IBP-Mail: Controlled Delivery of Large Mail Files

Wael R. Elwasif   James S. Plank   Micah Beck   Rich Wolski

Department of Computer Science
University of Tennessee
Knoxville, TN 37996

[elwasif,plank,mbeck,rich]@cs.utk.edu

Appearing in:
NetStore '99: Network Storage Symposium
Seattle, WA, October, 1999
http://dsi.internet2.edu/netstore99

http://www.cs.utk.edu/~plank/plank/papers/NS99-IBPM.html

Abstract

IBP-Mail is an improvement to the current state of the art in mailing large files over the Internet. It arises from the addition of writable storage to the pool of available Internet resources. With IBP-Mail, a sender registers a large file with an IBP-Mail agent, and stores it into a network storage depot. The sender then mails a pointer to the file to the recipient using the standard SMTP mailing protocol. When the receiver reads the mail message, he or she contacts the agent to discover the file's whereabouts, and then downloads the file. In the meantime, the agent can route the file to a storage depot close to the receiver.

IBP-Mail allows for an efficient transfer of the file from sender to receiver that makes use of the asynchronous nature of mail transactions, does not expend extra spooling resources of the sender or receiver, and utilizes network resources efficiently and safely. In this paper, we detail the current limitations of mail systems, describe the software architecture of IBP-Mail and its current implementation, and give a performance case study.

Currently, IBP-Mail uses network storage servers that are part of the Internet2 Distributed Storage Infrastructure project. It is based on the Internet Backplane Protocol (IBP) that has been developed at the University of Tennessee for the explicit purpose of utilizing network storage to improve application performance and functionality.

1 Introduction

Electronic mail is the *de facto* mechanism for communication on the Internet. In an electronic mail transaction, a sender desires to transmit a message to one or more recipients. Typically, this is done using the SMTP mailing protocol [13]. The sender sends his or her message to a local SMTP daemon which saves the message in its *spooling area* to a local file. Meanwhile, the SMTP daemon opens an end-to-end connection with the receiver's STMP daemon, and sends the message in its entirety to this daemon. When the message has been received by all recipient SMTP daemons, the sender daemon deletes it from its spooling area. The receiver stores the message in its spooling area, where it remains until the recipient asks for it using a mail client program. The recipient explicitly deletes the message from its SMTP daemon's spooling area when it is done with the message.

Electronic mail works extremely well for most mailing usages. However, the model has limitations when the sender desires to transmit large data files to one or more receivers. Given the current network infrastructure, there are four basic ways that a sender can employ to transmit a large data file to one or more recipients:

1. Send the file as an encoded mail message

One of the earliest email tools is Unix's `uuencode`. `Uuencode` transforms any binary data file into an ASCII message that can be sent and received by virtually all mail clients. `Uudecode` then transforms the message back into a binary data file. Later in time, the MIME protocol [4,8,14] was developed as a standard format for attaching non-textual files to mail messages and identifying their types so that mail clients can bundle multiple data files into an email message, send them to recipients, and then have the recipients unbundle them and launch file-specific applications to act on them.

While the MIME standard has certainly made the task of sending and receiving files much easier than using `uuencode/uudecode`, they share the same fundamental limitations when the files are very large. First, in order to work across all SMTP daemons, the files must be encoded into printable ASCII, which expands them roughly by a factor of 1.4. Second, spooler space at both the sender and the receiver must be expended for the file. Typically, spooler space is allocated by an institution's system administrator and is shared by all users of the system. Often, there is a limit on the amount of data that may be stored in the spooler space, meaning that files over a certain size cannot be sent with MIME or `uuencode` encoding. Moreover, if a sender sends to multiple recipients, a separate copy of the mail message will be stored for each recipient, even if recipients share the same spooling space. Therefore, for large files, the `uuencode`/MIME mechanism for sending the file is often untenable.

2. Store the file locally and mail a pointer

A typical example has the sender storing the file in a local HTTP [7] or anonymous FTP [12] server, and mailing the receiver a pointer to it. When the receiver reads the mail, he or she downloads the file. This solution solves the problem of expending spooler resources at the sender and receiver, but still has limitations. First, the sender may not have access to an HTTP or FTP server that contains sufficient storage. Second, a significant period of time may pass between the time that the sender sends the message and the time that the receiver downloads the file. When the file is served at a local HTTP or FTP server, this time is wasted, because the receiver will not start the download until he or she reads the message. Third, there is no automatic mechanism to inform the sender that the file has been downloaded, and may therefore be deleted from the server. And fourth, since HTTP and anonymous FTP downloads may be initiated by anyone on the Internet and both protocols have directory listing operations, the file may be discovered and read by unintended recipients.

3. Upload the file remotely and mail a pointer

A typical example of this is when the receiver owns an anonymous FTP server with an `incoming` directory in which anonymous users may upload files. The sender then uploads the file and mails the receiver a pointer to it. The receiver may then download the file upon reading the mail, and delete it when the download is finished. This solution solves many of the problems with the previous two solutions, but has three limitations. First, the sender has to wait for the file to be stored remotely before he or she may mail the pointer. Second, the receiver, in allowing anonymous users to write to his or her FTP server, opens the server to invasion by malicious users. Finally, the receiver may not have access to such a server.

4. Save the file to tape and use surface mail

While this solution seems truly primitive, it is sometimes the only way to transmit very large files from a sender to a receiver.

The IBP-Mail Solution

The IBP-Mail solution to this problem relies on the existance of writable storage depots in the network. Given such depots, the sending of large data files can be performed with near optimal efficiency. With IBP-Mail, the storage depots are registered with IBP-Mail agents. The sender registers with one of these agents and stores the file at a nearby depot. The sender then mails a message with the file's ID, as assigned by the agent. Meanwhile, the agent attempts to move the file to a depot close to the receiver. When the receiver reads the message, he or she contacts the agent to discover the whereabouts of the file, and downloads it from the appropriate depot. The file may then be deleted from the depot.

The IBP-Mail solution exhibits none of the problems of the above solutions:

- The file is not expanded due to mailer limitations.
- No extra storage resources are required at the sender or receiver.
- If enough time has passed between the message's initiation and the receiver's reading of the message, the file should be close to the receiver for quick downloading.
- The sender does not have to wait for the file to reach the receiver before sending the message.
- Storage resources are not opened to malicious users.

- Multiple recipients may downlaod shared copies of the file.

Additionally, the IBP-Mail model may be extended to allow other interesting functionalities such as compression, data mining, fragmentation of very large data files, and uses of remote compute cycles.

The remainder of this paper is outlined as follows. First we discuss the Internet2 Distributed Storage Infrastructure (I2-DSI) and Internet Backplane Protocol (IBP) projects and how they address the assumption of the existance of storage depots in the network. Next, we detail the exact software architecture of IBP-Mail and its current implementation. Finally, we present a performance example and discuss related work.

IBP-Mail exists as an application available to *anyone on the Internet with a web browser* with no code porting necessary. Although there is an optimized mail sending program for Unix systems, files may be sent using IBP-Mail by pointing any web browser to `http://www.cs.utk.edu/~elwasif/cgi-bin/ibp-mail.cgi`. The mail message which is sent to the receiver contains a web pointer that allows the receiver to download the file using any web browser. If the receiver uses a web-enabled mail client (such as Netscape mail), this is all achieved with one click of the mouse. Therefore IBP-Mail requires no special infrastructure for general-purpose use. It currently uses the I2-DSI deployment machines as storage depots.

2 I2-DSI and IBP

A central assumption of IBP-Mail is that storage exists in the network. Given the legacy Internet infrastructure, this is not a valid assumption. However, the infrastructure of the Internet is changing. The Internet2 Distributed Storage Infrastructure (I2-DSI) [1] project is paving the way for storage resources to be added to the network commons. Currently, I2-DSI has received donations of five machines from IBM that each contain 2 gigabytes of RAM, 72 gigabytes of disk storage and 900 gigabytes of tape storage. These are distributed around the country (Indiana, North Carolina, South Dakota, Tennessee and Texas), serving storage to I2-DSI applications. IBP-Mail is one such application. The logistical backbone (L-Bone) project at Tennessee [11] is a deployment project whose goal is to provide storage to Internet applications such as IBP-Mail. The L-Bone has been designed to encourage machine owners to donate their spare storage resources. If successful, the L-Bone should easily satisfy the assumption of storage availability in the network.

The Internet Backplane Protocol (IBP) is a related project from the University of Tennessee [11]. IBP is software that allows applications to manage and make use of remote storage resources. It is structured as server daemons that run at the storage sites, serving up dedicated disk, spare disk, and physical memory to IBP clients. IBP clients can run anywhere on the Internet and do not have to be authenticated with the IBP servers. Thus, when an IBP server is running, *anyone* may use it.

There are several features of IBP's design that make offering storage as a network resource feasible:

- **There are no user-defined names.** IBP clients allocate storage, and if the allocation is successful, then it returns three *capabilities* to the client — one each for reading, writing, and management. These capabilities are text strings, and may be viewed as server-defined names for the storage. The elimination of user-defined names facilitates scalability, since no global namespace needs to be maintained. Additionally, there is no way to query an IBP server for these capabilities, so unlike an anonymous FTP server, there is no way for a user to gain access to a file unless he or she knows its name *a priori*.
- **Storage may be constrained to be *volatile* or *time-limited*.** An important issue when serving storage to Internet applications is being able to reclaim the storage. IBP servers may be configured so that the storage allocated to IBP clients is *volatile*, meaning it can go away at any time, or *time-limited*, meaning that it goes away after a specified time period.
- **Clients can direct remote data operations**. One of IBP's primitive operations is the ability to copy a file from one IBP server to another. This is ideal for applications that need to route data from one place to another, and direct this routing from a third party.
- **Reference counts are maintined on the files**: One operation that IBP clients may perform with management capabilities is an explicit increment and decrement of reference counts for reading and writing. When the write reference count is zero, the file becomes read-only. When the read reference count becomes zero, the file is deleted.

3 The Structure of IBP-Mail

IBP-Mail has been designed with three goals in mind. First, it must work, and be available to general Internet users with a minimum of effort. Second, it must solve the problems with mailing large data files as detailed in the Introduction. Third, it must facilitate testing. In this section, we detail the structure of IBP-Mail that allows it to meet these three goals.

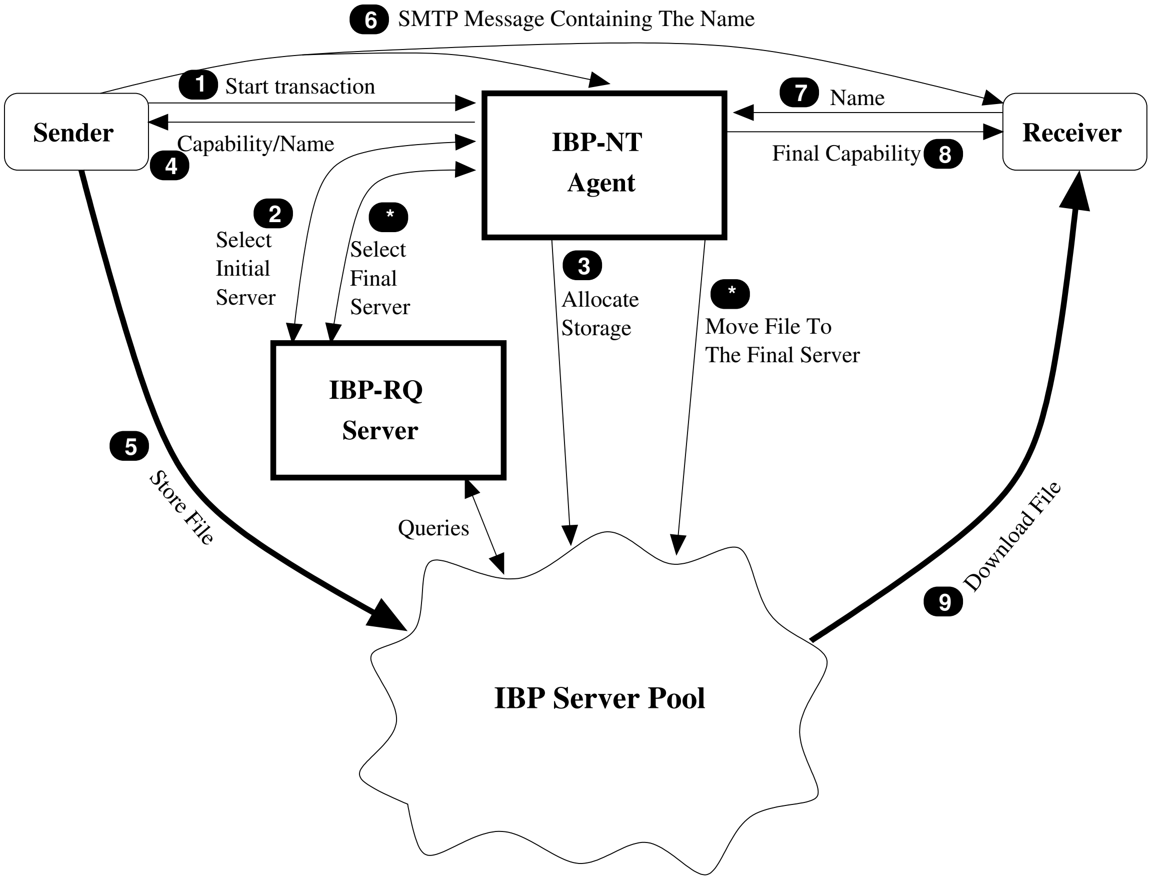

Figure 1: An IBP-Mail Transaction

IBP-Mail consists of three components, as depicted in Figure 1. These are:

- **A pool of IBP servers**. Only one server is necessary in the pool for IBP-Mail to work. Ideally the pool will consist of a collection of servers with wide geographic distribution.
- **IBP-RQ: A registration/query server**. This is a server that maintains the state of the IBP server pool. Servers register with the IBP-RQ server, and clients may query the IBP-RQ server for information about servers. More information about the IBP-RQ server is given below.
- **IBP-NT: An agent for naming and transport**. The IBP-NT keeps track of where the data file is in the IBP server pool, and directs the movement of the file from server to server. More information about the IBP-NT is given below.

An IBP-Mail transaction takes the following nine steps, also outlined in Figure 1:

1. The sender contacts an IBP-NT agent. In the current implementation, there is only one such agent, but multiple agents are possible. The size of the file is communicated to the agent.
2. The IBP-NT agent queries the IBP-RQ server to list appropriate IBP servers from the server pool that can store the file.
3. The IBP-NT agent allocates storage for the file on an IBP server, and receives the capabilities for the storage.
4. The IBP-NT agent creates a name for the transaction and returns that name, plus the write capability of the file, to the sender.
5. The sender stores the file into the IBP server.
6. The sender sends mail to the receiver with the name of the transaction. At the same time, the sender informs the agent that the file has been written to the IBP server.

7. The receiver presents the name to the IBP-NT agent.

8. The IBP-NT agent returns the read and manage capabilities of the file to the receiver.

9. The receiver downloads the file from the IBP server, and may delete it if desired, by decrementing the file's read reference count. Note that if a file is shared by multiple recipients, the agent may increment its reference count to equal the number of recipients, and then the file will be deleted only after each recipient has decremented the reference count (or when the time limit expires).

There are two steps in Figure 1 labeled with an asterisk. These may be performed by the IBP-NT agent after the sender stores the file, if the agent determines that it may be able to move the file close to the receiver. If enough time passes between steps 6 and 7 (due to the receiver not reading his or her email instantly), then this time may be used by the agent to move the file close to the receiver(s), thereby improving the time for downloading. If a receiver tries to download the file before it has been copied, it may do so from the original IBP server, with reduced performance.

The current implementation of each individual component is described below.

3.1 The IBP-RQ Server

The IBP-RQ server provides basic directory registration and query services to the IBP-NT agent. The directory part of the server is a two-level structure, with *groups* containing *hosts* (IBP servers). A *host* can belong to more than one *group* at the same time.

While the directory structure can be implemented using existing technologies (e.g. LDAP [20]), what sets the IBP-RQ server apart is the query component. Queries in IBP-RQ provide a ranking of IBP servers according to selection criteria embedded into IBP-RQ. Currently, two rankings are implemented: free storage available, and proximity to a given host. The former is easy to implement given IBP semantics. The latter is more difficult. Currently, each IBP server runs a SonarDNS daemon [18], and the IBP-RQ can force all its servers to perform Sonar performance queries to the given host. While this implementation is not a long-term solution (for example, it is not scalable to a large number of IBP servers), it suffices for our current implementation. Similarly, IBP server failures are currently not detected, and the IBP-RQ is not a distributed service, problems which will have to be solved as IBP-Mail scales.

3.2 The IBP-NT Agent

The IBP-NT agent provides naming and transport to IBP-Mail clients. Currently there is only one, but there could potentially be many IBP-NT agents working independently. A client starts a mail transaction by contacting an IBP-NT agent and providing initial information about the transaction: sender location and file size. The agent then queries the IBP-RQ server to find an IBP server in which the sender should insert the file. Currently, this query attempts to find a server with enough storage that is closest to the sender (using Sonar DNS's metrics for closeness). In the future, this may be improved to take account of server load or reliability as well. The agent allocates the storage, and then stores the capabilities for that storage into a second IBP file located at a server on or near the agent's host. We will call this the name file. The read capability of name file is returned to the sender, along with the write capability of the first IBP file. The sender stores the data into the first IBP file, and then sends mail to the receiver with the capability name file. This capability serves as a name for the mail transaction.

At the same time, the sender informs the agent that the IBP file is has been stored, and gives the location(s) of the receiver(s). Meanwhile, the agent is free to transport the data file to a server near the receiver (or receivers). This may be done with a simple third-party **IBP_copy()** call. When finished, the agent updates the information in the IBP name file and deletes the initial data file. When the receiver presents the agent with the name, the agent returns the read and manage capabilities of the data file, and the receiver downloads it and decrements the read capability if desired.

The decision to use IBP for the name file was for flexibility. With this structure, it is possible to have multiple agents manage the transport, to have agents restart upon failure without losing information, and even to have the receiver find the location of the file without contacting the agent.

The allocation of the data files and the name files are performed using IBP's time-limited allocation policy. Currently, the time limit default is 10 days, and this may be adjusted by the sender. As discussed above, the time-limited allocation is necessary for storage owners to be able to reclaim their storage resources from the network. It also solves the lost pointer problem in the case of agent failure or lost email. Additionally, it frees the sender and receiver from having to explicitly delete the file. One ramification of this is that there is a new failure mode for mail – time-limit expiration on the data file. There are several ways to address this mode – send warning mail or error reporting mail back to the sender, send warning mail to the receiver, allow the sender to extend the time limit, or simply delete the file.

3.3 The Current Mail Interface

There are currently two sender interfaces to IBP-Mail. The first is a command-line Unix program that works like `mpack`[1]. for sending MIME mail attachments. It may be obtained on the web at `http://www.cs.utk.edu/~el-wasif/ibp-mail`. The second interface is web-based. Senders may use this interface by pointing their web-browsers to the URL `http://www.cs.utk.edu/~el-wasif/cgi-bin/ibp-mail.cgi`. This is a CGI enabled web page that allows the sender to specify a file name, plus a message, and have the message/file sent to a set of recipients using IBP-Mail. File uploading in HTTP requires the file to go to the server of the web page; therefore the web interface requires that all attachments be uploaded to Tennessee's web server and the web server then acts as a proxy sender of the message. This makes this interface less efficient than the command-line version, since all files must go through the server at Tennessee. We are working to have the web server redirect the sender to a closer web server in a manner analogous to the receiving protocol described below. We will also write a Windows 95/98/NT sending program in the near future.

The message that the receiver gets has a MIME-encoded URL that points to the IBP-NT agent and contains the capabilities of the name file. The IBP-NT agent must be running on a host that contains a web server, so that it can resolve this URL. When the receiver clicks on the URL, the IBP-NT agent is contacted through a CGI script on its web server, and it returns a web page with an HTTP redirection link. This link contains a URL for the IBP server that holds the data file. The receiver's web browser, upon receiving the redirection, automatically attempts to get this new URL. We have written a small HTTP gateway daemon that can run on IBP hosts so that these redirection requests may be served directly by IBP. In this way, the receiver's browser downloads the data file directly from IBP without any knowledge of IBP.

An alternative to this interface would be to provide a standalone application that retrieves IBP-Mail files, which could be launched as a special MIME type by MIME-enabled mailers. Or we could modify a mailer such as Pine or MH to recognize IBP-Mail attachments and download the files accordingly. We chose the web-based alternative so that users can receive IBP-Mail files without needing to install any code; indeed, they do not even have to understand what IBP-Mail is.

[1] Mpack may be obtained at `ftp://ftp.andrew.cmu.edu/pub/mpack/`

4 A Performance Example

As a simple motivating example, suppose a user in the Tennessee computer science department would like to mail her colleague at Princeton a 9 MB data file. Below we detail the results of attempting various ways of mailing the file. These results were obtained during a single working day. All numbers are the average of at least three tests.

#1: Mailing the file

Attempts to mail the file, using either MIME or `uuencode` fail because Princeton's mail daemon, like most, restricts the size of incoming mail. In Princeton's case messages over 6,000,000 bytes are rejected, meaning that this file, which becomes 12.2 MB as a MIME attachment and 12.4 MB with `uuencode` cannot be mailed. The file can be split into three parts and mailed, but this defeats the purpose of limiting the size of incoming mail.

#2: Using FTP

Here the user uploads her file to Tennessee's anonymous FTP server, `cs.utk.edu`, which is a relatively low-end SPARCstation-2 running SunOS 4.1.4. The upload takes 2 minutes and 36 seconds using Sun NFS. The user at Princeton, running on a DEC Alpha, can download the file using anonymous FTP in three minutes.

#3: Using HTTP

In this case, the user sets a soft link to the file in her World Wide Web home directory. This takes no time. Tennessee's main HTTP server is a much higher end machine than its anonymous FTP server: a SPARCserver-1000 running Solaris 5.5.1. The user at Princeton can download the file using any web browser (`lynx` was used for the timing) in 29 seconds.

#4: Uploading the file

Uploading the file is not feasible, because the user at Tennessee does not have access to any writable FTP server at Princeton.

#5: IBP-Mail

To test the performance of IBP-Mail, we set up two IBP servers: one the I2-DSI machine located at Tennessee, and another DEC Alpha at Princeton. Storing the file to I2-DSI machine takes one second. Performing a remote copy from the I2-DSI IBP server to the Princeton IBP server takes 46 seconds. Retrieving the file at Princeton takes 10 seconds from the IBP server at Princeton, and 57 seconds from the I2-DSI machine.

Comparison

A comparison of the solutions to mailing the file is tabulated in Table 1. The times are given in *min:sec*. In terms of time spent by the sender, uploading the file to the HTTP server and using IBP-Mail are rougly equal. In terms of time spent by the receiver, if the receiver reads the mail as soon as it is sent and tries to retrieve the file, the HTTP transaction is quicker than downloading from the I2-DSI machine. However, if the receiver does not read the mail until the file has been moved to Princeton (roughly a minute after it has been sent), then it takes only 10 seconds to download. Thus, in the typical case, where the receiver does not read mail instantly, IBP-Mail outperforms uploading the mail to the HTTP server. Additionally, in this typical case, there will be less variability in the performance of the download, since the file is on a local network with IBP-Mail, instead of on a remote HTTP server.

	FTP	HTTP	IBP-Mail
Time to send	2:36	0:00	0:01
Time to retrieve instantly	3:00	0:27	0:57
Time to retrieve later	3:00	0:27	0:10
Manual deletion required	Yes	Yes	No
Data protected	No	No	Yes

Table 1: Comparison of mailing solutions

The last two lines of Table 1 pertain to ease-of-use and security. With FTP and HTTP, the sender must manually delete the file when she is certain that the receiver has retrieved it. With IBP-Mail, the file is automatically deleted either by the receiver or by the time-limit on the file. The "data protected" line concerns who can actually read the file. With anonymous FTP, anyone who can connect to the server may discover and read the file. With HTTP, anyone can read the file, but discovery may be more difficult if the sender arranges that outside users cannot list the directory containing the file. However, since the directory must be world-readable for the web server to serve it, all users at Tennessee are able to find and read the file. With IBP-Mail, anyone can read the file, but only if they have access to the read capability. Since there are no directory semantics exported by IBP, the likelihood of someone discovering the read capability is extremely small. Only the owner of the IBP server process has access to the file. Currently, the IBP servers do not encode their data cryptographically. Eventually, this will be an allocation option for IBP so that security and performance may be traded off.

Therefore, in this example, IBP-Mail is an attractive solution to mailing large data files in terms of performance, ease-of-use and security. Of course, this is just one scenario of many. However, it does demonstrate the plausibility of the idea in a real-world setting.

5 Experimentation and Extensions

IBP-Mail is the first application to use IBP and the L-Bone. We are planning to perform further experimentation with IBP-Mail to learn more about how storage can affect networking performance in applications such as IBP-Mail that are asynchronous in nature. First, we plan to experiment with server selection policies. Given the structure of IBP-Mail, server selection can be static, based on permanent network proximity metrics, user hints, or dynamically measured and/or predicted network performance metrics. We intend to test all of these policies to determine the payoff for adding better performance monitoring and prediction mechanisms into IBP-Mail. We plan to use the Network Weather Service [19] as the base mechanism for performance and prediction.

For extremely large mail files, it may not be possible to store the file on any one IBP server. One simple extension of IBP-Mail would be to break the file into fragments and store them at different IBP servers, perhaps with one or more encoded fragments that would allow the system to lose one or more of the mail fragments without losing the entire file [10].

While IBP-Mail makes use of storage in the network, it can also benefit from computational elements in the network. For example, compressing mail messages before sending them between IBP servers may improve performance. Additionally, instead of delivering an entire data file to a receiver, it may be advantageous to mine it instead. For example, a condensed snapshot of a large image or a low resolution version of a video file may be suffice for some content-exchange scenarios. We will experiment with providing such functionalities within the context of IBP in the future.

A situation where a structure like IBP-Mail should prove useful is in remote computation environments. For example, NetSolve is a brokered RPC system that allows simple clients (such as Matlab or Mathematica) to outsource large computations to more suitable computational elements [5]. Some applications make use of NetSolve to perform multiple similar computations in parallel, coalescing the results as they complete. Currently, the collection of results is synchronous and performed by the client program. An approach similar to IBP-Mail can be used to get results from the computational servers back to the client in an asynchronous manner, or to coalesce results in similar remote locations remotely by storing them in IBP servers near the remote comptuation servers. In collaboration with Dongarra (Tennessee), Berman and Casanova (University of San Diego), we are exploring the integration of IBP, NetSolve and Application Level Scheduling (AppLeS) [2] to improve performance, functionality and resource utilization in brokered RPC systems.

6 Related Work

Mail systems have been around for decades; however most mail systems work within the structure of SMTP mail. The Grapevine project [3, 17] developed at Xerox PARC in the early 1980's was a distributed mail handling system that provided functionality beyond simple point-to-point delivery of email messages. Grapevine, however, required a substantial administrative overhead as it handled many features outside mail transport (e.g. user registration and authentication). It was assumed that servers participating in a Grapevine system were under control of a single administrative entity (or several closely cooperating entities). It is extremely difficult in today's Internet to satisfy this requirement. IBP-Mail assumes no administrative intervention beyond setting up the various servers that are part of the system, and requires no special priviliges for its setup and/or operation.

A more recent project is Porcupine from the University of Washington [15,16]. Porcupine uses clusters of computers to handle mail traffic that can be as much as a billion messages a day. As such, Porcupine is appropriate for organizations that have huge amounts of incoming mail. Like Grapevine, Porcupine works within a single administrative domain, and does not address mail between arbitrary and perhaps unrelated entities on the Internet.

The *External-Body* subtype [9] of the MIME protocol may be used to affect the delivery of an email attachment through a receiver-initiated action. Through the use of this subtype, the sender specifies an access protocol that is recognized by the receiver and the proper protocol handlers are used to access the actual body of the message. The use of this technique requires the sender to pre-stage the message body at a storage site accessible through an access protocol (e.g. FTP or HTTP). While this technique works well for objects that are already accessible through the protocol in question, it requires additional administrative overhead for those objects that are intended to be exchanged only between the sender and the receiver. This overhead stems from the need to limit access to the object to the receiver(s), the need to detect when the object has been retrieved by the receiver to revoke it's protocol access (or remove it physically from permanent storage), or the need to stage the object at a protocol-enabled server.

`Click2send.com` [6] is a web-based company that allows users to send large files to other users by uploading them to a "deposit box" located at `click2send.com` and then sending a pointer to a recipient. Like IBP-Mail, `click2send.com` provides users with a free writable network storage depot for sending large files. The two have similar web interfaces, and like IBP-Mail, `click2send.com` files have a time limit after which they are deleted. The main difference between `click2send.com` and IBP-Mail is the use of multiple storage depots distributed geographically, so that the performance of both sending and receiving may be optimized. For reference, the performance of `click2send.com` on the example in Section 4 is 2 minutes, 44 seconds to deposit the file from Tennessee, and 1 minute, 20 seconds to download it from Princeton (`click2send.com` is located in Mountain View, California).

7 Conclusion

The current model for electronic mail message transfer is to move data completely from the sender to the receiver before the recipient is allowed to request access the message. Using the Internet Backplace Protocol, we have implemented a more flexible data transport model in which a message identifier is transmitted immediately to the receiver while the message data itself traverses a network of storage depots between sender and receiver. When the identifier is dereferenced by the receiver, the "nearest" copy of the data is located and fetched automatically. This more flexible approach avoids overrunning the receiver's spooling space and permits the mail system to optimize message routing.

The current IBP-Mail prototype is implemented transparently using web-enabled mailers and the I2-DSI storage backbone which hosts a pool of IBP servers. Although our model of mail delivery relies upon the availability of distributed application controllable storage, we believe such storage services will become more widely deployed as the Internet evolves. As part of our future work, we plan to study ways in which the transfer model that we have developed for IBP-Mail can be applied to other distributed network applications. Based on our experience developing the system, we believe that IBP and distributed storage will prove useful in Computational Grid and Internet computing settings.

8 Acknowledgements

This material is based upon work supported by the National Science Foundation under grants EIA-9975015, ACI-9876895 and CCR-9703390, and by the Department of Energy under grant DE-FC0299ER25396. The authors thank Terry Moore and Martin Swany for in-depth discussions, and Amy Phillipson for helping with the initial idea.

References

[1] 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.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao and. Application-level scheduling on distributed heterogeneous networks. In *Supercomputing '96*, November 1996.

[3] A. D. Birrell, R. Levin, R. M. Needham, and M. D. Shroeder. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[4] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies. IETF RFC 1521 (`http://www.ietf.org/rfc/rfc1521.txt`), September 1993.

[5] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[6] Click2send.com, Inc. Click2send. `http://www.click2send.com/`, 1999.

[7] R. Fielding, J. Gettys, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. IETF RFC 2068 (`http://www.ietf.org/rfc/rfc2068.txt`), January 1997.

[8] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) part one: Format of internet message bodies. IETF RFC 2045 (`http://www.ietf.org/rfc/rfc2045.txt`), November 1996.

[9] N. Freed and K. Moore. Definition of the URL MIME External-Body Access-Type. IETF RFC 2017 (`http://www.ietf.org/rfc/rfc2017.txt`), October 1996.

[10] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[11] J. S. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the network. In *NetStore '99: Network Storage Symposium*. Internet2, October 1999.

[12] J. Postel and J. Reynolds. File transfer protocol (FTP). IETF RFC 959 (`http://www.ietf.org/rfc/rfc0959.txt`), October 1985.

[13] J. B. Postel. Simple Mail Transfer Protocol – SMTP. IETF RFC 821 (`http://www.ietf.org/rfc/rfc0821.txt`), August 1992.

[14] P. Resnick and A. Walker. The text/enriched MIME content-type. IETF RFC 1896 (`http://www.ietf.org/rfc/rfc1896.txt`), February 1996.

[15] Y. Saito, B. N. Bershad, and H. M. Levy. Manageability, availability and performance in Porcupine: a highly-scalable cluster-based mail service. In *17th ACM Symposium on Operating Systems Principles*, December 1999.

[16] Y. Saito, E. Hoffman, B. Bershad, H. Levy, and D. Becker. The porcupine scalable mail server. In *8th ACM SIGOPS European Workshop*, Sintra, Portugal, September 1998.

[17] M. Schroeder, A. D. Birrell, and F. M. Needham. Experience with Grapevine: The growth of a distributed system. *ACM Transactions on Computer Systems*, 2(1):3–23, February 1984.

[18] M. Swany. Network proximity resolution with SONAR and SonarDNS. Technical Report CS-99-429, University of Tennessee, January 1999.

[19] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 1999.

[20] W. Yeong, T. Howes, and S. Kille. Lightweight directory access protocol. IETF RFC 1777 (`http://www.ietf.org/rfc/rfc1777.txt`), March 1995.