Scalable Sharing of Wide Area Storage Resources

Micah Beck, Terry Moore, James Plank
{mbeck,tmoore,plank}@cs.utk.edu

Department of Computer Science
University of Tennessee
1122 Volunteer Blvd.
Knoxville, TN 37996-3450
Corresponding Author: Micah Beck, (865) 974-3548

1. Introduction

Over the past two years a combination of circumstances has fueled intense efforts to rethink the integration of IP networking and mass storage. Chief among these facts are the universal acceptance of Internet technologies for network infrastructure, the relentless acceleration of the Ethernet substrate, and the ongoing, exponential rise in the amount of data that must be moved and managed. Taken together they present the network storage community with a compelling opportunity to address its traditional goals while scaling up the size and geographical reach of its systems and scaling down their complexity and total cost of ownership. But although a new synthesis of Internet and storage technologies seems inevitable under these conditions, what is less evident is that this potential new synthesis may take one or both of two distinctly different forms.

One likely form is rooted in the historic goals and technologies of the *Storage Networking* community. Without changing the traditional model of network storage, which attempts to create a distributed pool of storage resources while preserving the strong guarantees of processor-attached storage, various new approaches (e.g. iSCSI, iFCP, FCIP, and SoIP) are weaving IP technology into the interconnection fabric that knits together the storage devices composing the pool. Although this approach may significantly reduce cost of ownership and increase interoperability, it does not change the classic model of storage. Therefore there is reason to doubt that it can provide improved support to applications, such as multimedia distribution and telecollaboration, that need to share resources that also scale to large communities in the wide area, across corporate, administrative, and network boundaries. The traditional model of storage was not designed for the kind of open, unbrokered access that IP networking pioneered, and translating the strong semantics that are possible in local services to the wide area has always been problematic [10]. Standard storage services prove to be no exception.

A second paradigm for synthesizing Internet and storage technologies starts from the premiss that future wide area applications will require a different fundamental model of storage. It comes from an emerging research field called *Logistical Networking* [2], which explores the possibility of putting storage *into the network* in order to support advanced multimedia and high performance distributed computing applications of all types. As the name suggests, Logistical Networking adopts a point of view that treats data transmission and storage as part of a unified framework, just as military or industrial logistics view transportation lines and storage depots as coordinate elements of a single infrastructure. To explore this perspective, we have created a primitive abstraction of network storage, the *Internet Backplane Protocol (IBP)*[7], as the lowest network layer in a new "storage stack." Modeled on IP datagram service, IBP implements a model of storage with semantics that are *as weak as possible*. The goal is to maximize flexible shareability of storage resources in a way that *can scale to the wide-area*.

Thus we have two approaches to synthesizing IP and storage technologies, one that changes the underlying model of storage and another that doesn't. Whatever the novelty value of the unorthodox

This work is supported by the National Science Foundation Next Generation Software Program under grant # EIA-9975015, the Department of Energy Scientific Discovery through Advanced Computing Program under grant # DE-FC02-01ER25465, and by the National Science Foundation Internet Technologies Program under grant # ANI-9980203.

paradigm might be, it currently exists only as research papers, open source software, and a testbed deployment. Given the fact that the Storage Networking approach to IP/Storage integration seems to be rapidly becoming a technological and industry juggernaut, it might, therefore, seem doubtful that the Logistical Networking approach could have any effect on the course that such integration will take. On the other hand, the very drive to augment or replace Storage Networking's interconnection fabric with IP-based technology illustrates a pattern that the rise of the Internet has repeatedly exhibited: approaches that can scale to the wide-area tend to change, and ultimately supplant, those that cannot. In this paper we describe the Logistical Networking model of network storage, which we believe will, if successful, affect the synthesis of IP and Storage Networking in a similar way.

2. IBP: The network layer of the storage stack

In the context of Storage Networking, "IP/Storage integration" means putting IP networking into the interconnection fabric (i.e. into the data transmission substrate) that underlies the storage pool. For Logistical Networking, on the other hand, IP/Storage integration means *putting storage into the network infrastructure itself*, creating a *shared resource fabric* that exposes storage resources for general use in the same way that the Internet now exposes transmission bandwidth for shared use. To create a resource fabric of this kind that can also scale, we set out to define a new *storage stack* using a bottom-up and layered design approach that adheres to the same end-to-end principles that have guided Internet engineering for two decades [9]. According to this philosophy, the key to achieving flexibility and scalability lies in defining the right basic abstraction of the physical resource to be shared at the lowest levels of the stack. For Logistical Networking the *Internet Backplane Protocol (IBP)* plays this role.

IBP is the lowest layer of the storage stack that is globally accessible from the network (figure 1). To provide an ideal resource fabric for Logistical Networking, it must supply an abstraction of *access layer resources* (i.e. storage services at the local level) that has "network transparency"[8]. This means it must satisfy the following two requirements:

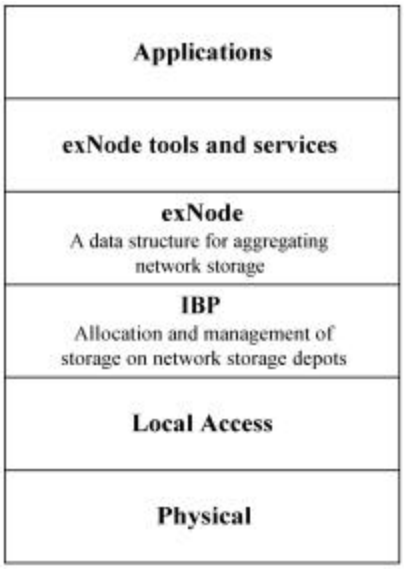

Figure 1: The network storage stack for Logistical Networking.

Expose underlying storage resources in order to maximize freedom at higher levels — The abstraction should create a mechanism that implements only the most indispensable and common functions necessary to make the storage usable *per se*, leaving it otherwise as primitive as it can be; all stronger functions must be built on top of this primitive layer. The goal of providing essential functionality while keeping the semantics of this layer as weak as possible is to expose the underlying resources to the broadest range of purposes at higher layers, and thereby foster ubiquitous deployment and free developers to innovate.

Enable scalable Internet-style resource sharing — The abstraction must mask enough of the peculiarities of the access layer resource (e.g. fixed block size, differing failure modes, and local addressing schemes) to enable lightweight allocations of those resources to be made by any participant in the network for their limited use and regardless of who owns them.

To implement this strategy we followed the IP paradigm and modeled the design of IBP on the design of IP datagram delivery. IP datagram service is based on packet delivery at the link level, but with more powerful and abstract features that allow it to scale globally. Its leading feature is the independence of IP datagrams from the attributes of the particular link layer, which is established as follows:

- Aggregation of link layer packets masks its limits on packet size;

- Fault detection with a single, simple failure model (faulty datagrams are dropped) masks the variety of different failure modes;
- Global addressing masks the difference between local area network addressing schemes and masks the local network's reconfiguration.

This higher level of abstraction allows a uniform IP model to be applied to network resources globally, which is crucial to creating the most important difference between link layer packet delivery and IP datagram service: *any participant in a routed IP network can make use of any link layer connection in the network regardless of who owns it.* Routers aggregate individual link layer connections to create a global communication service. This IP-based aggregation of locally provisioned, link layer resources for the common purpose of universal connectivity constitutes the form of sharing that has made the Internet the foundation for a global information infrastructure.

IBP is designed to enable the scalable, relatively unbrokered sharing of storage resources within a community in much the same manner. Just as IP is a more abstract service based on link-layer datagram delivery, IBP is a more abstract service based on blocks of data (on disk, tape or other media) that are managed as "byte arrays." The independence of IBP byte arrays from the attributes of the particular access layer (which is our term for storage service at the local level) is established as follows:

- Aggregation of access layer blocks masks the fixed block size;
- Fault detection with a very simple failure model (faulty byte arrays are discarded) masks the variety of different failure modes;
- Global addressing based on global IP addresses masks the difference between access layer addressing schemes.

This higher level of the byte array abstraction allows a uniform IBP model to be applied to storage resources globally, which is essential to creating the most important difference between access layer block storage and IBP byte array service: *Any participant in an IBP network can make use of any access layer storage resource in the network regardless of who owns it.* The use of IP networking to access IBP storage resources creates a global storage service.

Whatever the strengths of this application of the IP paradigm, however, it leads directly to two problems. First, in the case of storage, the chronic vulnerability of IP networks to Denial of Use (DoU) attacks is greatly amplified. The free sharing of communication within a routed IP network leaves every local network open to being overwhelmed by traffic from the wide area network, and consequently open to the unfortunate possibility of DoU from the network. While DoU attacks in the Internet can be detected and corrected, they cannot be effectively avoided. Yet this problem is not debilitating for two reasons: on the one hand, each datagram sent over a link uses only a tiny portion of the capacity of that link, so that DoU attacks require constant sending from multiple sources; on the other hand, monopolizing remote communication resources cannot profit the attacker in any way, it can only harm the victim. Unfortunately neither of these factors hold true for access layer storage resources. Once a data block is written to a storage medium, it occupies that portion of the medium until it is deallocated, so no constant sending is required. Moreover it is clear that monopolizing remote storage resources can be very profitable for an attacker and his applications.

The second problem with sharing storage network-style is that the classic definition of a storage service is based on processor-attached storage, so it includes strong semantics (near-perfect reliability and availability) that are difficult to implement in the wide area network. Even with Storage Networking technologies, which are used in "storage area" or local area networks, these strong semantics can be difficult to implement and are a common cause of error conditions. When extended to the wide area, it has so far proved impossible to support such strong guarantees for storage access, but then problems with strong service semantics in the wide area are not unique to storage systems [10]. Whether or not integrated IP and Storage Networking technologies can make progress on this front remains to be seen. Logistical Networking takes a different approach.

We address both of these issues through special characteristics of the way IBP allocates storage:

- Allocations of storage in IBP can be time limited. When the lease on an allocation expires, the storage resource can be reused and all data structures associated with it can be deleted. An IBP allocation can be refused by a storage resource in response to over-allocation, much as routers can drop packets, and such "admission decisions" can be based on both size and duration. Forcing time limits puts transience into storage allocation, giving it some of the fluidity of datagram delivery.
- The semantics of IBP storage allocation are weaker than the typical storage service. Chosen to model storage accessed over the network, it is assumed that an IBP storage resource can be transiently unavailable. Since the user of remote storage resources is depending on so many uncontrolled remote variables, it may be necessary to assume that storage can be permanently lost. Thus, IBP is a "best effort" storage service. To encourage the sharing of idle resources, IBP even supports "volatile" storage allocation semantics, where allocated storage can be revoked at any time. In all cases such weak semantics mean that the level of service must be characterized statistically.

IBP storage resources are managed by “depots,” or servers, on which clients perform remote storage operations. As shown in the Table 1 below, the IBP client calls fall into three different groups:

Storage Management	Data Transfer	Depot Management
`IBP_allocate, IBP_manage`	`IBP_store, IBP_load IBP_copy, IBP_mcopy`	`IBP_status`

Table 1: IBP API calls

The **`IBP_allocate`** function is the most important element. **`IBP_allocate`** is used to allocate a byte array at an IBP depot, specifying the size, duration (permanent or time limited) and other attributes. A chief design feature is the use of capabilities (cryptographically secure passwords). A successful **`IBP_allocate`** returns a set of three capabilities: one for reading, one for writing, and one for management of the allocated byte array. A more detailed account of the API and its other functions is available [7] online at (http://loci.cs.utk.edu/ibp/documents/). A description of the status of the current software the implements the IBP client, servers, and protocol is available at (http://loci.cs.utk.edu/ibp/software).

3. The exNode: A data structure for the flexible aggregation of network storage

From the point of view of the Storage Networking community, it is likely that one of the most striking (not to say shocking) features of the Logistical Networking storage stack is the way it appears to simply jettison the well known methods of usage for local area storage, viz. files systems, databases, and VM mapping. These familiar abstractions can be supported in the logistical paradigm, but that support must conform to its “exposed-resource” design principles. According to these principles implementing abstractions with strong properties — reliability, fast access, unbounded allocation, unbounded duration, etc.— involves creating a construct at a higher layer that *aggregates* more primitive IBP byte-arrays below it, where these byte arrays are often distributed at multiple locations. For example, caching requires that data be held in a home site, but temporary copies are made at various remote sites. Similarly, replication requires that multiple copies of data exist in various locations for purposes of performance and fault-tolerance. More advanced logistical applications require that data be explicitly routed through the network, and thus may have many “homes” throughout their lifetime.

To apply the principle of aggregation to exposed storage services, however, it is necessary *to maintain state that represents such an aggregation of storage allocations*, just as sequence numbers and timers are maintained to keep track of the state of a TCP session. Fortunately there is a traditional, well-understood model to follow in representing the state of aggregate storage allocations. In the Unix file system, the data structure used to implement aggregation of underlying disk blocks is the *inode* (*intermediate node*). Under Unix, a file is implemented as a tree of disk blocks with data blocks at the leaves. The intermediate nodes of this tree are the inodes, which are themselves stored on disk. The Unix inode implements only the aggregation of disk blocks within a single disk volume to create large files; other strong properties are sometimes implemented through aggregation at a *lower level* [3] or through modifications to the file system or additional software layers that make redundant allocations and maintain additional state [5, 11].

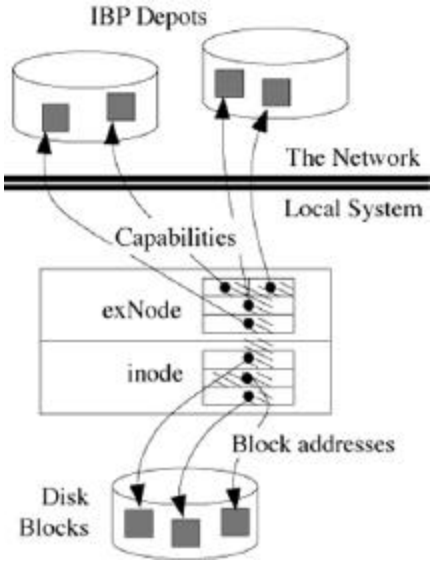

Figure 2: The exNode compared to a Unix inode.

Following the example of the inode, we have chosen to implement a single generalized data structure, which we call an *external node*, or *exNode*, in order to manage aggregate allocations that can be used in implementing network storage with many different strong semantic properties [1]. Rather than aggregating blocks on a single disk volume, the exNode aggregates byte arrays in IBP depots to form something like a file, with the byte arrays acting as disk blocks. We say "something like a file" because when an exNode uses IBP storage allocations, the time-limited or volatile nature of those allocations gives it a transient quality that files normally should not have. Two major differences between exNodes and inodes are that the IBP buffers may be of any size, and the extents may overlap and be replicated. But the key point about the design of the exNode is that it has allowed us to create storage abstractions with stronger properties, such as a network file, which can be layered over IBP-based storage in a way that is completely consistent with the exposed resource approach.

Since we intend to use the exNode to implement abstractions that can support a number of different applications, we have chosen to express the exNode concretely as an encoding of storage resources (e.g. IBP capabilities) and associated metadata in XML. Like IBP capabilities, these serializations may be passed from client to client, allowing a great degree of flexibility and sharing of network storage. If the exNode is placed in a directory, the file it implements can be imbedded in a namespace. But if the exNode is sent as a mail attachment, there need not be a canonical location for it. The use of the exNode by varying applications will provide interoperability similar to being attached to the same network file system. The exNode metadata must be capable of expressing at least the following relationships between the file it implements and the storage resources that constitute the data component of the file's state:

- The portion of the file extent implemented by a particular resource (starting offset and ending offset in bytes)
- The service attributes of each constituent storage resource (e.g. reliability and performance metrics, duration)
- The total set of storage resources which implement the file and the aggregating function (e.g. simple union, parity storage scheme)

One key role for the exNode, then, is to provide interoperability between heterogeneous nodes. For that reason we have chosen not to specify it as a language-specific data structure, but as an abstract data type with an XML serialization. The exNode data structure is the basis for interoperability within the Logistical Networking API, and the XML serialization is the basis of interoperability between storage nodes in the resource fabric.

More broadly, we are using the exNode as the basis for a set of generic tools for implementing files and other storage abstractions with a wide range of characteristics. The important elements to be developed in this regard are libraries that provide mechanisms for manipulating the exNode data structure in order to implement generic requirements such as large size (through fragmentation), fast access (through caching), and reliability (through replication). Applications requiring these characteristics should be able to obtain them from the aggregate even without having individual IBP depots that implement those specific characteristics available. Simply using the APIs should be sufficient to aggregate resources that are available for use somewhere on the network.

We have developed a prototype of a simple but useful set of tools that make it easy to employ and aggregate network storage in the form of IBP depots in robust and powerful ways. Again, these tools build on the lower levels of the network storage stack: IBP is the basic storage substrate, serving time-limited storage buffers to its clients. The *Logistical Backbone* (*L-Bone*), which is a testbed for Logistical Networking we describe below, is employed to find IBP depots matching various properties (size, duration, network proximity), and to do live proximity detection measurements using the Network Weather Service [12].

The tools are as follows:

`xnd_upload`: This tool uploads a local file into the network and returns an exNode for the upload. This upload may be parameterized in a variety of ways. For example, the user may partition the file into multiple blocks (i.e. stripe the file) and these blocks may be replicated on multiple IBP servers for fault-tolerance and/or proximity reasons. Moreover, the user may specify proximity metrics for the upload, so the blocks have a certain network location.

`xnd_download`: This takes an exNode as input, and downloads the file that it represents into a local file. This involves coalescing the replicated fragments of the file, and must deal with the fact that some fragments may be closer to the client than others, and some may not be available (due to time limits, volatility, and standard network failures). **`xnd_download`** may only download portions of the file, and if desired, it can operate in a streaming fashion, so that the client only has to consume small, discrete portions at a time.

`xnd_refresh`: This takes an exNode as input, and updates time limits of the IBP buffers that compose the file. An extention to **`xnd_refresh`** will be a persistent.service called the **`xnd_warmer`**, which will periodically call **`xnd_refresh`** (and perhaps **`xnd_augment`** and **`xnd_trim`**), so that a file composed of time-limited IBP buffers can stay alive indefinitely.

`xnd_augment`: This takes an exNode as input, adds more replicas to it (or to parts of it), and returns an updated exNode. Like **`xnd_upload`**, these replicas may have a specified network proximity.

`xnd_trim`: This takes an exNode, deletes specified fragments, and returns a new exNode. These fragments may be specified individually, or they may be specified to be those that represent expired IBP allocations. **`xnd_augment`** and **`xnd_trim`** may be combined to effect a routing of a file from one network location to another — first it is augmented so that it has replicas near the desired location, then it is trimmed so that the old replicas are deleted.

Note, the exNode tools are much more powerful than IBP capabilities, since they allow users to aggregate network storage for various reasons: They can create files of *extremely large capacity* from smaller IBP allocations; it is not hard to visualize files that are tens of gigabytes in size, split up and scattered around the network. They can *stripe* the files, breaking files into small pieces that can be downloaded simultaneously from multiple IBP depots, providing better performance than downloading from a single source. These tools can enable r*eplication for caching*, storing files in multiple locations so that the performance of downloading may be improved by downloading the closest copy.

They can used to implement *replication for fault-tolerance*, storing files in multiple locations so that the act of downloading may succeed even if many of the copies are unavailable; by breaking the file up into blocks and storing error correcting blocks calculated from the original blocks (based on parity as in RAID systems [3]or on Reed-Solomon coding [6], downloads can be robust to even more complex failure scenarios. Finally, as presented in the explanation of **`xnd_trim`** above, they make *routing* possible, moving the file to one node or another for the purposes of scheduling, improved resource conditions, etc. Thus this small set of exNode tools provides the basis for a runtime framework that will enable applications to gain the benefits of a well-provisioned storage resource fabric.

4. Prototype applications on the Logistical Backbone

The highest level of our network stack (Figure 1), and in some respects the most important level, contains the applications that benefit from the kind of shareable, wide area storage services Logistical Networking can provide. Network multimedia applications offer good examples of such applications, and below we describe an application we are experimenting with on our Logistical Networking testbed, which we call the *Logistical Backbone* (*L-Bone*).

The L-Bone is a deployment of IBP depots in the wide area network and provides a distributed runtime service that allows clients to perform IBP depot discovery. IBP depots register themselves with the L-Bone, and clients may then query the L-Bone for depots that have various characteristics, including minimum storage capacity and duration requirements, and basic proximity requirements. For example, clients may request an ordered list of depots that are close to a specified city, airport, US zipcode, or network host. The majority of IBP depots registered with the L-Bone are at Tennessee, but there are also depots in Texas, North Carolina, California, and other locations. Once the client has a list of IBP depots, it may then request that the L-Bone use the NWS to order those depots according to bandwidth predictions using live networking data. Thus, while IBP gives clients access to remote storage resources, it has no features to aid the client in figuring out which storage resource to employ. The L-Bone's job is to provide clients with those features. For the L-Bone API, and its up-to-date composition, see \url{http:///www.cs.utk.edu/lbone}.

4.1 IBP-ster: Multimedia delivery on the wide area

The rise of the Web and the development of compression techniques such as MPEG-1 [4] have transformed the Internet into a wide-area multimedia delivery substrate. Despite the advances of streaming technology and peer-to-peer content distribution systems, the storage of multimedia data has been downright archaic. Music files encoded with MP3 compression consume roughly 1MB per minute, while “Video CD” files stored with MPEG-1 compression consume roughly 10 MB per minute. As such, multimedia files are extremely large. However, they are typically stored as single files in the file system of one computer. When delivered across the network, these files are either moved in their entirety via FTP or HTTP, or streamed in smaller chunks. Caching of these large files is disabled by the well-known web caching entities (e.g.Akamai) because they are too large. Although peer-to-peer replication entities (e.g. Napster and Gnutella) effect a kind of caching, finding and downloading the closest copy is a black art. Inktomi markets a product (``Traffic Core") that caches streaming media, which is more efficient than caching whole files, but it is still limited, and must be custom-tailored to the stream-provider's application.

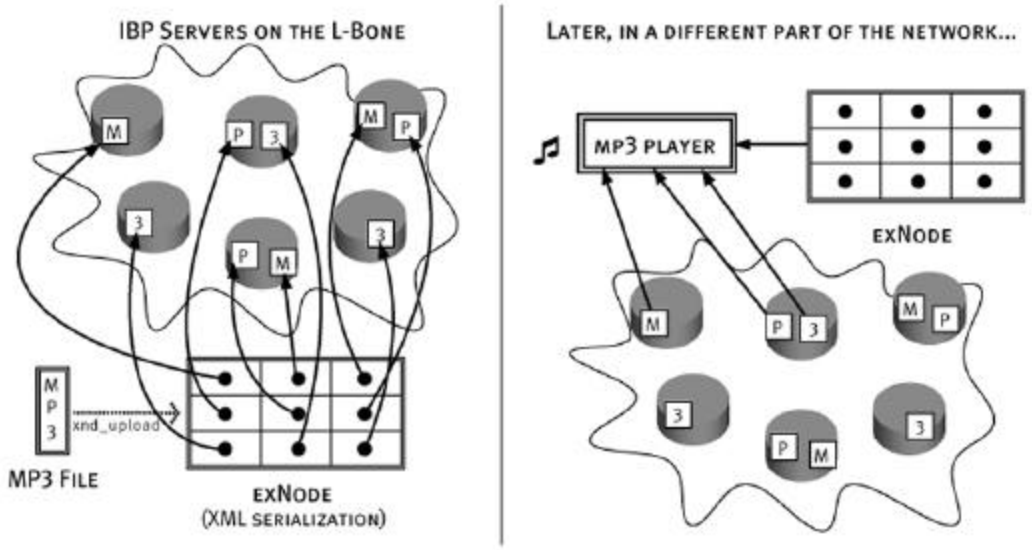

Figure 4: IBPster in use: (a) the user uploads an MP3 file, specifying three blocks, each replicated three times. (b) the user plays the MP3 file using an MP3 player at a different location. This MP3 player reads the exNode, and plays the MP3 in a streaming fashion, downloading blocks from the closest IBP depots.

IBPster is a project we developed to demonstrate the capabilities of Logistical Networking. The goal of IBPster is to employ network storage to perform better and more flexible multimedia delivery. IBPster has been built on primitive versions of the exNode tools, and is an excellent proof of concept for the power of these tools. With IBPster, a user may upload a multimedia file into IBP buffers on the L-Bone. This is done with a **`xnd_upload`**, which returns an exNode. The upload may be done so that the file is split up into blocks and replicated, and stored on IBP servers that are close to a certain network neighborhood. As an example, in Figure 4(a), an MP3 file is uploaded into IBP buffers, splitting the file into three blocks, and replicating each block three times.

Since exNode files are XML serializations, they may be transported easily by mail, FTP, HTTP, and so on. This means that whenever *and wherever* the user wants to listen to his MP3 file, he may call **`xnd_download`** to download the file, bringing in the pieces in order from whichever IBP servers are closest. Moreover, if the user has access to an IBPster MP3 player, then the *player* can play the exNode file by performing a streaming download, as depicted in Figure 4(b). As before, this process downloads the closest blocks of the file, using the L-Bone's tools for proximity detection, based on the Network Weather Service.

Obviously, the IBPster prototype can expand in many dimensions. For example, the technique does not have to be limited to MP3 files — any kind of file may be uploaded and downloaded. Moreover, once a file has been uploaded, it may be manipulated in IBP space in many ways: the allocation durations may be extended; it may be further fragmented and replicated; it may be routed so that it will be close to certain network locations, etc. We anticipate that the IBPster project will not only help us develop the exNode runtime suite of tools and services, but it will also help us develop the logistical scheduling policies that may be useful to a variety of applications, and the interfaces that allow users to specify those policies. When the user wants to download an MP3 file, he has very specific performance needs — his player should only start playing the file when it is assured that the download will proceed at a pace fast enough to keep the file playing. Obviously, the scheduler can take this into account when devising a downloading schedule, and when performing the upload, so long as the scheduler has some notion of where the download will come from.

5. Conclusion

In this paper we have presented *Logistical Networking*, a new paradigm for the sharing of storage in network communities, even in the wide area. We have contrasted Logistical Networking to *Storage Networking*, the approach that currently dominates the storage industry. The difference in the two paradigms can be seen as a convergence towards a middle point from two extremes:

- Storage Networking views the network as a generalization of the bus and I/O channel interfaces that traditionally tie hosts to servers.
- Logistical Networking views storage as an additional resource that can be served to the network community in an unbrokered manner.

The technical strength of Storage Networking lies in the control afforded to the client of the attendant storage services due to the strong semantics of those services. The limitation of Storage Networking is in its lack of scalability to large communities and across administrative domains.

But in the end storage technologies are validated in the marketplace and the state of future markets is a matter of conjecture. The strengths of Storage Networking address the legacy markets of owned, private storage repositories, which are the focus of intensive market attention, much as voice communication and mainframe computing once were. Logistical Networking, by contrast, is adapted to a future network environment that does not yet exist, and so is not the focus of market attention, much as data communication and personal computing once were not. In those historical cases, the companies that dominated the legacy markets (AT&T in the case of telephony, IBM in the case of computing) completely missed the ability of the disruptive technology to create new markets and ultimately make the old ones obsolete or unprofitable.

An example from the history of networking shows the dominance of wide area over local area considerations in setting standards. Networking was once dominated by file and print services implemented using the IPX protocol developed by Novell. The wide area network developed around the IP protocol that was built on the principles of end-to-end networking appropriate to a globally scalable network. Proponents of IPX complained, rightly, of performance and security issues with IP that made it less appropriate for local area networking, and Novell held out for their ultimate vindication when IP was found to be inadequate. As it happened, the value of IP as an internetworking protocol has overshadowed its shortcomings as a local area protocol for file services, and its ubiquity has created economies of scale in IP-based hardware and software systems that ultimately forced IPX out of the market.

Such examples suggest that *if* Logistical Networking actually succeeds as a technology for sharing of wide area storage, it will have a significant impact on the standards adopted for IP/Storage integration in local and system area Storage Networking. Under such circumstances, engineering arguments about the close match between Storage Networking and *today's* market for network-embedded storage are likely have little impact on the emerging market for storage as a shared resource in scalable networks. We believe that the companies that are paying attention to the unique characteristics of the wide area network will hold the key to those emerging markets, while those that insist that the future will resemble the past, only bigger and faster, will ultimately be selling commodity parts at ever-cheaper prices into the mature enterprise storage market.

References

1. A. Bassi, M. Beck, and T. Moore, "Mobile Management of Network Files," presented at Third Annual International Workshop on Active Middleware Services, San Franscisco, August 6, 2001.
2. M. Beck, T. Moore, J. Plank, and M. Swany, "Logistical Networking: Sharing More Than the Wires," in *Active Middleware Services*, vol. 583, *The Kluwer International Series in Engineering*

and Computer Science, S. Hariri, C. Lee, and C. Raghavendra, Eds. Boston: Kluwer Academic Publishers, 2000.

3. P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, pp. 145-1851994.

4. L. Chiariglione, "MPEG-1: Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s," International Organisation for Standardisation, Technical Report, ISO/IEC JTC1/SC29/WG11, 1996. http://mpeg.telecomitalialab.com/standards/mpeg-1/mpeg-1.htm.

5. J. H. Morris, M. Satyanarayan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith, "Andrew: A Distributed Personal Computing Environment," *Communications of the ACM*, vol. 29, no. 3, pp. 184-201, March, 1986.

6. J. S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems," *Software -- Practice and Experience*, vol. 27, no. 9, pp. 995-1012, September, 1997.

7. J. S. Plank, A. Bassi, M. Beck, T. Moore, M. Swany, and R. Wolski, "Managing Data Storage in the Network," *IEEE Internet Computing*, vol. 5, no. 5, pp. 50-58, September/October, 2001.

8. D. P. Reed, J. H. Saltzer, and D. D. Clark, "Comment on Active Networking and End-to-End Arguments," *IEEE Network*, vol. 12, no. 3, pp. 69-71, May/June, 1998.

9. J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277-288, November, 1984.

10. J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, "A note on distributed computing," Sun Microsystems, Technical Report SMLI, TR-94-29, November, 1994.

11. R. W. Watson and R. A. Coyne, "The Parallel I/O Architecture of the High-Performance Storage System (HPSS)," presented at IEEE Mass Storage Systems Symposium, 1995.

12. R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Future Generation Computer Systems*, vol. 15, pp. 757-7681999.