast push component.

The *multicast pull scheduling* component resolves contention among clients requests for the use of the warm multicast channel and establishes the order in which ADUs are sent over that channel. In the multicast pull scheduling component we have implemented the *Longest Total Stretch First (LTSF)* algorithm, which is the current experimental champion [1]. We have also implemented our *Subsumption-Based Scheduler (SBS)* which is based on LTSF and is tailored for disseminating multidimensional data [10].

Since *caching* can significantly improve performance by reducing the perceived latency, the middleware is designed to provide for client-side caching of ADUs. We have implemented the provably optimal caching algorithm from [7].

Transport Adaptation: The middleware utilizes the services provided by an underlying transport layer that enables multicast over the Internet. At this time, multicast protocols are the subject of extensive research and implementation, especially in the areas of reliable multicast [8] and end-to-end (overlay) approaches [4]. It is unlikely that a single multicast mechanism would be able to satisfy the requirement of all applications, and so the middleware must be able to interact with various underlying multicast transport protocols. However, different multicast protocols often present different API's and different capabilities. 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.

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. 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. TAL enables applications to select the most appropriate transport layer and get the benefits of a common multicast middleware.

Further, TAL can support a gateway application to integrate various multicast protocols. The objective of such gateway is to convert multicast flows from one protocol to another so that clients listening to different multicast channels can communicate with each other.

Currently, two packages have been considered: JRMS (that supports IP multicast, TRAM, and LRMP) and *Your Own Internet Distribution* (YOID) [11]. The integration of additional protocols is an area of on-going work.

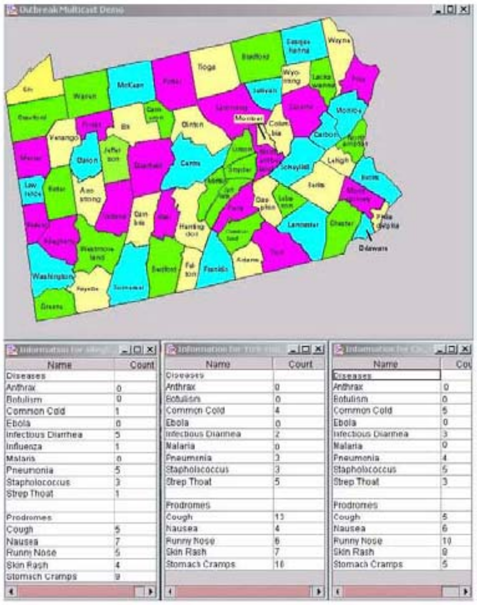
3 Related Work

Other prototype systems have addressed some of the data dissemination issues demonstrated here. In [6], the authors address the need for adaptive push-pull and the resulting data consistency issues. The DBIS-Toolkit [2] introduces a gateway for data delivery. DBIS differs from our middleware in four core items. First, our focus is to support multicast data transfers, whereas DBIS aims at the translation between different styles of data delivery (e.g., unicast and multicast) in one overlay network. Second, our middleware focuses on the core components that are the foundations for more complex system. Thus, the current middleware bridges the gap between networking research (e.g., the multicast transport layer) and broader data management issues. Third, we emphasize the performance, reliability, and security of each individual component by means of algorithmic and experimental methods. For example, we can prove analytically the optimality of several components and we have demonstrated their effectiveness through extensive simulations. Finally, we support a wider range of functionality, such as indexing and consistency.

Several Internet multicast protocols have been proposed over the years. Initially, the focus was on extension to the IP protocol to enable multicast within the core of the Internet infrastructure [5]. However, few providers have enabled IP multicast and the focus has shifted toward overlay multicast schemes (e.g., [4]) that implement this functionality through application-layer modules that execute at the network edges. The middleware extends these efforts at the transport layer with advanced data management functionality.

4 Demonstration

Real-Time Outbreak and Disease Surveillance (RODS): We will demonstrate our middleware within the context of the RODS application. RODS is a healthcare alert system developed by the Center for Biomedical Informatics at the University of Pittsburgh [9]. The RODS system has been deployed since 1999 in Western Pennsylvania and since December 2001 in Utah for the Winter Olympic Games.



Name	Count
Diseases	
Anthrax	0
Botulism	0
Common Cold	1
Ebola	0
Infectious Diarrhea	5
Influenza	1
Malaria	0
Pneumonia	5
Stapholococcus	3
Strep Thoat	1
Prodromes	
Cough	5
Nausea	7
Runny Nose	5
Skin Rash	4
Stomach Cramps	9

Name	Count
Diseases	
Anthrax	0
Botulism	0
Common Cold	4
Ebola	0
Infectious Diarrhea	2
Malaria	0
Pneumonia	3
Stapholococcus	3
Strep Thoat	5
Prodromes	
Cough	13
Nausea	4
Runny Nose	6
Skin Rash	7
Stomach Cramps	16

Name	Co
Diseases	
Anthrax	0
Botulism	0
Common Cold	5
Ebola	0
Infectious Diarrhea	3
Malaria	0
Pneumonia	4
Stapholococcus	5
Strep Thoat	3
Prodromes	
Cough	5
Nausea	6
Runny Nose	10
Skin Rash	9
Stomach Cramps	5

Figure 2. A snapshot of the emulated RODS systems.

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 multidimensional data. For example, queries often involve joins over several large tables to perform statistical analysis, e.g., computing daily percentage of patients with a particular prodrome (symptom) in a region for a one month period. Also, currently RODS displays spatio-temporal plots of patients presenting with seven key prodromes through a web interface.

RODS can use our multicast 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.

Application to RODS: The demonstration system utilizes the middleware to disseminate summary data found in RODS and it is a three tier application. The schema for the database was adapted from RODS. The database, implemented on an Oracle server, 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. For the purpose of this demonstration, 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 a user clicks on a county, the counts of disease and prodromes in that county are requested via the middleware (see Figure 2). If the requested data is not locally cached, then it is fetched from the server. When these counts are returned by the middleware, they are displayed on a separate window in a new table.

In our demonstration, we will show the performance of various combinations of the techniques and scheduling algorithms mentioned in Section 2. Specifically, with the help of a dynamic monitoring tool, we will present how different middleware configurations affect the perceived latency by the clients. Additionally, we will illustrate the middleware scalability at different requests rates.

Acknowledgments: We would like to thank all the members of the RODS Laboratory, especially its directors M. Wagner, and R. Tsui, for providing us with the needed information on RODS.

References

[1] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. *Proc. of the Fourth Annual ACM/IEEE Int'l Conference on Mobile Computing and Networking*, pp. 43–54, 1998.

[2] M. Altinel, D. Aksoy, T. Baby, M. J. Franklin, W. Shapiro, and S. B. Zdonik. Dbis-toolkit: Adaptable middleware for large scale data delivery. *Proc. of the ACM SIGMOD Conf.*, pp. 544–546, 1999.

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

[4] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. *Proc. of ACM SIGMETRICS*, pp. 1–12, 2000.

[5] S. Deering. Multicast routing in internetworks and extended lans. *Proc. of the ACM SIGCOMM Conf.*, pp. 55–64, 1988.

[6] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Dissemination of dynamic data. *Proc. of the ACM SIGMOD*, pp. 599, 2001.

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

[8] M. Luby, V. K. Goyal, S. Skaria, and G. B. Horn. Wave and equation based rate control using multicast round trip times. *Proc. of the ACM SIGCOMM Conf.*, pp. 191–204, 2002.

[9] RODS: http://www.health.pitt.edu/rods.

[10] M. A. Sharaf and P. K. Chrysanthis. Semantic-based delivery of OLAP summary tables in wireless environments. *Proc. of the ACM CIKM Conf.*, 2002.

[11] P. Francis. Yoid: Extending the Internet Multicast Architecture. ICIR Techical report, 2000.