An Exposed Approach to Reliable Multicast
in Heterogeneous Logistical Networks

Micah Beck, Ying Ding, Erika Fuentes & Sharmila Kancherla
Logistical Computing and Internetworking Laboratory
Computer Science Department, University of Tennessee
{mbeck,ying,fuentes,kancherl}@cs.utk.edu

Abstract

An exposed approach in computer service architecture is one that offers client software a primitive service whose semantics are closely based on the underlying physical infrastructure. The exposed approach relies on the client to build higher-level services, with more abstract semantics, out of such primitive tools using sophisticated compilation or run-time algorithms. Current approaches to reliable multicast focus on encapsulated algorithms for efficient retransmission of datagrams to sets of receivers that require them. These approaches include augmenting the primary multicast data channel with direct TCP connections or with secondary multicast channels for retransmissions, and on the possibility of retransmissions originating from nodes in the middle of the network. In this paper we offer an exposed approach to multicast that uses an underlying Logistical Networking infrastructure that makes possible the implementation of any of the current retransmission algorithms, as well as new strategies yet to be devised.

1. Introduction

An exposed approach in computer service architecture is one that offers client software a primitive service whose semantics are closely based on the underlying physical infrastructure [1]. An exposed service relies on the client to build higher-level services, with more abstract semantics, out of such primitive services using sophisticated compilation or run-time algorithms. Classical examples of exposed services are RISC

instruction sets and IP datagram service, and each is an alternative to a more "encapsulated" approach which defines a client interface with less primitive semantics: CISC instruction sets and connection-based networking respectively.

The fundamental service of an IP network is the unreliable delivery of a datagram from a sender to a receiver, traversing a path of adjacent intermediate nodes all of which support this service. At each node, the datagram is stored in a buffer and then forwarded along the path. Logistical *Networking*, which uses scalable storage resources in the network to model communication in *both* its synchronous and asynchronous aspects [2], is more exposed than IP in that the fundamental service is the movement of data between buffers on adjacent nodes. In principle, IP datagram delivery service could be built on top of Logistical Networking. By exposing buffer management as an explicit network service, Logistical Networking enables the implementation of services such as reliable multicast that are not easily integrated into unreliable IP datagram delivery service.

1.1 Logistical Networking

The Internet Backplane Protocol (IBP) is a Logistical Networking mechanism that implements scalable management of storage in the network using shared physical resources [3, 4]. IBP is implemented by intermediate nodes called "depots" that implement standard storage operations including allocate, load, store and third party copy. From a networking perspective, an IBP depot can be viewed as a kind of router that exposes buffers to clients, which enables the implementation of flexible network services, such as an overlay style of multicast that enables an end-node to specify the delivery tree explicitly. Reliable multicast can be achieved by integrating retransmission as an end-to-end service at a higher level.

As the name suggests, the goal of Logistical Networking is to bring data transmission and storage within one framework, much as military or industrial logistics treat transportation lines and storage depots as

coordinate elements of one infrastructure. Achieving this goal requires a form of network storage that is globally scalable, meaning that the storage systems attached to the global data transmission network can be accessed and utilized from arbitrary endpoints. To create a shared resource fabric that exposes network storage for general use in this way, we set out to define a new *storage stack* (Figure 1), analogous to the Internet stack, using a bottom-up and layered design approach that adheres to the end-to-end principles. The important thing to note here is that the key to achieving scalability using this model 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.
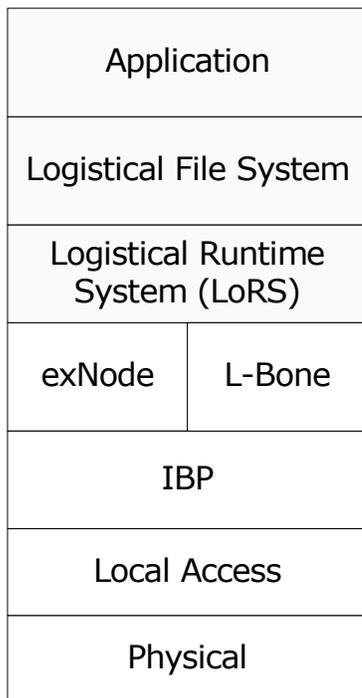
Application
Logistical File System
Logistical Runtime System (LoRS)
exNode	L-Bone
IBP
Local Access
Physical

Figure 1: Network Storage Stack

IBP is the lowest layer of the storage stack that is globally accessible from the network. Its design is modeled on the design of IP datagram delivery. Just as IP is a more abstract service based on link-layer datagram delivery, so IBP is a more abstract service based on blocks of data (on disk, memory, tape or other media) that are managed as "byte arrays." By masking the details of the storage at the local level — fixed block size, differing failure modes, local addressing schemes — this byte array abstraction allows a uniform IBP model to be applied to storage resources generally. The use of IP networking to access IBP storage resources creates a globally accessible storage service.

As the case of IP shows, however, in order to scale globally the service guarantees that IBP offers must be weakened, i.e. it must present a "best effort" storage service. First and foremost, this means that, by default, IBP storage allocations are time limited. When the lease on an IBP allocation expires, the storage resource can be reused and all data structures associated with it can be deleted. Additionally 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; more importantly, it makes network storage far more sharable, and easier to scale.

The semantics of IBP storage allocation also assume that an IBP storage resource can be transiently unavailable. Since the user of remote storage resources depends 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 "soft" 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," which are servers on which clients perform remote storage operations. A detailed account of the API and its other functions is available [3] online at (http://loci.cs.utk.edu/ibp/documents/). A description of the status of the current software that implements the IBP client, servers, and protocol is available at (http://loci.cs.utk.edu/ibp/software).

1.2   Approaches to Reliable Multicast

Native IP multicast distributes data to a group of destinations by defining a tree, each node of which represents a router; the edges out of each node represent a point-to-multipoint transfer between routers. IP multicast takes advantage of the fact that every recipient must receive every packet, so if the multicast is reliable or reliability is not required, every packet received is in fact required by that recipient. However, when IP datagrams are dropped independently on separate network links, the pattern of required retransmissions is irregular, and is unknown to the sender. A primary challenge of reliable multicast is the efficient distribution of retransmissions.

The simplest but least efficient approach to multicast retransmissions is to send them on the same channel as the original transmission. One source of inefficiency stems from the fact that every receiver must receive every retransmission, even if they don't require it. Another is that datagrams that have reached an

intermediate node must be retransmitted from the source. Greater efficiency can be achieved by retransmitting datagrams to individual receivers (perhaps using TCP) or to groups of receivers that have a high density requiring them [5-10] or by maintaining state at intermediate nodes and retransmitting from there [11, 12].

Various algorithms have been proposed to increase the efficiency of multicast retransmissions, but the efficiency of each depends strongly on the pattern of dropped datagrams. Furthermore, when complete reliability is not required, as in video streaming, the best algorithm may be application-dependent. The exposed approach that we describe in the next section is not inherently more efficient than any one proposed algorithm, but has the advantage that it can be used implement many of them, with varying degrees of efficiency. In Section 2 we describe the exposed approach, in Section 3 we describe how it allows the use of both UDP multicast and multiple TCP streams in the implementation of a multicast tree, and Section 4 presents performance results from the implementation of one particular algorithm. Finally, we present our conclusions in Section 5.

2. Exposed Reliable Multicast using IBP in Heterogeneous Networks

IP multicast encapsulates the tree used to distribute datagrams within IP routers, so that appropriate transfers are automatically executed whenever an IP datagram in the appropriate multicast group reaches the router. Exposed multicast using Logistical Networking is similarly implemented by a distribution tree with IBP depots at the nodes; but the transfer of data between the nodes of the tree is implemented by explicit buffer-to-buffer data movement operations invoked by an end-point, as we will describe further in Section 2.1. The state of the multicast distribution is explicitly implemented by the contents of the buffers, and it is the responsibility of the end-point to manage that state to ensure correct distribution of the data.

The data transfers that are represented by the edges of the exposed multicast distribution tree can be implemented using a variety of mechanisms, such as IBP Data Mover modules, as we will describe further in Section 2.1. Some of these modules are built on reliable mechanisms such as TCP, while others are built on unreliable mechanisms such as UDP. In either case, the fact that the depot buffers data between transfers means that end-to-end guarantees and services must be implemented at the end-points of the transfer. These end-to-end services, including reliability through the use of checksums and retransmission, are implemented by the Logistical Runtime System (LoRS), as will be discussed further in Section 2.2. Because reliability

through retransmission is implemented by the end-point as an explicit, exposed data transfer operation, it can be applied uniformly to data transferred using any available transfer service. This approach of layering end-to-end services over IBP is captured in the notion of the Network Storage Stack, represented in Figure 1, with unreliable IBP storage and data transfer mechanisms near the bottom, and the end-to-end services near the top. The Network Storage Stack is modeled after the IP Networking Stack, but with IBP playing a role analogous to that of IP. For a more detailed discussion see [4].

2.1 The Internet Backplane Protocol and the Data Movers

An IBP depot implements a simple service consisting of operations that allow clients to allocate storage space for limited periods of time (`IBP_allocate`), store data from a client into depot storage (`IBP_store`), transfer data between depots (`IBP_mcopy`) and deliver data from depot storage to a client (`IBP_load`) [13]. The intent of the service architecture is to allow the storage implemented by depots to be shared among a community of clients much as bandwidth and router resources, including memory and processor cycles, are shared in an IP network. While there are more details to the IBP API, for the purposes of this paper, the key properties are that a user need not authenticate their identity to the depot in order to allocate or use storage resources, and that the data transfer operations between depots are point-to-multipoint. These two properties allow clients to make use of a set of depots to construct a multicast distribution tree.

Given a multicast distribution tree with IBP depots at the nodes, storage space is allocated at each node to buffer data as it flows through the tree. Data to be transferred exists in a buffer at the root of the tree. Our discussion will concentrate on the movement of data between depots and the delivery of data to clients. Each transfer of data from a depot to a set of depots is implemented by a call to `IBP_mcopy`, which takes a source buffer, a set of destination buffers, and a specifier of a data movement mechanism as arguments. This abstraction allows the details of implementing each mechanism to be hidden from the client, but still allows substantial flexibility in the use of this primitive to implement an aggregate flow of data. Features such as parallel transfers between nodes using multiple TCP streams can be implemented on top of the fundamental TCP data movement mechanism that uses a single stream.

Software modules called Data Movers implement the different data movement mechanisms offered by the `IBP_mcopy` call. Each Data Mover has a unique identifier that is specified by the client as an argument.

Not every depot will implement every Data Mover, and some mechanisms can only be used between sets of depots that satisfy specific topological or configuration requirements, such as being connected by a private network or belonging to a single IP multicast cloud. It is the responsibility of the client to establish the applicability of a particular Data Mover in any given scenario; if a Data Mover is called inappropriately, it may return in error or it may have an unpredictable effect. By invoking an appropriately chosen Data Mover at each node in the multicast distribution tree, the resources of a heterogeneous network can be used to maximum advantage.

2.2 The exNode, End-to-End Services and LoRS

In order to build a storage service with stronger semantics than IBP provides, it is necessary to aggregate IBP allocations, fragmenting large storage allocations across multiple IBP depots and storing data redundantly [1]. To implement such aggregation, we have defined a data structure called the *exNode* to keep track of the mapping of a single data extent to multiple IBP allocations [14]. The exNode serves a role in Logistical Networking that is analogous to that of the *inode* defined within the Unix file system to aggregate fixed size disk blocks into a single large file.

The dual nature of Logistical Networking, as both a storage and a networking technology, is made evident by the way in which the exNode is also the data structure that is used to implement algorithms for end-to-end services. These services, which are typically implemented by transport layer network protocols such as TCP, include reliability, through checksums and retransmission, and security, through encryption. When data is copied between remote and local IBP allocations, each can be represented as a replica of the data within an exNode. Then when data is accessed through the exNode, the local copy will be accessed first. If the copy operation was unreliable and the local replica is incomplete, then the missing data can be retrieved directly from the source. Thus, retransmission can be implemented as replication.

A collection of end-to-end services (e.g. fault tolerance, encryption and compression) have been implemented as a library of operations on exNodes, and these services are used to implement a higher-level library of data handling functions known as the Logistical Runtime System (LoRS). The basic LoRS functions are `lors_upload` (copy data from an end-point to IBP allocations), `lors_download` (copy data from IBP allocations to an end-point), `lors_augment` (copy data between IBP allocations and enlarging the exNode accordingly), `lors_trim` (remove IBP allocations from an exNode) and `lors_refresh` (extend the duration of IBP allocations in an exNode) [15-17]. These functions are available as user commands as well as programming library calls.

Like the `IBP_mcopy` call, `lors_augment` specifies the creation of multiple copies of the data represented by the exNode. However, while the `IBP_mcopy` is limited to invoking a single specified Data Mover module to accomplish point-to-multipoint data movement, `lors_augment` operates at a higher level, and can use sophisticated data routing techniques. Specifically, it can build a multicast tree and then use a collection of `IBP_mcopy` calls to implement data movement as specified by that tree, perhaps making use of different Data Movers at each node to implement a heterogeneous data transfer.

3. Point-to-Multipoint Data Mover Operations

The `IBP_mcopy` call implements point-to-multipoint data transfer that can take advantage of optimizations that cannot be implemented using only point-to-point operations [1]. These optimizations include fine-grained buffer management, flow control and retransmission algorithms, which are either implemented within the operating system network stack or require low-latency responsiveness that is not possible through the currently defined IBP API.

The choice of operations available for any particular data transfer depends on the set of Data Movers implemented at each depot and on the nature of the network services available between them. For instance, if all of the participants in the transfer are in a single IP Multicast cloud that has a sufficiently low packet loss rate, then an Unreliable UDP Multicast Data Mover can be used. Otherwise, it may be necessary to use the TCP Data Mover, which relies on parallel point-to-point transfers over TCP connections. In our Network Storage Stack, `lors_augment` invokes the appropriate Data Mover.

The two Data Movers that we are most concerned with in this paper are the Unreliable UDP Multicast Data Mover and the TCP Data Mover. However, we have experimented with a reliable UDP-based data mover that uses TCP-like windowing and retransmission, and we are collaborating with the Web100 project [18] to develop a Data Mover that uses tuned TCP streams to achieve higher throughput. Other Data Movers may be developed as needed.

3.1 Point-to-Multipoint TCP Data Mover

While a point-to-point multicast can simply be iterated to implement a point-to-multipoint data transfer, the speed of sending will decrease linearly with the number of destinations. This is because each operation,

including reading data from the IBP allocation on disk at the source, is performed for each transfer. However, by loading a segment of the data from the IBP allocation into the address space of the data mover and then sending it to each destination in turn, the per-destination overhead can be minimized [19].

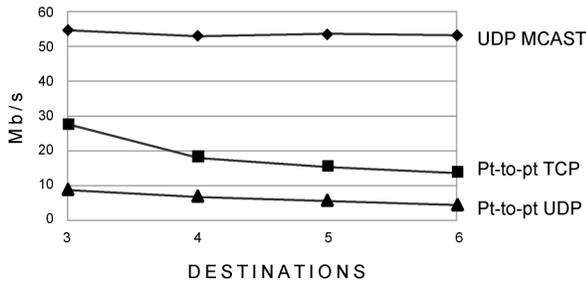**Multicast vs TCP vs UDP sending rate comparison**

Figure 2: Comparison between the sending rate obtained using Reliable TCP and UDP point to point, and Unreliable UDP Multicast Data Movers

Figure 2 shows a comparison of the "sending rate" obtained using three different Data Movers in the local area network. Two of the three Data Movers are implemented data movement using multiple reliable point-to-point protocols (TCP and UDP) and one is implemented using unreliable UDP multicast. The sending rate is the total amount of data transferred, 50MB in these cases, divided by the total time for the transfer. In particular, it does not increase with the number of receivers. The speed of the reliable UDP is lower in these experiments because it is optimized to operate in a high bandwidth, high latency, low loss wide area network. It is included here only to illustrate the heterogeneity that is available through Data Movers.

The machines that were used to implement these experiments represent a variety of architectures and configurations, all connected to the network using a 100Mb/s interface. Two were Sparc workstations running SunOS with 512MB of RAM; the remainder were various PCs with Intel Pentium processors running Linux. The RAM capacities of the PCs were 256MB with the exception of one system that has 512MB. The specific processor implementation and speed of the Intel Pentiums varied from Pentium III (Coppermine) to Pentium 4 at 1700MHz. Each reported speed is the average of at least 5 experiments.

3.2 Unreliable UDP Multicast Data Mover

While IP Multicast is highly efficient and scalable, universal deployment has proved difficult. However, when a source a set of destinations within a multicast tree all fall within the same IP Multicast cloud, we can use the Unreliable UDP Multicast Data Mover to implement the transfer. Determining when UDP Multicast can be used, and assigning multicast addresses to individual transfers are a non-trivial tasks; but these issues apply to all applications of IP Multicast and are not unique to our Data Mover. As we described above, the unreliable nature of UDP Multicast is overcome in the LoRS library by retransmissions from the source copy using TCP. Thus, the efficiency of the Data Mover depends heavily on the packet loss rate. In addition, the UDP Multicast Data Mover currently has no flow control, and so may be inappropriate for use in congested networks. As Figure 2 shows, the sending rate of the UPD Multicast Data Mover is constant as the number of destinations increases, as would be expected.

4. Building a Heterogeneous Multicast Tree

Given the flexible tools described in Sections 2 and 3 for the direction of data transfers in the wide area network, and given a source and multiple destinations for data, it is necessary to build a multicast tree that implements the specified transfer, as illustrated in Figure 3. Because we are working in an overlay network, the possible topologies of such a tree are essentially limitless. Even if we had direct access to the physical topology of the network, if an IBP is not deployed at every router, we cannot use unicast routing tables directly to construct the tree as in IP Multicast. The problem of optimizing the layout of such trees is a difficult research area, beyond the scope of this paper. However, we describe a simple approach to indicate the type of solution that we see as promising.

The characteristics of our current UDP Multicast Data Mover make it inappropriate for use in networks that may be congested, and so the first step is to identify maximal clusters of destinations that lie within a single IP Multicast clouds and that are also connected to a single (hopefully over provisioned) local area network. Within those clouds, we can choose a root node and then connect the source node to the root of each IP multicast cloud (considering the source itself to be the root of the IP multicast cloud it lies in) with a single TCP Data Mover operation. This simple approach ignores the question of whether the TCP Data Mover is more efficient than the UDP Multicast Data Mover in certain cases, as suggested by the results in Figure 2. Also, TCP may be more effective in networks that have high packet loss, as the UDP Data Mover will require the endpoint to retransmit significant amounts of data.

Figure 3 presents an example of a test topology on which we conducted preliminary experiments using this approach: node S represents the source data, and A through D and 1 through 8 are the destinations. The primary purpose of these experiments was to compare the performance of deploying multicast in two different

ways: building an overlay multicast tree vs. using multiple TCP connections to reach all the endpoints directly.
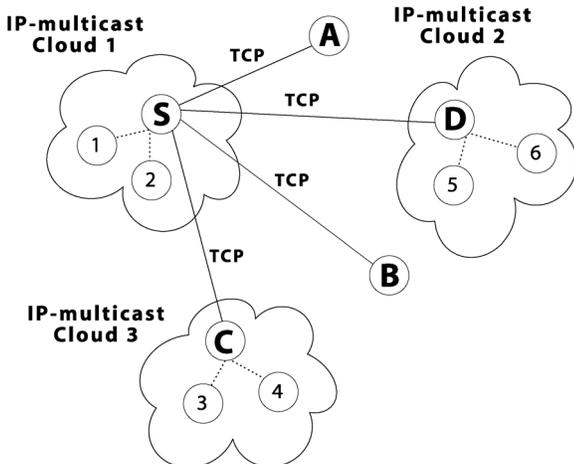

Figure 3: Example of topology showing heterogeneity.

The tests were executed on wide area network test bed which includes nodes on the University of Tennessee campus at Knoxville (UTK) and PlanetLab, a global overlay infrastructure for developing and accessing new network services [20]. All systems in IP-multicast cloud 1 (nodes S, 1 and 2) are single-processor Linux machines with 256MB of RAM located on the UTK campus. Except as nodes, all remaining depots in the multicast tree run on standard PlanetLab nodes: single-processor systems with 883.8MB of RAM running Linux and connected by a 100Mb/s network interface. Machine A is located in Pittsburgh and is connected with a 10Mb/s network card while machine B is located in Italy. The three machines in IP-multicast Cloud 3, located in Cambridge, UK, are equipped with 1Gb/s network cards.

To increase performance, each transfer was executed using 10 concurrent TCP connections. Each test transferred 50MB, and the throughput reported here is an average of at least 5 runs. The sending rate measured for direct TCP transfer from source to all receivers was 3.4 Mb/s, whereas using the overlay multicast distribution tree built on top of TCP connections between depots, the sending rate was 5.1 Mb/s, or an increase of 50%.

Another experiment was performed, comparing the performance of the overlay multicast tree build using TCP connections to one which uses UDP multicast to send from nodes S, C and D to the nodes within their local native IP clouds: {1, 2}, {3, 4} and {5, 6} respectively. However, because the UDP multicast Data Mover does not implement flow control, it is not possible to obtain high data rates by running multiple instances in parallel. However, comparing a non-concurrent instance of the TCP-only implementation of the overlay multicast tree to an non-concurrent instance of one that use uses UDP multicast locally, we still saw a speedup of some 15%. Thus, even when the TCP implementation is not stressed by concurrent transfers, the implementation that uses native IP multicast can still obtain an improvement.

Using the UDP multicast Data Mover to obtain high performance transfers is complex, due to the lack of flow control and reliability. These limitations will be corrected in future versions of this Data Mover, and can also be addressed through more complex strategies for building and transferring data on the overlay distribution tree. These early experiments merely serve to show that the exposed approach is feasible and can obtain better performance than simple repeated unicast.

5. Conclusions

We have defined an approach to reliable multicast that implements the overall multicast tree as a concatenation of exposed point-to-multipoint data transfers. Reliability is implemented at the end-points, using a simple retransmission strategy. This approach allows the use of UDP Multicast in scenarios where there is a low packet loss rate, and reverts to a TCP-based approach that is optimized for point-to-multipoint transfer in other cases. The framework we have defined relies heavily on our ability to find heuristics for constructing an effective multicast tree given a particular network topology, IP Multicast configuration, traffic characteristics and packet loss rates. The overlay nature of our implementation gives us tremendous flexibility in the construction of multicast trees, and managing this explosion of choices will be the subject of future research.

There is an innovative approach to reliable multicast using the active networking execution environment Tamanoir in combination with IBP as proposed in [21]. Tamanoir services run in the application layer of the network storage stack; it was designed mainly for data transfers using unreliable protocols (such as UDP) for multimedia distribution. The possibility of placing active services at intermediate nodes allows us to consider implementing some services such as rate control and retransmission within the network rather than only at the end-points. This might allow us to implement end-to-end services more efficiently, although it does not eliminate the need to implement them at the end-points to ensure end-to-end guarantees.

6. Acknowledgements

preparation of this paper. Experimentation using the Internet Backplane Protocol is possible only because of the Alex Bassi's extraordinary implementation skills. Finally, this work is only possible through Jim Plank's shared leadership of the Logistical Computing and Internetworking Laboratory and the talents and tireless efforts of its professional staff and students.

7. References

[1] M. Beck, T. Moore, and J. S. Plank, "Exposed vs. Encapsulated Approaches to Grid Service Architecture," in *Grid Computing -- GRID 2001*, *LNCS 2242*, C. Lee, Ed. Denver, CO: Springer Verlag, 2001, pp. 124-132.

[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] 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.

[4] M. Beck, T. Moore, and J. S. Plank, "An End-to-end Approach to Globally Scalable Network Storage," in *Proceedings of ACM Sigcomm 2002*. Pittsburgh, PA: Association for Computing Machinery, 2002.

[5] M. H. Ammar and L. Wu, "Improving the Throughput of Point-to-Multipoint ARQ Protocols Through Destination Set Splitting," in *Proceedings of IEEE Infocom*, 1992, pp. 262-269.

[6] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang, "A Reliable Multicast Framework for Ligh-weight Sessions and Application Level Framing," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 784-803, December, 1997.

[7] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-Based Receiver Reliable Multicast for Distributed Interactive Simulation," in *Proceedings of ACM SIGCOMM*, 1995, pp. 328-341.

[8] S. K. Kasera, G. Hjalmtysson, D. F. Towsley, and J. F. Kurose, "Scalable reliable multicast using multiple multicast channels," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, 2000.

[9] R. Gau, Z. Haas, and B. Krishnamachari, "On Multicast Flow Control for Heterogeneous Receivers," *IEEE/ACM Transactions on Networking*, vol. 10, no. 1, pp. 86-101, February, 2002.

[10] D. Chiu, M. Kadanski, and J. Wesley, "A Flow Control Algorithm for ACK-based Reliable Multicast," Sun Microsystems Labs, May, 1999.

[11] L. Lehman, S. Garland, and D. Tennenhouse, "Active Reliable Multicast," in *Proceedings of IEEE Infocom,*, 1998, pp. 581-589.

[12] C. Papadopoulos, G. Parulkar, and G. Varghese, "An Error Control Scheme for Large-Scale Multicast Applications," in *Proceedings of IEEE Infocom*, 1998, pp. 1188-1196.

[13] A. Bassi, M. Beck, J. Plank, and R. Wolski, "Internet Backplane Protocol: API 1.0," Department of Computer Science, University of Tennessee, Knoxville, TN, CS Technical Report, ut-cs-01-455, March 16, 2001. http://www.cs.utk.edu/~library/2001.html.

[14] A. Bassi, M. Beck, and T. Moore, "Mobile Management of Network Files," in *Third Annual International Workshop on Active Middleware Services (AMS 2001)*. San Franscisco, CA: Kluwer Academic Publishers, 2001, pp. 106-115.

[15] S. Atchley, S. Soltez, and J. S. Plank, *LoRS C-API Documentation ver. 0.7*, LoCI Laboratory, Department of Computer Science, University of Tennessee, http://loci.cs.utk.edu/lors/lors_api/index.html, 2002.

[16] S. Atchley, S. Soltesz, J. S. Plank, M. Beck, and T. Moore, "Fault-Tolerance in the Network Storage Stack," presented at Annual Workshop on Fault-Tolerant Parallel and Distributed Systems (in conjunction with International Parallel & Distributed Processing Symposium), Ft. Lauderdale, FL, USA, April 15-19, 2002.

[17] J. S. Plank, S. Atchley, Y. Ding, and M. Beck, "Algorithms for High Performance, Wide-Area, Distributed File Downloads," Department of Computer Science, University of Tennessee, Knoxville, Technical Report, UT-CS-02-485, October, 2002. http://www.cs.utk.edu/~plank/plank/papers/CS-02-485.html.

[18] M. Mathis, J. Heffner, R. Reddy, and J. Saperia, "TCP Extended Statistics MIB," IETF, Internet Draft, November, 2002. draft-ietf-tsvwg-tcp-mib-extension-02.txt.

[19] E. Fuentes, "The Internet Backplane Protocol Data Mover Module," Department of Computer Science, University of Tennessee, Knoxville, Technical Report, UT-CS-03-503, February, 2003.

http://www.cs.utk.edu/~library/TechReports/2003/ut-cs-03-503.ps.

[20]   "Proceedings of ACM HotNets-I Workshop." Princeton, New Jersey, USA, 2002.

[21]   A. Bassi, J.-P. Gelas, and L. Lefˋevre, "Tamanoir-IBP : Adding Storage to Active Networks," in *Fourth Annual International Workshop on Active Middleware Services (AMS 2002)*. Edinburgh, Scotland: IEEE Computer Society, 2002, pp. 27-34.