

# Information Security on the Logistical Network: An End-to-End Approach

Micah Beck, James S. Plank, Jeremy Millar, Scott Atchley  
Stephen Soltesz, Alessandro Bassi, Huadong Liu

Logistical Computing and Internetworking Laboratory  
Department of Computer Science  
University of Tennessee, Knoxville, TN 37996

## Abstract

*We describe the information security aspects of logistical networking. The security model adopted by logistical networking is an end-to-end model that provides tunable security levels while maintaining the scalability of the network as a whole.*

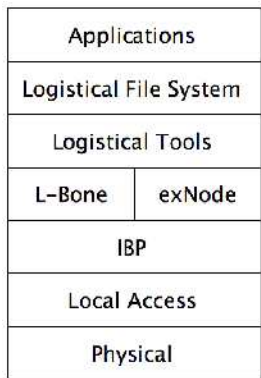
## 1. Introduction

Information security has been defined as "being able to control who has access to your information" [11], with the goal of providing

- Confidentiality of the data.
- Non-repudiation.
- Data integrity.
- Access control.
- Authentication.
- Resilience in the face of attack, particularly with respect to denial of service.

The Logistical Networking project from the University of Tennessee [1] seeks to build a scalable, globally shared storage and compute infrastructure to support asynchronous communications. A key feature of logistical networks (LONs) is the *public* availability of storage embedded in the network fabric, modulo some simple IP address-based access controls. This allows LON users to access storage when and where they need it, with a near-zero administrative cost.

The goals of information security can be difficult to meet in the context of a corporate or departmental network. Those challenges are even more difficult in the context of a large distributed system like a logistical network, where data must flow over and be stored in a public infrastructure. This paper explores the security issues inherent in such a



**Figure 1. The network storage stack.**

---

system, and describes the approach taken by logistical networking to address them. We begin with a discussion of the core logistical networking technologies in Section 2. Section 3 discusses the particulars of information security on logistical networks. Section 4 provides some discussion of related projects, and Section 5 concludes.

## 2. Logistical Networking

Logistical networking has been defined as the global optimization and scheduling of data storage, data movement, and computation [5]. Practically speaking, logistical networking is the augmentation of networks with storage resources for the purposes of enabling asynchronous communications. Applications of logistical networking include the storage and high-speed movement of large data sets [15], content distribution [4], caching [9], and multicast [6].

The core technologies of logistical networking are arranged in a layered architecture known as the network storage stack, Fig. 1. The following sections describe each layer of the stack in detail.

## 2.1. Physical Storage

The bottom layer of the network storage stack is the physical storage resource. The physical storage layer can be as simple as an IDE disk, or as complex as a sophisticated storage area network or hierarchical storage system. Physical storage resources may also be non-disk forms of storage, e.g., memory or tape. The primary characteristic of a physical storage resource is that it is capable of storing some amount of data for some period of time. Moreover, the logistical networking conception of physical storage is that it is best effort, i.e. storage resources can fail or become unavailable without warning.

## 2.2. Local Access

The next layer of the network storage stack is local access to storage. This layer is generally provided by the operating system in the form of a file system. Other access methods such as user-level file systems and APIs do exist for specific storage resources. The primary purpose of the local access layer is to provide a means of allocating space on a storage resource, and reading and writing to that allocation.

## 2.3. IBP

The next layer in the storage stack is the Internet Backbone Protocol (IBP). IBP is also the first logistical layer of the stack. IBP attaches storage resources to the network and provides access to them in the form of time-limited allocations. The combination of one or more storage resources and a network service that speaks IBP is referred to as a *storage depot*.

Depots are placed directly into the network, thereby augmenting the network with user-addressable storage. This is similar to the addition of computational capabilities provided by active networking.

Storage resources provided by IBP depots have a number of unique characteristics. First, storage is provided to users on a lease basis. Users request storage from a depot for a *limited* amount of time. When the lease expires, the depot is free to reclaim the storage space for other users. Thus it is incumbent upon users to renew their leases or move their data in a timely manner, or risk losing it.

Second, storage is provided in a "best effort" manner. Because depots are network services, they can be no more reliable than the network itself. In particular, depots may become unavailable for unknown periods of time without warning, due to network outages, high traffic levels, operating system faults, or physical storage failures. Best effort storage semantics often come as a surprise to new users; however, storage systems are never able to guarantee more

than best effort service. The important contribution of IBP is that it makes that limitation explicit.

Using IBP is straightforward. Users request a fixed-size allocation from a particular depot, and specify a lease length. The depot responds with a failure notification (perhaps it's low on storage space) or a set of *capabilities* that can be used in further communications with the depot. Capabilities are cryptographically secure strings that represent the right to read from, write to, or otherwise operate on a specific IBP allocation. Users must present the depot with the appropriate capability in order to make use of an allocation.

## 2.4. L-Bone

The next layer of the stack is comprised of a pair of resource management technologies. The first of these is the Logistical Backbone, or L-Bone. The L-Bone provides a resource discovery mechanism for applications on a logistical network by maintaining a directory of known IBP depots. Additionally, the L-Bone tracks a variety of metadata for each known depot, including current status (up or down), longest available lease, available storage, and location. Applications can query the L-Bone to find IBP depots that match their particular requirements, e.g., available storage or lease length.

The L-Bone also functions as a resolution service. In other words, applications can query the L-Bone for depots that are "close" to an arbitrary location, either in geographic or network terms. This allows applications to store data close to themselves, or in cases where data will be consumed at another location, close to the consumer.

To date, the public L-Bone directory (housed on the National Logistical Networking Testbed) contains some 160 depots with an aggregate storage capacity of approximately 20 terabytes. The L-Bone is expected to grow to a storage capacity of 50 terabytes over the next few years.

## 2.5. exNode

The second component of the resource management layer is the exNode. The exNode is a portable data structure designed to ease the management of data stored on a logistical network. It achieves this goal by providing a framework for solving a number of problems exposed by the lower levels of the storage stack.

The first among these is the need to store and manage the capabilities from multiple IBP allocations. This need is a direct consequence of the weakened storage semantics provided by IBP. In particular, applications often find it necessary to fragment their data across multiple IBP depots, for two reasons: 1) it may not be possible to store large data sets on a single depot due to policy constraints governing the

size of any given allocation, and 2) best effort storage service implies the need to replicate data across multiple depots for fault tolerance purposes.

Since each allocation has three associated capabilities, the number of capabilities the application needs to keep track of quickly explodes. Moreover, the individual capabilities provide no clue as to which portion of the stored data they reference. The exNode addresses this problem by defining a *mapping* from some contiguous subset of the data to a particular set of capabilities. A mapping may include the entire data set, or it may be only a fragment of the data. Additionally, the exNode allows for multiple mappings for a subset of the data, i.e., a subset may be mapped to more than one set of capabilities. Moreover, a particular portion of the data may be represented by overlapping mappings.

The second problem addressed by the exNode is one of abstraction. Dealing with logistical storage in terms of IBP is akin to operating directly with disk blocks. This quickly becomes cumbersome, particularly if any sort of replication or fragmentation is desired. The exNode allows applications to deal with sets of IBP allocations as if they were something more like a file, i.e., the exNode is a logistical analogue of the file system's inode.

Other capabilities offered by the exNode include tagging mappings with arbitrary (possibly application-specific) metadata. Mappings can have any number of metadata items (consisting of a name, value, and type) attached to them. The metadata facility is commonly used by higher levels of the network storage stack to encode information about the stored data – for instance, if the data has been encrypted, the particular cipher and any associated keys may be stored as metadata in the exNode.

The exNode can also store information about any transformations performed on the data prior to its storage on an IBP depot. Generally speaking, data stored into a depot is the result of a composition of invertible transformation functions. The exNode contains the exact composition function applied to the data so that the data can be recovered. This facility is used by higher layers of the stack to track the application of various end-to-end services (checksums, encryption, and compression).

## 2.6. Logistical Runtime System

The next layer of the network storage stack is the Logistical Runtime System, or LoRS. This is the entry point into the logistical network for most applications. LoRS provides facilities for storing data into a logistical network, retrieving data from a logistical network, moving data around a logistical network, and managing data on a logistical network. These facilities are provided via user tools and a programming library.

LoRS handles much of the bookkeeping and administrative required to operate with data stored on a logistical network. For instance, LoRS provides facilities to fragment and replicate data. The resulting storage pattern may be complex: not all fragments need have replicas, and some may be replicated more frequently. LoRS allows users to operate on this data transparently, as if it were a single, non-fragmented copy.

LoRS also handles many of the logistical aspects of the network for the user. For example, assume a data set has been fragmented into  $N$  fragments, and that each fragment has  $C$  copies, stored in a total of  $N * C$  allocations. Furthermore, assume a goal of maximizing read throughput for the entire data set. It is obvious that not all copies of a given fragment are equal in terms of performance – one particular copy may be on a slow link, while another is hosted on a heavily loaded machine, etc. Moreover, the performance characteristics of a given fragment change as a function of time. LoRS takes a number of approaches to ensure the best possible throughput.

The first is parallel download, similar to the swarming mechanism used by BitTorrent [2]. LoRS will start multiple transfers at various offsets in the data set, as described by the mappings contained in an exNode. Concurrent transfers allow LoRS to maximize utilization of the client's bandwidth.

In addition to parallel download, LoRS uses an adaptive algorithm to ensure the most performant IBP depots service the largest number of requests. This is achieved by job queuing, i.e., the depots that respond the fastest ultimately service the largest number of requests.

LoRS also supports a notion of redundant requests to ensure progress. If download progress is stalled by a particularly slow fragment, LoRS will dispatch a redundant request to retrieve a copy of the fragment. When one of these requests returns, the redundant request is canceled, and the download progresses. See [15] for a complete discussion of these algorithms.

To ensure that users can retrieve their data in a location independent manner, LoRS traffics exclusively in exNodes. The result of a LoRS store operation is an exNode describing where each fragment was placed, and what transformations were applied to the data prior to storage. Additional application-specific metadata can be included in the exNode and applied to individual mappings, as well as the entire data set. Since exNodes are portable, it is possible for users to store data from one location, and retrieve it from somewhere else.

LoRS also affords the user great control over the layout of their data on the network. It is possible to stage data near where it will be read, to disperse data over a wide geographic area, and in general, to emulate any sort of RAID encoding that might be desired.

To ensure that data is not altered by the storage and retrieval process, and to ensure that data stored on untrusted IBP depots is not compromised, LoRS provides a number of end-to-end security-oriented services. These include checksumming, encryption, compression, and Reed-Solomon coding. These will be discussed in detail in the next section.

### 3. Security on the Logistical Network

While logistical networks provide a flexible, high-performance storage solution, their public nature makes them untrustworthy. Coupled with "best-effort" service guarantees, this fact makes information security on logistical networks a paramount concern. Here we take information security to mean control over access to data. As mentioned in the introduction, the goal of information security is to provide the following:

- Confidentiality of the data.
- Non-repudiation.
- Data integrity.
- Access control.
- Authentication.
- Resilience in the face of attack, particularly with respect to denial of service.

We add

- Fault tolerance.

to the list to account for the unpredictable nature of network storage.

We begin our discussion of security issues by investigating the nature of trust relationships as applied to storage services.

#### 3.1. Trust Relationships

When investigating the information security afforded by a particular storage system, it is important to also investigate who the user is required to trust. For instance, storing data on a local area network requires the user to trust the systems administrators not to snoop through potentially private data. Indeed, even encryption may not stop a singularly malicious administrator, since the encryption keys are presumably stored in files on the system.

While it appears at first blush that we must only trust the local administrators, the web of trust relationships is somewhat more complex. Assuming we trust local personnel, can we guarantee that the operating system vendor has not installed back doors that will enable access to data? If not, we must also trust the vendor.

This may seem like extreme paranoia, but such levels of security can be warranted. Consider notes from a corporation's board meeting that discuss various corporate strategies, or financial information. Corporate espionage is not unknown, and a diligent corporation will take measures to protect itself. More ominously, many attacks on an organization's information originate from *inside* the organization itself.

The issues of trust relationships are compounded in a distributed storage solution. Not only must local administrators and systems vendors be trusted, but so must networking vendors, ISPs, etc. These problems are magnified further if the storage system is public, as is the case with logistical networking. While it might be feasible for a large organization to vet its storage vendors in a partnering relationship, that is clearly not possible for logistical networking.

In particular, IBP depots may be set up by anyone, for any purpose. There is no way to know if a depot operator is copying data as it is stored for later analysis. There is no protection against packet snooping on the wire, since logistical networking employs an overlay on the public Internet.

These issues point to the basic fact that logistical networks are untrustworthy. Hughes applies a similar analysis to more traditional storage systems in [10] and reaches the same conclusion.

#### 3.2. End-to-End Security

How, then, are we to store data on logistical networks (or any other storage system) where the basic storage mechanism is untrustworthy? The answer is simple – ensure information security in an end-to-end manner. That is, information must be secured as close as possible to the producer, and unsecured as close as possible to the consumer.

A side effect of end-to-end security is that security considerations do not impact the scalability of the storage system. Indeed, the end-to-end principle elucidated by Reed, Saltzer, and Clark [16] implies that the only way to maintain scalability while providing security is to do so end-to-end.

Logistical networking supports end-to-end security through the LoRS layer of the network storage stack. LoRS offers the following security-oriented services:

- Checksums. LoRS can apply MD5 or SHA-1 checksums to data prior to storing it. This allows the user to ensure that the data was not tampered with during storage or transmission.
- Encryption. LoRS can apply one of DES, AES, or XOR encryption schemes to data prior to storing it. The relevant keys are stored in the exNode so that the

data can be decrypted when it is read. This allows data to be stored without worry on untrusted depots.

- Fragmentation. LoRS can fragment data and store it on multiple depots. This lessens the likelihood that an attacker will be able to compromise an entire data set.

These services may be combined with one another in a variety of ways. Typically, data is fragmented first. Each fragment is then checksummed, and then encrypted. A successful attacker must obtain all fragments of the data and crack the encryption on each one individually. Note that different encryption algorithms can be applied to each fragment, and LoRS is not limited to DES, AES, and XOR encryption, assuming the user provides a suitable encryption routine.

In the spirit of the paranoia expressed in the previous section, one might ask why the LoRS code should be trusted. The answer is two-fold: 1) the LoRS code is open source and can be audited, and 2) users can specify their own encryption routines for execution by LoRS. The truly paranoid also have the option of encrypting their data prior to passing it to LoRS, but this has a number of drawbacks.

The first drawback is purely performance oriented. Encrypting data prior to handing it off to LoRS requires that the entire data set be retrieved prior to decrypting it. This is clearly not optimal for large data sets, and in some cases may not be possible. Allowing LoRS to perform encryption on a block-oriented basis lets data be retrieved piecemeal, with a minimum of extraneous data transfer. Moreover, multiple encryption keys can be used (one per fragment), thereby increasing security.

The second drawback is primarily related to ease of use. If security services are imposed outside of LoRS, then it is incumbent upon the application to take notice and retry. However, the LoRS interface does not provide a mechanism for specifying which replica should be retrieved. Therefore, it is entirely possible that the damaged replica will be retrieved again, wasting time and resources. On the other hand, if LoRS is allowed to impose security services itself, it is able to transparently retry in the event of a failure (such as data tampering); moreover, LoRS is smart enough to reject a replica that has previously resulted in failure.

### 3.3. Security Assurance

Mei, Mancini, and Jajodia have proposed a measure of the security of a distributed storage system in terms of static and dynamic assurance [13]. Briefly, static assurance is the probability that a given data set has not been compromised by a successful attack. Dynamic assurance is an extension of this concept to the movement of data through the storage network. A primary result of their work is that storage systems that utilize fragmentation can be made arbi-

trarily secure simply by fragmenting the file enough. Practically speaking, the number of fragments required to achieve a high assurance can be quite large, and so fragmentation is augmented with encryption. Moreover, encryption provides a second axis of security that must be compromised by an attacker.

### 3.4. Fault Tolerance

While checksums and encryption are adequate to provide for the confidentiality and integrity of data stored on a logistical network, they are not enough to ensure availability in the face of network faults, or more ominously, denial of service attacks. LoRS combats this threat in two ways. The first defense is through fragmentation and replication. The sheer number of storage resources that must be attacked to deny service makes such attacks impractical. Moreover, the mapping from fragments to storage allocations is contained wholly within the exNode, which is generally not publically available. Therefore *all* of the logistical network must be attacked to provide assurance of success.

Additionally, LoRS implements Reed-Solomon coding for fault-tolerance purposes. This allows the user to reconstruct a missing fragment of data from the remaining fragments and special coding blocks. Thus, even if an attacker is able to deny access to a fragment (and all of its replicas), the user may be able to reconstruct the data, negating the attack.

A more low-level denial of service attack can be made against the storage resources themselves simply by requesting all available storage. This attack is unlikely to succeed in the long-term due to the time-limited nature of IBP storage allocations. The attacker is required to maintain a long running process in order to renew storage leases, or risk having the attack dissipate.

### 3.5. Key Management

Much of the security available to logistical networking applications hinges on the exNode data structure. The exNode maintains the mapping of data subsets to allocations, as well as the keys and checksums required to recover that data. Thus it is critically important to protect the exNode.

We begin by noting that if data is not to be shared, i.e., it is produced and consumed by the same user, then protecting the exNode equates to protecting any other (local) file, and the same mechanisms can be employed. It is not until exNodes are shared that protection becomes an issue.

There are three primary issues with respect to sharing exNodes:

1. Ensuring only authorized parties gain access to the exNode.

2. Ensuring parties with access to an exNode are not granted improper rights, e.g., write access.
3. Ensuring that encryption keys are protected.

The first of these devolves into standard access control issues for files. Because the exNode is serialized as an XML file, it can be treated like any other file. Thus, standard access controls apply.

The second issue is dealt with by selectively stripping write or manage IBP capabilities from the exNode prior to dissemination. This allows others to read and access the data addressed by an exNode, but does not allow writing or otherwise manipulating the underlying storage allocations. This is the approach taken by our work in content distribution systems to ensure that malicious parties do not alter publically available data.

The third issue is by far the trickiest. If exNodes are to be distributed over public channels, and data confidentiality is desired, it is necessary to ensure that encryption keys are not leaked. While we don't have a comprehensive answer for the problem of key leakage, we do have some theories. The simplest approach is to always use a secure (e.g., SSL) communications channel for exNode distribution. This should be adequate to keep network snoopers from acquiring the keys. It is not adequate; however, if the receiver's computer is compromised. To combat this eventuality, we recommend encrypting the encryption keys stored in the exNode with the receiver's public key ala PGP [17]. This has the side effect of creating a unique exNode for each recipient. Also, we should note that this method is only as secure as the recipient's private key, and that it doesn't scale terribly well.

## 4. Related Work

There is much related work in the area of distributed storage/file systems. The PASIS system [18] is a decentralized storage system operating purely on data fragments. PASIS makes no provision for encryption of each fragment, however, limiting its security assurance to a single axis.

AFS [14], CODA [3], and CFS [8] are replicated file systems. Attacking these systems requires compromising only a single server. Moreover, as the number of replicas increases, the probability that the data is compromised becomes quite high. Encryption can help, but only one key need be broken.

OceanStore [12] and FarSite [7] are replicated, encrypted storage systems with much the same scope as logistical networking. They suffer from the same limitations as AFS, CODA, and CFS.

Logistical networking differs from these primarily in that it offers both replication and fragmentation, as well as encryption of individual fragments. This allows for complex security structures to be built into the storage layout of a file, achieving high security assurances.

## 5. Conclusion

We have presented an analysis of the security aspects of logistical networking. We have argued that end-to-end security is the only reasonable model, and shown how logistical networking technologies provide end-to-end security services. We have provided some analysis of the information security assurance achievable on a logistical network, and we believe that assurance is a good bit better than that offered by similar projects.

Moreover, we have shown that it is possible to build a highly secure and scalable storage system on top of untrusted storage resources.

## References

- [1] <http://loci.cs.utk.edu>.
- [2] <http://bitconjurer.org/BitTorrent>.
- [3] <http://www.coda.cs.cmu.edu>.
- [4] S. Atchley, M. Beck, H. Hagewood, J. Millar, T. Moore, J. S. Plank, and S. Soltesz. Next generation content distribution using the logistical networking testbed. Technical Report UT-CS-02-498, University of Tennessee, 2002.
- [5] A. Bassi, M. Beck, T. Moore, and J. S. Plank. The logistical backbone: Scalable infrastructure for global data grids. In *Proc. of the Asian Computing Science Conference*, Hanoi, Vietnam, Dec. 2002.
- [6] M. Beck, Y. Ding, E. Fuentes, and S. Kancherla. An exposed approach to reliable multicast in heterogeneous logistical networks. In *Proc. of the Workshop on Grids and Advanced Networks*, Tokyo, Japan, May 2003.
- [7] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proc. Int'l Conf. Measurement and Modeling of Computer Systems*, 2000.
- [8] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proc. 18th ACM Symp. Operating Systems Principles*, 2001.
- [9] J. Ding, J. Huang, S. Liu, T. Moore, and S. Soltesz. Remote visualization by browsing image based databases with logistical networking. In *Proc. SC'03*, Phoenix, AZ, Nov. 2003.
- [10] J. P. Hughes. Stored information security, a systems approach. *StorageTek Insight*, 1(1):2–6, Fall/Winter 1997-98.
- [11] J. P. Hughes and A. Guha. Task force on network storage architecture: Information security. In *Proc. of the Hawaii Int. Conf. on System Sciences*, Maui, USA, Jan. 1997.
- [12] J. Kubiawicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2000.
- [13] A. Mei, L. V. Mancini, and S. Jajodia. Secure dynamic fragment and replica allocation in large-scale distributed file systems. *IEEE Trans. on Parallel and Distributed Systems*, 14(9):885–896, Sept. 2003.

- [14] J. Morris, M. Satyanarayanan, M. Conner, J. Howard, D. Rosenthal, and F. Smith. Andrew: A distributed personal computing environment. *Comm. ACM*, 29(3).
- [15] J. S. Plank, S. Atchley, Y. Ding, and M. Beck. Algorithms for high performance, wide-area, distributed file downloads. Technical Report UT-CS-02=485, University of Tennessee, 2002.
- [16] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. on Computer Systems*, 2(4):277–288, Nov. 1984.
- [17] W. Stallings. *Cryptography and Network Security Principles and Practice*. Prentice-Hall, Upper Saddle River, NJ 07458, second edition, 1999.
- [18] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliccote, and P. K. Khosla. Survivable information storage systems. *Computer*, 33(8):61–68, Aug.