IBP-MIME: Controlled Delivery of Large Mail Files

Wael R. Elwasif Micah Beck James S. Plank

April 2, 1999

Technical Report UT-CS-99-421
Department of Computer Science
Uninversity of Tennessee
April 2, 1999

Abstract

The current model of mail delivery over the Internet guarantees that an email message, when delivered, will have all its parts present at the recipient's mail server. The message is typically spooled pending retrieval by the recipient at a later time. This model has serious efficiency problems when handling messages that contain large files. Such messages impose a heavy burden on storage resources at the mail server, where they can occupy space for an indefinite period of time pending retrieval. The situation is made worse when the message is directed to multiple users who share the same mail server. In this paper we present a new model for delivery of email messages containing large files. The proposed model separates the text part of the email message from the non-text part, and entrusts the latter to a set of network storage servers that keep it until prompted for delivery by the recipient. The proposed model makes use of new initiatives in distributed storage that are part of the Internet2 project. In addition, it allows for recipient initiated preprocessing of the sent file remotely, before actually receiving it. This option is becoming more feasible with the advent of computational servers such as NetSolve and can have wide implications in areas such as electronic commerce and security.

1 Introduction

The use of email as a content-exchange medium has gained tremendous momentum over the last decade with the almost universal Internet connectivity in academia and the research community, and the relatively recent commercialization of the Internet. As the contents of email messages evolved from simple text messages to messages containing (usually large) binary files, schemes were developed to address the issue of encoding and decoding such files for transport using existing mail protocols (in particular SMTP [JP85]) and tools (e.g. the `uuencode` and `uudecode` tools). The development and subsequent standardization of MIME (Multipurpose Internet Mail Extensions) [RW96] has made the exchange of non-text content via email an easy task. However, these schemes, while addressing the issue of content encoding, do not address the efficiency issues introduced through the use of existing email delivery protocols, which were designed when smaller messages were the norm.

Users who wish to exchange large files, but are geographically separated by large distances, have several options in carrying out such an exchange.

1. The sender can store the file(s) in question on disk or tape and send it to the intended recipient(s) via snail mail. This approach, while very slow, may be the only viable method for extremely large data sets common in certain research fields (e.g. astrophysics).

2. Alternatively, the sender can use a MIME-enabled mailer to send the file(s) directly to the receiver using existing mail protocols. While faster than surface mail, this approach has many drawbacks. First, it consumes resources on both the sender's system (in the mail queue) and on the receiver's system (in the mail spooler). If the attached file is large enough, there may not be enough space on the queue/spooler to accommodate it. Additionally, the act of MIME encoding expands the file, making it larger still. If the recipient is connected to the spooler via a slow link (for example via Sun NFS over a 10 Mb Ethernet), the performance of the user's mailer may suffer significantly until file is moved from the spooler. And finally, if the sender wants to broadcast the file to a number of recipients, the file uses up spooling resources on every recipient's system, or multiple copies on the same system.
3. A third approach is to use one of the existing file transfer protocols, such as FTP [Pos82] or HTTP [FGFBL97]. With this approach, the sender exports the file on an FTP or HTTP server, and emails the address to the recipient. The recipient, upon reading the email, downloads the file from the sender's FTP or HTTP server. This approach solves many of the problems of the above two methods, especially those regarding resources on the recipient's spooler. However, there are drawbacks. If the network path between the sender and the recipient is slow, then it may take a long time for the recipient to download the file. Instead of consuming resources at the receiver's site, it consumes resources at the sender's site, and those resources must be reclaimed at the sender's discretion once he/she knows that the receiver has downloaded it. An additional drawback is that by serving the file with FTP or HTTP, it either becomes visible to the entire world, or the recipient must be authenticated by the server by a password or similar means. Finally, there may be administrative problems in putting the file on such server.

In this paper, we propose a different delivery mechanism for large files that has better performance and fewer administrative details. It is based on the ability to explicitly manage storage in the network, and therefore to use the network to buffer messages. This ability is at the core of the Internet2 Distributed Storage Infrastructure (I2-DSI) project [BM98], described below in Section 2. We make use of a piece of middleware called the Internet Backplane Protocol (IBP) that allows us to manage and use storage in the network. The result is a mechanism that we call IBP-MIME.

The basic structure of IBP-MIME is as follows. The sender stores the file to an IBP server in the network (preferably close to the recipient) and mails the recipient a pointer. Upon receiving the mail, the recipient downloads the file from the network, and then deletes it. While similar to the FTP/HTTP method described above, it has four distinct advantages. First, neither user expends extra storage resources for spooling or buffering. Instead, the network is used as a buffer. Of course, the success of this approach relies on there actually being storage in the network, which is currently a reality under I2-DSI. Second, the sender can direct the file to be stored in a network location close to the receiver. Thus, when a receiver downloads the file, he/she will receive it much faster than with anonymous FTP or HTTP. Third, the receiver deletes the file on the network storage after downloading it. Fourth, the file is neither visible to the entire world, nor protected by a password-based scheme, making the transmission both safe and convenient.

One can view this method as a hybrid of email and anonymous FTP: the sender starts the file moving toward the receiver, who completes the communication.

The rest of this paper is organized as follows. The I2-DSI project is detailed in section 2. We provide relevant details of the IBP storage server architecture in section 3. In section 4, we describe the elements in the proposed IBP-MIME architecture. Applications that can make use of the proposed architecture, implementation issues, and deployment are discussed in section 5. Finally, we present the conclusions and possible areas of future work in section 6.

2 I2-DSI

The Internet2 Distributed Storage Infrastructure (I2-DSI) initiative is one part of the Internet2 project. I2-DSI's stated goal is to resolve accessibility issues associated with sharing and using

Internet-based educational content [BM98]. A fundamental principle of I2-DSI is that explicit storage in the network is necessary to achieve high performance in future generation networking. To that end, the I2-DSI project has received donations of (currently five) large storage servers that are deployed throughout the United States. At least five more have been promised by various industry affiliates. IBP-MIME fits the mission of the I2-DSI project, and the storage servers in the I2-DSI deployment are available for use by the IBP-MIME project. As described below in section 3, IBP is structured so that private storage owners may "loan" their resources to the I2-DSI pool without impacting themselves adversely. Thus, the requirement of network storage IBP-MIME is one that is currently met by the I2-DSI deployment, and should continue to be met in the future.

3 IBP: The Internet Backplane Protocol

The Internet Backplane Protocol (IBP) is a piece of middleware for the management and accessing of remote storage. IBP is structured as server daemons invoked by the storage resource owners, and a library of client calls to be linked with client applications. IBP may be envisioned as providing a gateway to distributed storage services as DNS does to directory services. Motivation for IBP and complete description of the interface may be found in [BPM98]. We describe the features relevant to IBP-MIME below.

3.1 The IBP server

Each IBP server manages storage that it serves up to the clients in the form of append-only byte arrays. The IBP server is designed to run without any special access privileges. IBP servers may be started by any user on a machine, and servers work subject to policies that allow the initiating user some control over how IBP makes use of the storage. One of the policies is that the IBP server may use spare physical memory, or it may use disk space that the initiating user may revoke either arbitrarily or on a set schedule. The intent of this feature is to encourage users to donate their resources to IBP, since they can reclaim the resources rather easily.

In its current form, IBP supports two types of storage: reliable (guaranteed) storage and volatile storage which can be reclaimed by the initiating user if the need arises. Clients choose which type of storage best meets their requirements (e.g. volatile storage can be used to implement replication services). In addition, the IBP server supports time-limited storage areas, which are purged with no client action at the expiration of a specified life time, and permanent storage areas which are purged only at the client's request. The combined effect of these two properties (reliability and life time) determine if a specific storage area exists on the IBP server at any point in time.

3.2 The IBP client interface

IBP client calls may be made by *anyone* who can attach to an IBP server. The IBP clients communicate with the servers via a TCP/IP stream. Clients initially gain access to byte arrays by allocating storage on an IBP server. If the allocation is successful, the server returns three *capabilities* to the client: one for reading, one for writing and one for management. These capabilities can be viewed as names that are assigned by the server. Currently, each capability is a text string encoded with the IP identity of the IBP server, plus other information that is intended to be interpreted only by the server. Applications may pass IBP capabilities among themselves without registering these operations with IBP. In this respect, IBP capabilities are like URL's in web systems.

A client needs to identify an IBP server only for the initial allocation of storage. All other requests are performed through the capabilities. Below we present the subset of the IBP client interface that is relevant for implementing IBP-MIME.

- `IBP_capability_set IBP_allocate(host, size, attributes)`
  This call allocates a new storage area on the IBP server running on *host*. The client specifies

attributes that determine the area's maximum allowable size, its reliability and its life time. On success, the server returns a trio of capabilities that control access to the new storage area.

- `IBP_store(write-cap, data, size)`
  In this call, a chunk of data in memory of size *size* is appended to any previous data at the storage area accessed through the write capability *write-cap*. Note that *any* client may call `IBP_store()`, so long as the capability *write-cap* is valid.
- `IBP_read(read-cap, buf, size, offset)`
  The *IBP_read()* call allows a client to retrieve *size* bytes of data that has been stored at the storage area accessed through the read capability *read-cap* at the specified offset.
- `IBP_copy(read-cap, writ-ecap, size, offset)`
  This call copies *size* bytes, starting at offset *offset* from the storage area accessed through the read capability *read-cap* to the end of the storage area accessed through the write capability *write-cap*. *IBP_copy()* is an example of a third-party interaction – client A employs *IBP_copy()* to instruct server B to send bytes directly to server C. This is one of the primitives that makes IBP different from other storage management systems such as distributed file systems, databases and content servers.
- `IBP_manage(manageCap, cmd, capType, info)`
  This call allows the client to perform certain management operations on the storage area accessed through the management capability *manageCap*. The specific operation is specified in the *cmd* parameter and *capType* specifies the target capability for some commands (read capability or write capability). *Info* is an in-out parameter that conveys information back and forth between the client and the IBP server. The current implementation of IBP supports management operations to increase the reference count to a capability, decrease such a reference count, increase/decrease the maximum storage area, and probe the storage area for its current state. Decrementing the reference count of a read capability can result in the storage area being deleted from the IBP server (if the reference count drops to zero).

Currently, all IBP client calls are synchronous in that they block until the call is completed. If clients are threaded, this should impose no restrictions of client functionality or performance. If the synchronous calls should prove to be a problem to non-threaded clients, asynchronous calls will be implemented.

4 The IBP-MIME architecture

The basic IBP-MIME architecture is outlined in figure 1. The sender, instead of MIME-encoding an attachment before sending, performs an IBP-staging operation that moves the attached file (unencoded) to one (or more) IBP servers in the IBP cloud(using *IBP_allocate()* and *IBP_write()* calls). This operation produces a file containing meta-data that describes the staged file (this file basically contains the access capabilities to the staged file, along with any other relevant information that could be of use to the receivers). This returned file is MIME encoded, given an extension that identifies it as a pointer to an IBP-staged file (e.g. *.ibp*), and sent as an attachment along with any text message.

When the recipient retrieves the email message from the mail server, a special IBP-MIME handler is invoked to remotely process the staged file. The complexity of such processing depends on file size, type, and intended use by the recipient. The recipient could simply download the file to local storage (using *IBP_read()*) and invoke a local application on it (e.g. an image viewer for an image file). Alternatively, the recipient may decide to direct the received file to a *local* IBP server (using *IBP_copy()*) for future retrieval and processing. Another option is for the recipient to use an IBP-enabled (possibly remote) computational server to process the incoming file, thus eliminating the need for local delivery altogether. This third alternative can be useful in many areas as will be outlined in section 5.

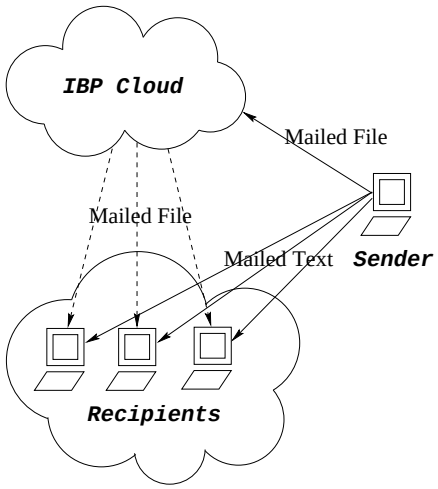

Figure 1: file delivery using *IBP-MIME*

In its simplest form, an IBP-MIME communication makes use of a single IBP server that is accessed by both the sender and receiver(s). This basic scheme can be enhanced to make better use of existing resources. For example, the file can be transferred progressively closer to the recipient(s), with a distributed naming service built on top of IBP used to retrieve the *current* location of the file at time of actual delivery. Other enhancements can be achieved through replicating the staged file at one or more IBP servers that are located *close* to various recipients of the original email message.

5 Applications and implementation

5.1 Applications

In this section, we list some applications that could make use of the proposed IBP-MIME architecture. We can broadly classify such applications into three main categories.

- **Wide area collaborative computing**: Scientists collaborating on research projects can make use of the proposed architecture to facilitate the exchange and processing of massive data files. By combining the network storage model embodied in IBP with the availability of computational servers such as NetSolve [CD97], scientists can have access to computational services with little or no administrative overhead. As an example, a scientist can use IBP-MIME to send a huge uncompressed video file to a colleague, who can instruct the local IBP-MIME handler to forward the incoming file directly to to an IBP-enabled NetSolve server for processing and/or enhancements, before examining it and discussing the results with the sender. This could be particularly useful in areas such as medical imaging, where files in the 2-20GB range are not uncommon.
- **Commercial network services**: The receiver may initiate a pre-delivery processing of the incoming message via a commercial third party. One example of this scenario would be a virus detecting server checking all attachments before delivery to the target user. By having the message remotely stored and checked before delivery, any chance of a security risk is virtually eliminated.
- **Network and storage management**: This area of applications aims at improving utilization of network and storage resources through control of the large file movement. A *best effort* algorithm can be developed to progressively move the file in transit closer to the recipient, while not imposing unnecessarily strict requirements on available network resources. In the

current delivery model, the entire email message is delivered as soon as possible to the target system, while the large file part of the message may not be accessed/needed until a later time. By having IBP *hubs* along the way, file movements can be performed in such a way so as not to cause network congestion along the way. This issue is linked to the Quality of Service (QoS) in networking, which is currently an active area of research.

5.2 Implementation and deployment

We have successfully implemented the client and server parts of the IBP storage server architecture. Currently there are several IBP servers running on several machines that are part of the I2-DSI infrastructure. A prototype for the IBP-MIME has also been developed and tested successfully with the existing IBP servers. We are currently working to expand the base of installed IBP servers and enhance the functionality available to end users via IBP-MIME.

5.3 An example

In this section we present an example that illustrates the advantages obtained through the use of IBP-MIME. Assume that a user at the University of Tennessee campus wishes to send an email that contains a large video file to a user at Princeton University. We present the expected transfer and retrieval time using regular mail protocols, and using IBP-MIME through the I2-DSI IBP server running at University of North Carolina. The bandwidth values were determined empirically on a work day in March 1999. We use the following nomenclature:

Original file size (S_{origin})	=	5.355 MB
MIME-encoded file size (S_{mime})	=	7.24 MB
Bandwidth from UT to Princeton ($BW_{UT \to P}$)	=	0.1775 MB/Sec.
Bandwidth from UT to DSI machine ($BW_{UT \to DSI}$)	=	0.2571 MB/Sec.
Bandwidth from DSI to Princeton ($BW_{DSI \to P}$)	=	0.4601 MB/Sec.
NFS Disk to memory bandwidth (BW_{NFS})	=	0.166 MB/Sec.
DSI machine disk to memory bandwidth (BW_{DSI})	=	9.865 MB/Sec .

Method	Operation	Formula	Time (Sec.)
Regular mail	Send	$S_{mime} * (\frac{2}{BW_{NFS}} + \frac{1}{BW_{UT \to P}})$	128.017
	Retrieve	$\frac{S_{mime}}{BW_{NFS}}$	43.614
HTTP/FTP	Retrieve	$S_{origin} * (\frac{1}{BW_{NFS}} + \frac{1}{BW_{UT \to P}})$	62.43
IBP-MIME	Send	$S_{origin} * (\frac{1}{BW_{NFS}} + \frac{1}{BW_{UT \to DSI}} + \frac{1}{BW_{DSI}})$	53.63
	Retrieve	$S_{origin} * (\frac{1}{BW_{DSI}} + \frac{1}{BW_{DSI \to P}})$	12.182

Table 1: Large file transfer using IBP-MIME

The *Send* operation refers to the steps taken by the sender to deliver the file to a location where it can be retrieved by the receiver. The *Retrieve* operation consists of all actions taken by the receiver to fetch the file from a known location. We assume that the time to transfer the text part of the

message is relatively small and will not significantly affect the results. We also assume that all servers involved are not heavely loaded. As can be seen from Table 1, the retrieval time experienced by the recipient is significantly lower when using IBP-MIME. Although the total send and retrieve time is minimized with HTTP/FTP, in situations where the receiver reads the mail more than 1 minute after the sender sends it, the IBP-MIME solution is far preferable..

6 Conclusions and future work

In this paper, we have presented an architecture for controlled delivery of large files via email using an infrastructure of distributed storage servers. The proposed model allows for better utilization of resources on the end users' systems through the use of the network itself as a large buffer via the IBP protocol. We envision a deployment of IBP servers that closely follows the current pattern of bandwidth installation, with massive IBP servers close to the Internet backbone, and more (albeit smaller) IBP servers closer to end users.

There remain several important questions that merit further investigation regarding the efficient use of IBP-MIME. Foremost among them is the issue of fault tolerance: how to guard against mail loss due to IBP server crash? The scalability of any algorithm that is used to manage file movement between various IBP servers is also important. Work is also needed to formulate efficient policies for fair storage allocation on the intermediate IBP servers and in securing transmitted files while in transit.

The IBP code is currently available at `http://www.cs.utk.edu/~elwasif/IBP`. IBP-MIME code can be found at `http://www.cs.utk.edu/~elwasif/IBP-MIME`.

References

[BM98] M. Beck and T. Moore. The Internet2 Distributed Storage Infrastructure project: An architecture for internet content channels. *Computer Networking and ISDN Systems*, 30(22-23):2141–2148, 1998.

[BPM98] M. Beck, J. S. Plank, and T. Moore. IBP – the Internet Backplane Protocol: Two page summary. Technical Report CS-98-407, University of Tennessee, November 1998.

[CD97] Henry Casanova and Jack Dongara. NetSolve: A Network-Enabled Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3), Fall 1997.

[FGFBL97] R. Fielding, J. Gettys, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. `http://www.ietf.org/rfc/rfc2068.txt`, January 1997.

[JP85] J. Reynolds J. Postel. File Transfer Protocol – FTP. `http://www.ietf.org/rfc/rfc959.txt`, October 1985.

[Pos82] Jonathan B. Postel. Simple Mail Transfer Protocol – SMTP. `http://www.ietf.org/rfc/rfc821.txt`, August 1982.

[RW96] P. Resnick and A. Walker. The text/enriched MIME Content-type. `http://www.ietf.org/rfc/rfc1896.txt`, February 1996.