Logistical Networking

Sharing More than the Wires

Micah Beck, Terry Moore, Jim Plank, Martin Swany

Innovative Computing Laboratory,
Department of Computer Science,
University of Tennessee

Key words: logistical networking, active networking, storage, caching, electronic mail

Abstract: Logistical Networking is the global scheduling and optimization of data movement, storage and computation based on a model that takes into account all the network's underlying physical resources. In this paper we contrast Logistical and Active Network as approaches to flexible implementation of advanced network protocols. We describe the Internet Backplane Protocol, a simple mechanism we have devised for experimentation with Logistical Networking, and describe some applications which can take particular advantage of way that Logistical Networking exposes resources directly as network services.

1. Introduction — Sharing the Wealth

While traditional approaches to engineering focus on optimizing the use of scarce resources, exponential growth in all areas of computing resources — processing power, communication bandwidth, and storage — has presented forward-looking researchers with a different problem: designing new architectures that can take meaningful advantage of this bounty. The approach we describe in this paper, which we call *Logistical Networking*, presents one general way of using this rising flood of resources to create a common distributed infrastructure that can share out this growing bounty of storage and computation the way the current network shares communication bandwidth.

Of course in some sense sharing resources is the essence of what networking is about. The goal of computer networking is typically taken to be communication, i.e. the transmission of data between end-systems, where the core resource required to implement transmission is bandwidth. With simple point-to-point connectivity between end-systems, it is possible to build networks, such as the early e-mail system, that are completely managed by the hosts that use them to communicate. Normally, then, computer networking is defined in terms of communication between end-systems through the management of bandwidth resources; and, as the users of a network must access portions of a common medium owned and operated by others, it is natural to think of them as *sharing* this medium.

In public networks, such as the Internet, that sharing now takes a very strong form in which the entire infrastructure is potentially open to any user, subject only to the constraints imposed by high level peering that implements Acceptable Use Policies. But this radical form of sharing, made possible by IP networking, requires the network to be provisioned with more than just bandwidth. Routers and switches, in particular, represent significant computational resources. It is the addition of these intermediate computing elements to the management of the network, allowing them, instead of the end-systems, to control access and bandwidth allocation on individual links, that makes this more powerful and transparent approach to resource sharing possible. Very simple end-system applications that require connectivity are able to make use of a complex global Internet, massively provisioned with bandwidth and management infrastructure.

But this network-oriented approach to resource sharing has not yet been applied to storage or computation. While a lot of attention today is being focused on the deployment of advanced network

This work is supported by the National Science Foundation Next Generation Software Program under grant # EIA-9975015 and the Department of Energy Next Generation Internet Program under grant # DE-FC02-99ER25396

infrastructure in the world's research and education communities, the advanced nature of this infrastructure is defined almost completely in terms of a few characteristics of IP delivery, such as high bandwidth, advanced protocols (IPv6, multicast) and quality of service. Network applications that require resources other than IP delivery, like storage and computation, must still implement the management of those resources at the end-systems. If such capabilities are to be *put into the network itself*, so that it becomes a more comprehensive storehouse of resources that can be shared by users in Internet-like fashion, then intermediate resource management elements, accessible to every end-system, must be introduced to govern the way that applications access, draw from, and utilize this common pool.

In order to achieve Internet-like sharing of resources other than bandwidth, our research in Logistical Networking is exploring a radical approach:

> *Logistical Networking is the global scheduling and optimization of data movement, storage and computation based on a model that takes into account all the network's underlying physical resources, in contrast with more traditional networking, which does not explicitly model storage or computation resources in the network.*

We call this approach to networking "logistical" because of the analogy it bears with the systems of warehouses, depots and distribution channels commonly used in the logistics of military and industrial activity.

A*ctive Networking* is an approach to advanced networking that takes a major step in this direction. By allowing users and applications to make shared use of the computing power on the routers and switches, Active Networking creates the kind of management capability necessary to make good use of the exploding supply of computing and storage that can now be installed as part of the common network infrastructure.

A leading characteristic of Logistical Networking, which adopts a somewhat different paradigm than Active Networking, is the aggressive way it exposes resources for application to use. This is exemplified in our current research, which focuses on the use of network storage and uses a simple experimental protocol, called the *Internet Backplane Protocol* (*IBP*) [1, 2]. It implements a primitive storage service that applications can use for logistical purposes. While IBP allows network applications to make use of resources allocated at intermediate elements, it does not assume that the clients of those services are themselves located at intermediate elements, such as routers. Unlike most Active Networking approaches, IBP controlled resources can be used by either Active Networking elements or directly by end-systems. Logistical Networking aims to enable a wide variety of different ways of using storage in the network.

In the paper below we discuss the key aspects of this strategy. In the next section we discuss the paradigm that informs our approach to Logistical Networking and the development of IBP, contrasting it Active Networking. In section three we present IBP and the IBP API, which puts some of the details of our implementation of the Logistical Networking paradigm on display. We believe that IBP can be usefully applied to a number of application areas that currently implement their own middleware solutions to the management of state in the network, including Web caching and content distribution, distributed file and database systems, distributed data collection and management, and management of state in distributed computation; in section 4 we illustrate two of these. In section 5 we discuss some of the other research in the area that is more or less closely related to Logistical Networking.

2. Architectural alternatives for complex network functionality

The network designers of the mid-nineties faced a dilemma analogous to that faced by microprocessor designers of the mid-seventies. In the 1970s the design of processors was limited by the ability to place components on processor boards. As the number of transistors that could be placed on a chip rose to new heights, space became available that could easily accommodate a processor with room left over for more components. The problem of how to make use of this extra space led to the development of what later

came to be known as Complex Instruction Set Computers (CISC). The solution of CISC processor designers was to use extra capacity on the chip to store microcode and then to adapt the processor to the needs of the user by implementing complex instructions. In cases where a processor was shared, alternate microcode was sometimes dynamically downloadable. While several important assumptions drove CISC processor design, the one of most interest here was this:

> *Users would best be served by having a very simple view of the processor state but implementing complex transformations of data, because*
>
> 1. *Human programmers would be able to make use of highly complex instructions, and*
> 2. *Compilers could not make complex scheduling decisions on the basis of a complex model of the processor state.*

As we now know, the development of Reduced Instruction Set Computer (RISC) technology showed that these assumptions were flawed. Since humans *were* willing to give up assembly language programming and compilers *were* soon able to do very complex optimization and scheduling, making use of the reduced instruction set, it was preferable to expose the complex state of the processor to the instruction interface rather than hide it. Of course the trade-off between CISC and RISC also rested on the fact that the combination of reduced design and the continued increase in miniaturization meant that a RISC processor could fit on a single chip and still leave room for a substantial on-chip cache; but that is not part of our analogy.

Now fast-forward to the mid-nineties, when the designers of routers found that they had capacity to spare in the decision-making component of a router and sought to make use of it to enhance the power of the network. The Active Networking approach was to adapt the router to the needs of the user by implementing complex functionality. But because a network is essentially shared, it is not acceptable for a single user to statically adapt it to his or her needs, and so Active Networking requires that users be able to dynamically download code into the routers. While several important assumptions seem to be driving Active Network design, the one of most interest here is this:

> *Users are best served by having a very simple view of the network resources, but implementing complex data movement operations (e.g. caching, reliable multicast), because*
>
> 1. *Application developers will be able to make use of highly complex protocols, and*
> 2. *End systems will not be able to make complex scheduling decisions on the basis of a complex model of network resources.*

By contrast, the point of view taken in Logistical Networking is essentially the RISC philosophy applied to network design. Instead of encapsulating the resources of intermediate nodes in complex routing operations, we seek to expose them to the network through new operations that make the architecture of those nodes more visible. If an intermediate node has processing capabilities, then let it act as a processor of data, not merely a router on steroids. And if an intermediate node has storage capability, let it act as a depot to store data. The role of the (Internet Backplane Protocol) IBP is to support this latter capability, allowing a network's intermediate nodes to be equipped with storage resources, which are then exposed to applications through a new set of fundamental network operations.

3. The Internet Backplane Protocol (IBP)

IBP is middleware for managing and using remote storage [1, 2]. Invented to support Logistical Networking in large scale, distributed systems and applications, it acquired its name because it was designed to enable applications to treat the Internet as if it were a processor backplane. Whereas on a typical backplane, the user has access to memory and peripherals and can direct communication between them with DMA, IBP gives the user access to remote storage and standard Internet resources (e.g. content

servers implemented with standard sockets) and can direct communication between them with the IBP API.

By providing a uniform, application-independent interface to storage in the network, IBP makes it possible for applications of all kinds to use Logistical Networking to exploit data locality and more effectively manage buffer resources. We believe it represents the kind of middleware needed to overcome the current balkanization of state management capabilities on the Internet, any application that needs to manage distributed state to benefit from the kind of standardization, interoperability, and scalability that have made the Internet into such a powerful communication tool.

Since IBP draws on elements from different traditional designs, it does not fit comfortably into the usual categories of mechanisms for state management: IBP can be viewed as a mechanism to manage either *communication buffers* or *remote files*. Both characterizations are equally valid and useful in different situations. If, in order to use a neutral terminology, we simply refer to the units of data that IBP manages as *byte arrays*, then these different views of IBP can be presented as follows:

- **IBP as buffer management** — Communication between nodes on the Internet is built upon the basic operation of delivering packets from sender to receiver, where each packet is *buffered* at intermediate nodes. Because the capacity of even large storage systems is tiny compared with the amount of data that flows through the Internet, allocation of communication buffers must be time limited. In current routers and switches, time-limited allocation is implemented by use of FIFO buffers, serviced under the constraints of fair queuing. Against this background, *IBP byte arrays can be viewed as application-managed communication buffers in the network*. IBP supports time-limited allocation and FIFO disciplines to constrain the use of storage. With such constraints in place, applications that use these buffers can improve communication and network utilization by way of application-driven staging and course-grained routing of data.
- **IBP as file management** — Since high-end Internet applications often transfer gigabytes of data, the systems to manage storage resources for such applications are often on the scale of gigabytes to terabytes in size. Storage on this scale is usually managed using highly structured file systems or databases with complex naming, protection and robustness semantics. Normally such storage resources are treated as part of a host system and therefore as more or less private. From this point of view *IBP byte arrays can be viewed as files that live in the network*. IBP allows an application to read and write data stored at remote sites, as well as direct the movement of data among storage sites and to multiple receivers. In this way IBP creates a network of shareable storage in the same way that standard networks provide shareable bandwidth for file transfer.

This characterization of IBP as a mechanism for managing state in the network supplies an operational understanding of our approach to the problem of Logistical Networking for storage. The usual view is that routing of packets through a network is a series of *spatial* choices that allows control of only one aspect of data movement. An incoming packet is sent out on one of several alternative links, but any particular packet is held in communication buffers for as short a time as possible. But Logistical Networking with storage makes it possible to route packets in two dimensions, not just one: IBP allows for data to be stored at one location while *en route* from sender to receiver, adding the ability to control data movement *temporally* as well as spatially. This is a key aspect of Logistical Networking, but to see how IBP implements this concept we need to look at its API in detail.

3.1 IBP Structure and Client API

IBP has been designed to be a minimal abstraction of storage to serve the needs of Logistical Networking. Its fundamental operations are:

1. Allocating a byte array for storing data.
2. Moving data from a sender to a byte array.

3. Delivering data from a byte array to a receiver (either another byte array or a client).

We have defined and implemented a client API for IBP that consists of seven procedure calls, and server daemon software that makes local storage available for remote management. Currently connections between clients and servers are made through TCP/IP sockets. As we experiment with IBP we plan to explore other networking protocols (e.g. UDP) that can move IBP functionality closer to the network.

IBP client calls may be made by anyone who can attach to an IBP server (which we also call an IBP depot to emphasize its logistical functionality). IBP depots require only storage and networking resources, and running one does not necessarily require supervisory privileges. These servers implement policies that allow an initiating user some control over how IBP makes use of storage. An IBP server may be restricted so that it uses only idle physical memory and disk resources, or to enforce a time limit on all allocations, ensuring that impact on the host machine is either negligible or only finite in duration. The goal of such policies is to encourage users to experiment with Logistical Networking without over-committing server resources.

Logically speaking, the IBP client sees a depot's storage resources as a collection of append-only byte arrays. There are no directory structures or client-assigned file names. 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 to be interpreted only by the server. This approach enables applications to pass IBP capabilities among themselves without registering these operations with IBP, and in this way supports high-performance without sacrificing the correctness of applications.

The IBP client API consists of seven procedure calls, broken into three groups, defined in Table 1. The full API is described separately [3], but it has been implemented in prototype for Unix-based clients and servers and source code is available at http://icl.cs.utk.edu/ibp. We omit error handling for clarity.

Table 1. The IBP Client API

Allocation:
`IBP_cap_set IBP_allocate (char *host, int size, IBP_attributes attr)`
Reading/Writing:
`IBP_store(IBP_cap write_cap, char *data, int size)`
`IBP_remote_store(IBP_cap write_cap, char *host, int port, int size)`
`IBP_read(IBP_cap read_cap, char *buf, int size, int offset)`
`IBP_deliver(IBP_cap read_cap, char *host, int port, int size, int offset)`
`IBP_copy(IBP_cap source, IBP_cap target, int size, int offset)`
Management:
`IBP_manage(IBP_cap manage_cap, int cmd, int capType, IBP_status info)`

Since IBP's approach to allocation is the heart of its innovative storage model, the allocation procedure is the key part of this API. Storage resources embedded in a logistical network, especially if it’s a public network, cannot be allocated in the same way as they are on a host system. A public network serves a community which is not closely affiliated and which may have no social or economic basis for cooperation. To understand how IBP needs to treat allocation of storage for the purposes of Logistical Networking, it is helpful to compare the problem of sharing resources in the Internet with that of allocating storage resources on host systems.

In the Internet, the basic shared resources are data transmission and routing. The greatest impediment to sharing these resources is the risk that their owners will be denied the use of them. The reason that the Internet can function in the face of the possibility of denial-of-use attacks is that it is not possible for the attacker to profit in proportion to their own effort, expense and risk. When other resources (e.g. disk space in spool directories) are shared, we tend to find administrative mechanisms that limit their use by restricting the size of allocations or the amount of time for which data will be held. By contrast, a user of a host storage system is usually an authenticated member of some community that has the right to allocate certain resources and to use them indefinitely. Sharing of resources allocated in this way cannot extend to an arbitrary community. For example, an anonymous FTP server with open write permissions is an invitation for someone to monopolize those resources; such servers must be allowed to delete stored material at will.

In order to make it possible to treat storage as a shared network resource, IBP supports some of these administrative limits on allocation, while at the same time seeking to provide guarantees that are as strong as possible for the client. So, for example, under IBP, allocation can be restricted to a certain length of time, or specified in a way that permits the server to revoke the allocation at will. Clients who want to find the maximum resources available to them must choose the weakest form of allocation that their application can use.

To allocate storage at a remote IBP depot, the client calls **IBP_allocate()**. The maximum storage requirements of the byte array are noted in the **size** parameter, and additional attributes (described below) are included in the **attr** parameter. If the allocation is successful, a trio of capabilities is returned.

There are several allocation attributes that the client can specify:

- **Permanent vs. time-limited** — The client can specify whether the storage is intended to live forever, or whether the server should delete it after a certain period of time.
- **Volatile vs. stable** — The client can specify whether the server may revoke the storage at any time (volatile) or whether the server must maintain the storage for the lifetime of the buffer.
- **Byte-array/Pipe/Circular-queue** — The client can specify that the storage is to be accessed as an append-only byte array, a FIFO pipe (read one end, write to another), or a circular queue where writes to one end push data off of the other end once a certain queue length has been attained.

We expect that applications making use of shared storage in a logistical network will be constrained to allocate storage that is either permanent and volatile, or time-limited and stable.

Once space is allocated, all reading and writing to IBP byte arrays is done through the four reading/writing calls in Table 1. These calls allow clients to read from and write to IBP buffers. **IBP_store()** and **IBP_read()** allow clients to write from and read to their own memory. **IBP_remote_store()** and **IBP_deliver()** allow clients to direct an IBP server to connect to a third party (via a socket) for writing/reading. Finally, **IBP_copy()** allows a client to copy an IBP buffer from one server to another. Note that **IBP_remote_store()**, **IBP_deliver()** and **IBP_copy()** all allow a client to direct an interaction between two other remote entities. The support that these three calls provide for third party transfers are an important part of what makes IBP different from, for example, typical distributed file systems.

The semantics of **IBP_store()**, **IBP_remote_store()**, and **IBP_copy()** are append-only. **IBP_read()**, **IBP_deliver()** and **IBP_copy()** allow portions of IBP buffers to be read by the client or third party. If an IBP server has removed a buffer, due to a time-limit expiration or volatility, these client calls simply fail, encoding the reason for failure in an **IBP_errno** variable.

All management of IBP byte arrays is performed through the **IBP_manage()** call. This procedure requires the client to pass the management capability returned from the initial **IBP_allocate()** call. With **IBP_manage()**, the client may manipulate a server-controlled reference count of the read and write

capabilities. When the reference count of a byte array's write capability reaches zero, the byte array becomes read-only. When the read capability reaches zero, the byte array is deleted, though of course it may also be removed by the server, due to time-limited allocation or volatility. In this case, the server invalidates all of the byte array's capabilities. The client may also use **IBP_manage()** to probe the state of a byte array and its IBP server, to modify the time limit on time-limited allocations, and to modify the maximum size of a byte array.

4. Applications of Logistical Networking

Having articulated the Logistical Networking paradigm and described the IBP mechanism that implements it for the case of network storage, we now want to illustrate what such an implementation would mean for applications in terms of new functionality or performance enhancements.

Caching of objects with IBP — Consider a protocol such as HTTP that implements the transfer of objects over the network. One possible enhancement of network functionality is to build a cache for those objects. In order to identify the objects, extract them from the packet flow and correctly maintain a consistent cache, it is necessary to either support the abstraction of an object in the transport protocol or else have the cache understand the application protocol. Current Web caches operate on standard TCP-connected streams, and so must interpret the application protocol, HTTP.

A self-contained Web cache must be provisioned with a substantial dedicated disk resource (e.g. 10-100GB). Allocation, access and deallocation of storage are all implicit in requests to the cache: allocation is a side effect of a cache miss, access to stored objects is a side effect of a cache hit, and deallocation is carried out autonomously based on cache consistency and object replacement policies. The result is a system that provides a standard Web service interface and does not expose any explicit network storage functions. A well-functioning Web cache should never require a user to know about the state of its internal storage, even in the face of network, processor or storage failures.

One of the important limitations of Web caches is that they are specific to the HTTP protocol. There are many Web applications that might usefully make use of an object cache, and the only way that they can make use of a Web cache is to layer their object transfer on top of HTTP, which may have a negative impact on their performance and functionality. In the spirit of Active Networking, an execution platform which supports Web caching might also support a cache for another protocol, and an appropriately designed node OS might support the sharing of disk resources between them. However, the layering of resources occurs in the intermediate node, which becomes increasingly complex and less amenable to standardization.

The Logistical Networking approach to caching starts by redesigning application protocols, such as HTTP, in order to express object transfer using IBP, and this allows them to be flexibly stored and transferred between IBP depots located throughout the network. IBP depots are explicit network storage resources, and so the operations they implement (e.g. allocate, store, read) expose the state of their storage resources. A Web cache would then use IBP depots to maintain stored objects, implementing its own resolution and replacement policies. In that sense, the model of storage resources is pushed out of the node OS and onto the network, increasing the modularity of both the Active Networking policy engine and the storage resource manager, in this case the IBP depot (see figure 1).

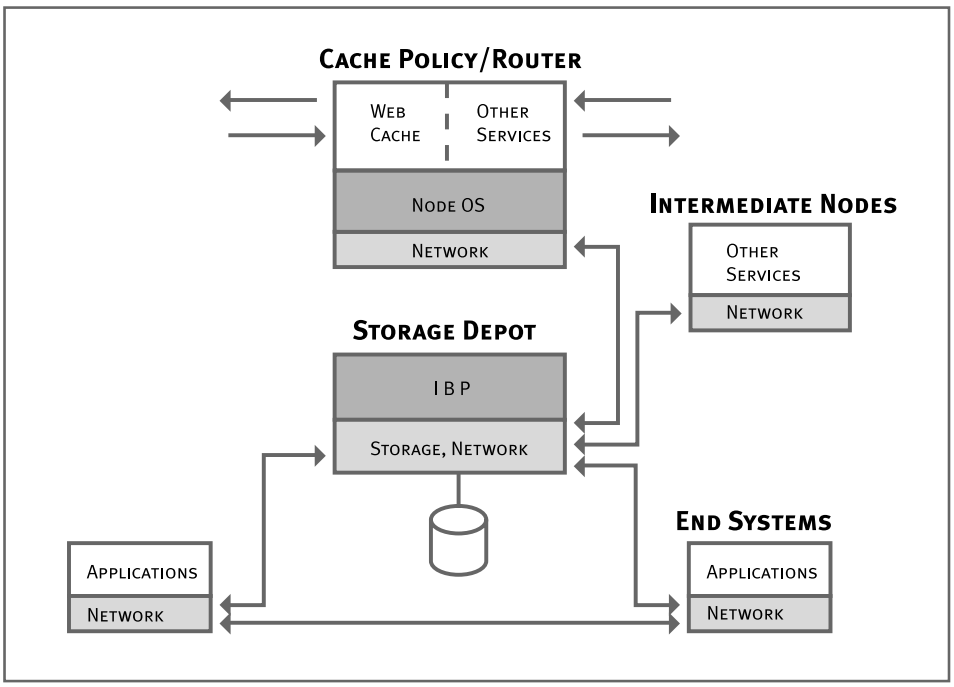

Figure 1. IBP depots expose network storage for applications of all types. An Active Netwok node cache and other storage-based network services, such as IBP-mail name service, are depicted

The Logistical Networking approach is not only an alternative architecture for Active Networking functions, it also allows for protocols to be implemented at end-systems which make use of network storage resources at IBP depots directly. If we think of the policy engine as a scheduler, then the scheduling agent can sometimes be replaced by explicit management of the network resource by the application or a scheduler at the end-system. Examples of applications which might benefit from doing their own scheduling are scientific computations which must manage their distributed state to achieve maximum performance (e.g. NetSolve [1, 4]), or database applications which have their own replication and consistency criteria which need to be implemented in a shared network policy agent. By exposing the resource directly to the network, a number of new architectural possibilities are enabled, only some of which put control of the storage resource directly at an intermediate node. An Active Networking function which does not use storage would not need to agree on storage services with other Active Networking modules in order to share a platform, and using this architectural approach, new network resources (such as a different model of storage, or perhaps computation) can be added without modifying the Active Networking platform.

IBP-Mail and Asynchronous Multicast — Electronic mail as implemented by the Simple Mail Transfer Protocol (SMTP) is one of the most powerful networking technologies and has become an integral part of all walks of everyday life. The basis of SMTP is the transfer of files that contain the text of an e-mail message and can also include MIME-encoded attachments. The entire SMTP payload, text and attachments, are delivered across the network to a mail server and stored in a spool directory allocated to the recipient. The transfer of files via SMTP attachment is a universal mechanism for file transfer, being independent of storage medium and operating system.

Unfortunately, SMTP attachments cannot be used to transfer files that are larger than the receiving mail server is willing to allocate in its spool directory, typically less than 10MB. Users seeking to transfer massive files sometimes resort to the use of an “upload directory” on an auxiliary FTP server, but this can result in a serious security lapse if that directory is unprotected. In application areas where massive data

transfer is required, the most common solution is still the use of a removable medium, usually magnetic tape, and Federal Express.

IBP-Mail is an application which makes use of storage on publicly available IBP depots to store large attachments, sending only a reference via IBP. The IBP-Mail reference is not, in fact, an IBP capability but a name that the receiver can resolve through an IBP-Mail name server to obtain the capability. This added level of indirection allows the IBP-Mail system to implement some sophisticated policies regarding placement and replication of the attachments in order to optimize the use of storage and bandwidth resources while maximizing availability and speed of access.

When the sender of an IBP-Mail message must pass a large file the IBP-Mail system, it chooses a depot close the sender to optimize the transfer, with proximity being defined according to the network metrics supplied by the Network Weather Service[5]. The IBP-Mail system passes a name for the attachment on to the receiver. A depot is then identified close to the receiver and transfer between them is initiated. If the receiver resolves the name before the transfer is complete they be directed to the former depot. If the receiver resolves the name after the transfer is complete they will be directed to the latter. Additional copies of the attachment may be kept on other depots to achieve a higher level of reliability or longevity.

A noteworthy aspect of the IBP-Mail system is the decision that must be made when there is more than one receiver, or if the attachment is forwarded. Assuming that the attachment is read-only, copies can be shared. However, for reasons of performance and reliability it may be preferable to make some number of copies and share them among sets of users. Under this view, the attachment is distributed through a form of asynchronous multicast, in a manner similar to the Network News system [6], with a new branch being created whenever a physical copy is made. Decisions about when to copy can be made using purely local information, or usable a global directory of all available copies. This form of asynchronous multicast could be a very powerful form of transparent optimization of file distribution.

5. Conclusion

Our work in Logistical Networking is closely related to the Active Networking community in spirit and goals. Researchers from MIT describe active networking elements as changing the traditional view of the network so that "...the entire network may be treated as part of the overall system that can be specialized to achieve application efficiency."[7] While they do not propose a general formulation of data logistics as part of that specialization, they do mention data caching applications for active networks, which must rely on significant storage resources. Further, Tennenhouse et. al. make reference to network storage of the sort that IBP provides in their discussion of new applications that will be enabled by active network infrastructures [8].

We anticipate implementing IBP on a variety of active network platforms such as ANTS[9], Bowman/CANEs[10], PLAN[11], and NodeOS[12] even though their designers had somewhat different problems in mind. While an extensible active networking node OS can provide a node execution environment with access to arbitrary storage resources, Logistical Networking envisions storage services such as IBP themselves being exposed to the network.

The underlying principle of Logistical Networking is to model the management of network resources using protocols that are as primitive as possible, and then to build higher-level abstractions on top of that primitive model. When we treat storage as a network resource, this means exposing the fact that network intermediate nodes have persistent state and can exhibit all of the failure modes associated with stateful devices. Our view is that higher-level protocols and other software components can take account of this complexity as well as software resident in or downloaded to the depot could.

By exposing the primitive view of a resource to the network, the hope is to create an infrastructure that is maximally applicable to different protocols and applications without requiring that these applications reside on the network node which implements the storage. In a Logistical Network, resources are exposed

and are managed in a variety of ways. Applications, policy and scheduling agents can all perform network resource management, and the code to manage those resources may reside at an end-system or in a network intermediate node such as a router, a proxy or a firewall. The challenge of managing a network rich in exposed resources may not be any easier than the creation of powerful Active Networking services based on resource-heavy routers, but it seeks to put the available resources into the hands of many more developers, hopefully allowing us to benefit from their determination to meet their particular challenges.

1. Beck, M., et al., *Logistical Quality of Service in NetSolve.* Computers and Communications, 1999. **22**: p. 1034-1044.
2. Plank, J., et al. *The Internet Backplane Protocol: Storage in the Network*. in *NetStore99: The Network Storage Symposium*. 1999. Seattle, WA.
3. Elwasif, W.R., J.S. Plank, and M. Beck, IBP - Internet Backplane Protocol: Infrastructure for distributed storage (V 0.2). Technical Report CS-99-430, University of Tennessee. February, 1999.
4. Casanova, H. and J. Dongarra, *Applying NetSolve's Network Enabled Server.* IEEE Computational Science & Engineering, 1998. **5**(3): p. 57-66.
5. Wolski, R., N. Spring, and J. Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*. Future Generation Computer Systems (to appear), 1999.
6. Kantor, B. and P. Lapsley, A proposed standard for the stream-based transmission of news. RFC 977, IETF. February, 1986.
7. Legedza, U., D. Wetherall, and J. Guttag, *Improving the Performance of Distributed Applications using Active Networks.* IEEE INFOCOM, 1998(April).
8. Tennenhouse, D., et al., *A Survey of Active Network Research.* IEEE Communications Magazine, 1997. **35**(1): p. 80-86.
9. Wetherall, D.J., J. Guttag, and D.L. Tennenhouse, *ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols.* IEEE OPENARCH, 1998(April).
10. Merugu, S., et al. *Bowman and CANEs: Implementation of an Active Network*. in *37th Annual Allerton Conference*. 1999.
11. Hicks, M., et al., *PLANet: An Active Internetwork.* IEEE INFOCOM, 1999(March).
12. Peterson, L., *AN Node OS Working Group. NodeOS Interface Specification*. 2000.