Fault-Tolerance in the Network Storage Stack

Scott Atchley Stephen Soltesz James S. Plank Micah Beck Terry Moore[*]

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

http://loci.cs.utk.edu, [atchley,soltesz,plank,mbeck,tmoore]@cs.utk.edu

IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, April, 2002.

Abstract

This paper addresses the issue of fault-tolerance in applications that make use of network storage. A network storage abstraction called the *Network Storage Stack* is presented, along with its constituent parts. In particular, a data type called the *exNode* is detailed, along with tools that allow it to be used to implement a wide-area, striped and replicated file.

Using these tools, we evaluate the fault-tolerance of several exNode "files," composed of variable-size blocks stored on 14 different machines at five locations throughout the United States. The results demonstrate that while failures in using network storage occur frequently, the tools built on the Network Storage Stack tolerate them gracefully, and with good performance.

1 Introduction

Many commercial and scientific applications need access to remote storage resources, usually very large resources. Ideally, these applications should be able to access these resources with the lowest possible latency and highest possible throughout. Network resources, however, are inherently unreliable. Many factors contribute to the network's unreliability: the remote application is busy or failed, the remote host is down, the remote network is congested or down, the local network is busy or down, etc. Applications that rely on network resources are at the mercy of all of these potential interruptions.

The **Lo**gistical **C**omputing and **I**nternetworking (**LoCI**) Lab at the University of Tennessee has been working to change the view of storage in the network, improving its performance and reliability. As such, The LoCI Lab has been demonstrating the power of *Logistical Networking* in distributed and wide-area settings.

Logistical Networking takes the rather unconventional view that storage can be used to augment data transmission as part of a unified network resource framework, rather than simply being a network-attached resource. The adjective "logistical" is meant to evoke an analogy with military and industrial networks for the movement of material which requires the coscheduling of long haul transportation, storage depots and local transportation as coordinate elements of a single infrastructure [BMP01].

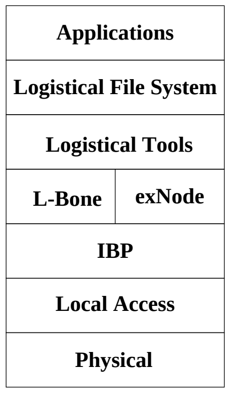

Figure 1: The Network Storage Stack

Our design for the use of network storage revolves around the concept of a *Network Storage Stack* (Figure 1). Its goal is to layer abstractions of network storage to allow storage resources to be part of the wide-area network in an efficient, flexible, sharable and scalable

[*]This material is based upon work supported by the National Science Foundation under grants ACI-9876895, EIA-9975015, EIA-9972889, ANI-9980203, the Department of Energy under the SciDAC/ASCR program, and the University of Tennessee Center for Information Technology Research.

way. Its model, which achieves all these goals for data transmission, is the IP stack, and its guiding principle has been to follow the tenets laid out by End-to-End arguments [SRC84, RSC98]. Two fundamental principles of this layering are that each layer should (a) *abstract* the layers beneath it in a meaningful way, but (b) *expose* an appropriate amount of its own resources so that higher layers may abstract them meaningfully (see [BMP01] for more detail on this approach).

In this paper, we describe the lower levels of the network storage stack, and demonstrate the powerful functionalities that may be achieved under this design. We then focus on the performance of storing and retrieving files on storage depots in the wide-area. In particular, we store striped and replicated files on 14 different storage units in five localities around the United States, and measure the data's availability and retrieval speed. While the tests are small, they show the effectiveness of the design, and demonstrate the power of aggregation of faulty storage units for larger storage needs.

2 The Network Storage Stack

In this section, we describe the middle three layers of the Network Storage Stack. The bottom two layers are simply the hardware and operating system layers of storage. The top two layers, while interesting, are future functionalities to be built when we have have more understanding about the middle layers.

2.1 IBP

The lowest level of the network accessible storage stack is the *Internet Backplane Protocol (IBP)*. [PBB$^+$01] IBP is server daemon software and a client library that allows storage owners to insert their storage into the network, and to allow generic clients to allocate and use this storage. The unit of storage is a time-limited, append-only byte-array. With IBP, byte-array allocation is like a network *malloc()* call – clients request an allocation from a specific IBP storage server (or *depot*), and if successful, are returned trios of cryptographically secure text strings (called *capabilities*) for reading, writing and management. Capabilities may be used by any client in the network, and may be passed freely from client to client, much like a URL.

IBP does its job as a low-level layer in the storage stack. It abstracts away many details of the underlying physical storage layers: block sizes, storage media, control software, etc. However, it also exposes many details of the underlying storage, such as network location, network transience and the ability to fail, so that these may be abstracted more effectively by higher layers in the stack.

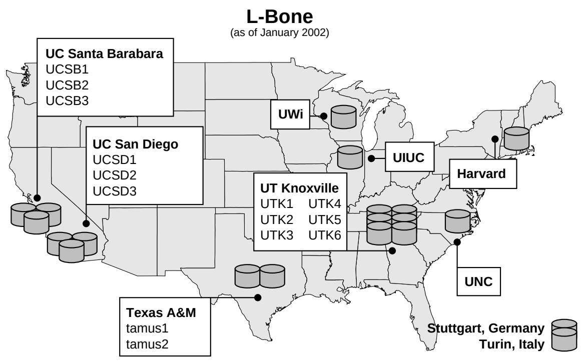

Figure 2: The L-Bone

2.2 The L-Bone and the exNode

While individual IBP allocations may be employed directly by applications for some benefit [PBB$^+$01], they, like IP datagrams, benefit from some higher-layer abstractions. The next layer contains the *L-Bone*, for resource discovery and proximity resolution, and the *exNode*, a data structure for aggregation. Each is defined here.

The L-Bone (Logistical Backbone) is a distributed runtime layer that allows clients to perform IBP depot discovery. IBP depots register themselves with the L-Bone, and clients may then query the L-Bone for depots that have various characteristics, including minimum storage capacity and duration requirements, and basic proximity requirements. For example, clients may request an ordered list of depots that are close to a specified city, airport, US zipcode, or network host. Once the client has a list of IBP depots, she may query the Network Weather Service (NWS) [WSH99] to provide live performance measurements and forecasts and decide how best to use the depots.

Thus, while IBP gives clients access to remote storage resources, it has no features to aid the client in figuring out which storage resources to employ. The L-Bone's job is to provide clients with those features. As of January 2002, the L-Bone is composed of 21 depots in the United States and Europe, serving roughly a terabyte of storage to Logistical Networking applications (Figure 2).

The exNode is is data structure for aggregation, analogous to the Unix inode (Figure 3). Whereas the inode aggregates disk blocks on a single disk volume to compose a file, the exNode aggregates IBP byte-arrays

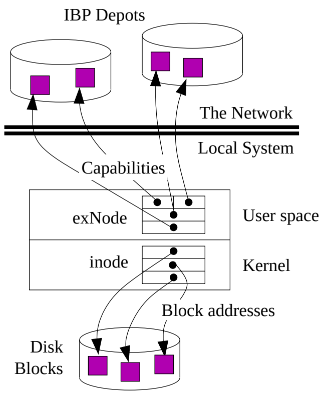

Figure 3: The exNode in comparison to the Unix inode

to compose a logical entity like a file. Two major differences between exNodes and inodes are that the IBP buffers may be of any size, and the extents may overlap and be replicated. For example, Figure 4 shows three exnodes storing a 600-byte file. The leftmost one stores all 600 bytes on IBP depot A. The center one has two replicas of the file one each on depot B and depot C. The rightmost exNode also has two replicas, but the first replica is split into two segments, one on depot A and one on depot D, and the second replica is split into three segments, one each on depots B, C, and D.

In the present context, the key point about the design of the exNode is that it allows us to create storage abstractions with stronger properties, such as a network file, which can be layered over IBP-based storage in a way that is completely consistent with the exposed resource approach.

Since our intent is to use the exNode file abstraction in a number of different applications, we have chosen to express the exNode concretely as an encoding of storage resources (typically IBP capabilities) and associated metadata in XML. Like IBP capabilities, these serializations may be passed from client to client, allowing a great degree of flexibility and sharing of network storage. The use of the exNode by varying applications provides interoperability similar to being attached to the same network file system. The exNode metadata is capable of expressing the following relationships between the file it implements and the storage resources that constitute the data component of the file's state:

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

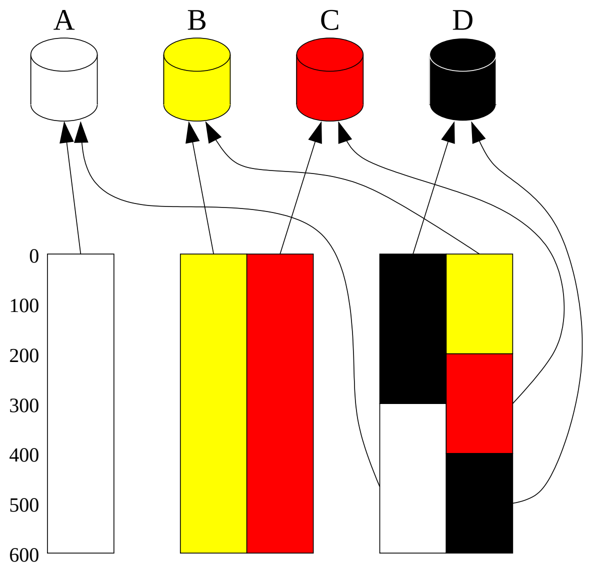

Figure 4: Sample exNodes of a 600-byte file with different replication strategies.

Logistical Tools

At the next level of the Network Storage Stack are tools that perform the actual aggregation of network storage resources, using the lower layers of the Network Stack. These tools take the form of client libraries that perform basic functionalities, and stand-alone programs built on top of the libraries.

Basic functionalities of these tools are:

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

Download: This takes an exNode as input, and downloads a specified region of the file that it represents into local storage. This involves coalescing the replicated fragments of the file, and must deal with the fact that some fragments may be closer to the client than others, and some may not be available (due to time limits, disk failures, and standard network failures). Download is written to check and see if the Network Weather Service [WSH99] is available locally to determine the closest depots. If so, then NWS information is employed to determine the download strategy: The file is broken up into multiple extents, defined at each segment boundary. For example, the rightmost file in Figure 4 will be broken into four extents – (0,199), (200-299), (300-399), and (400-599). Then the download proceeds by retrieving each extent from the closest depot. If the retrieval times out, then the next closest depot is tried, and so on.

If the NWS is not available, then the download looks for static, albeit unoptimal metrics for determining the downloading strategy. If desired, the download may operate in a streaming fashion, so that the client only has to consume small, discrete portions of the file at a time. Currently, the individual retrievals are not threaded so that they may occur in parallel. Adding this capability is future work for the LoCI lab.

List: Much like the Unix **ls** command, this takes an exNode as input and provides a long listing of the stored file's name and size and metadata about each segment or fragment of the file (individual IBP capability sets) including offset, length, available bandwidth and expiration.

Refresh: This takes an exNode as input, and extends time limits of the IBP buffers that compose the file.

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

Trim: This takes an exNode, deletes specified fragments, and returns a new exNode. These fragments may be specified individually, or they may be specified to be those that represent expired IBP allocations. Additionally, the framents may be only deleted from the exNode, and not from IBP.

The Logistical Tools are much more powerful as tools than raw IBP capabilities, since they allow users to aggregate network storage for various reasons:

Capacity: Extremely large files may be made from smaller IBP allocations. It fact, it is not hard to visualize files that are tens of gigabytes in size, split up and scattered around the network.

Striping: By breaking files into small pieces, the pieces may be downloaded simultaneously from multiple IBP depots, which may perform much better than downloading from a single source.

Replication for Caching: By storing files in multiple locations, the performance of downloading may be improved by downloading the closest copy.

Replication for Fault-Tolerance: By storing files in multiple locations, the act of downloading may succeed even if many of the copies are unavailable. Further, by breaking the file up into blocks and storing error correcting blocks calculated from the original blocks (based on parity as in RAID systems [CLG$^+$94] or on Reed-Solomon coding [Pla97]), downloads can be robust to even more complex failure scenarios.

Routing: For the purposes of scheduling, or perhaps changing resource conditions, **augment** and **trim** may be combined to effect a routing of a file from one network location to another. First it is augmented so that it has replicas near the desired location, then it is trimmed so that the old replica are deleted.

Therefore, the Logistical Tools enable users to store data as replicated and striped files in the wide area. The actual best replication strategy — one that achieves the best combination of performance, fault-coverage and resource efficiency in the face of changing network conditions — is a matter of future research. We view this paper as a first step toward that goal.

3 Testing Fault-Tolerance and Performance

To test the ability of the Logistical Tools to tolerate failures, and to test their performance in tolerating failures, we performed three experiments, using a collection of IBP depots from the L-Bone.

3.1 Test 1: Availability of Capabilities in an exNode

For the first test, we monitored the availability of a exNode file over a three day period. We stored a 10 MB file into an exNode that had five replicas, each replica being divided into two to nine fragments. These were stored on six machines in Knoxville TN (UTK 1-6), three in San Diego, CA (UCSD 1-3), three in Santa Barbara, CA

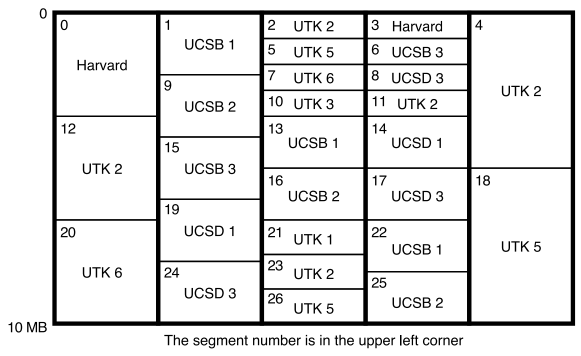

Figure 5: Test 1: The exNode. There are five copies of the 10 MB file, partitioned into 27 segments of differing sizes.

(UCSB 1-3), and one in Cambridge, MA (Harvard). The exact layout is shown in Figure 5.

We ran a test from UTK 1 over three days where we checked the availability by calling **list** on the exNode every 20 seconds. In that time, 16,189 out of the 335,880 fragments checked were unavailable at the time of listing, yielding an overall segment availability of 95.18%. Individual segment availability ranged from 60.51% to 100.00%, and is shown in Figure 6. The figure clearly illustrates that the resources were available for a great majority of the time. However when we look at a snapshot of a single **list**, we see that sometimes multiple segments are unavailable (Figure 7).

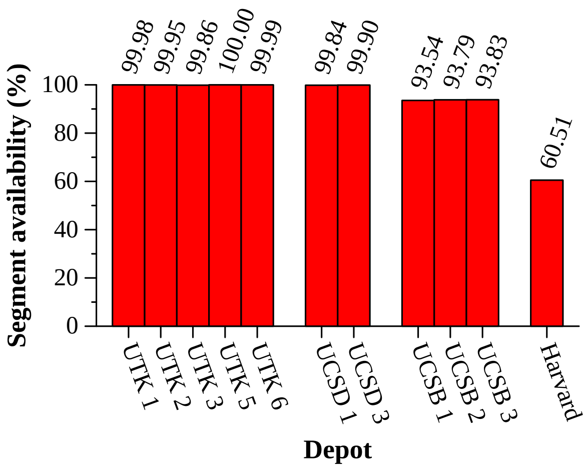

Figure 6: Test 1: The availability of each segment from UTK 1 over three days, testing every 20 seconds (12,400 tests).

This test shows that although individual pieces of the exNode have widely varying availabilities, the exNode as a whole should achieve a high level of availability.

```
UNIX> xnd_ls data10mb.xnd -b
data10mb.xnd: data10mb 10000000
Srwma    0   1     HARVARD    3333333          0    0.80 Jan 11 15:33:48 2002
?rwm-    1  -1      UCSB-1    2000000          0    0.67
Srwma    2   1       UTK-2     833333          0   88.47 Jan 11 15:41:47 2002
Srwma    3   1     HARVARD     833333          0    0.80 Jan 11 15:45:41 2002
Srwma    4   1       UTK-2    5000000          0   88.47 Jan 11 15:52:41 2002
Srwma    5   1       UTK-5     833333     833333   76.60 Jan 11 15:41:47 2002
?rwm-    6  -1      UCSB-3     833333     833333    0.86
Srwma    7   1       UTK-6     833333    1666666   88.76 Jan 11 15:41:47 2002
Srwma    8   1      UCSD-3     833333    1666666    0.64 Jan 11 15:45:41 2002
Srwma    9   1      UCSB-2    2000000    2000000    0.67 Jan 11 15:35:54 2002
Srwma   10   1       UTK-3     833334    2499999   83.98 Jan 11 15:41:47 2002
Srwma   11   1       UTK-2     833334    2499999   88.47 Jan 11 15:45:41 2002
Srwma   12   1       UTK-2    3333333    3333333   88.47 Jan 11 15:33:48 2002
?rwm-   13  -1      UCSB-1    1666666    3333333    0.67
Srwma   14   1      UCSD-1    1666666    3333333    0.64 Jan 11 15:48:25 2002
?rwm-   15  -1      UCSB-3    2000000    4000000    0.86
Srwma   16   1      UCSB-2    1666667    4999999    0.67 Jan 11 15:39:54 2002
Srwma   17   1      UCSD-3    1666667    4999999    0.64 Jan 11 15:48:25 2002
Srwma   18   1       UTK-5    5000000    5000000   76.60 Jan 11 15:52:41 2002
Srwma   19   1      UCSD-1    2000000    6000000    0.64 Jan 11 15:35:54 2002
Srwma   20   1       UTK-6    3333334    6666666   88.76 Jan 11 15:33:48 2002
Srwma   21   1       UTK-1    1111111    6666666 1467.61 Jan 11 15:41:09 2002
?rwm-   22  -1      UCSB-1    1666667    6666666    0.67
Srwma   23   1       UTK-2    1111111    7777777   88.47 Jan 11 15:41:09 2002
Srwma   24   1      UCSD-3    2000000    8000000    0.64 Jan 11 15:35:54 2002
Srwma   25   1      UCSB-2    1666667    8333333    0.67 Jan 11 15:51:24 2002
Srwma   26   1       UTK-5    1111112    8888888   76.60 Jan 11 15:41:09 2002
```

Figure 7: Test 1: Output from one of the **list** calls from UTK 1. Each line represents a segment. A (-1) in the third column indicates that the segment is unavailable. Numbers in the 7th column are NWS bandwidth forecasts, and the subsequent dates are the byte-array expiration dates.

3.2 Test 2: Availability and Download Times to Multiple Sites

For this test, we stored a 3 MB file in an exNode which again had 5 copies and each copy had multiple fragments (Figures 8). The same depots from Test 1 were used, with the addition of a node in Raleigh, NC (UNC). At a five minute interval, we checked the segment availability by performing a **list**, and then we immediately timed a download. This test was run from UTK 1 (Knoxville), UCSD 1 (San Diego), and Harvard (Cambridge) for a three day period.

The availability plots are in Figures 9, 10, and 11, and display some interesting results. Since the majority of segments are at Tennessee, we expect to see the highest availability numbers from UTK 1, and this is the case. Similarly, we expect the availability numbers from UCSD to favor the California depots, however, interestingly there were more network outages from San Diego to Santa Barbara than from Knoxville.

The most surprising result from Figure 11 is that the availability of Harvard segments is so low. The reason is that the Harvard IBP depot went down for a period of time during the tests. The depot has automatic restart as a **cron** job, but during that time, none of the tests could get to the Harvard segments.

Overall, the segment availabilities were as follows:

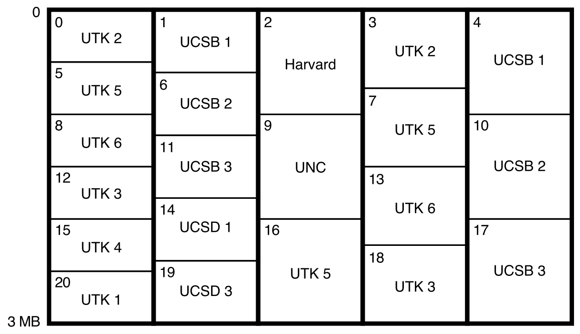

Figure 8: Test 2: The exNode. There are five copies of the 3 MB file, partitioned into 21 segments of differing sizes.

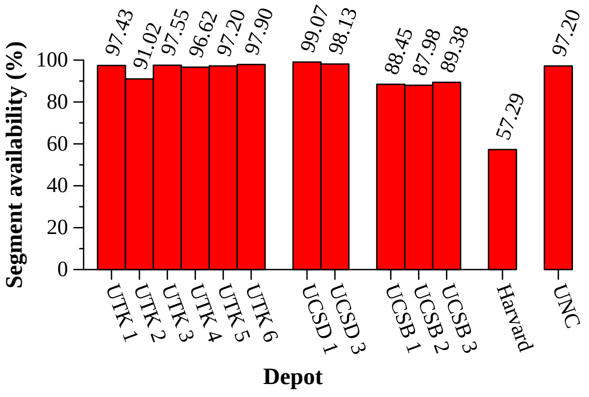

Figure 10: Test 2: Availability from UCSD 1

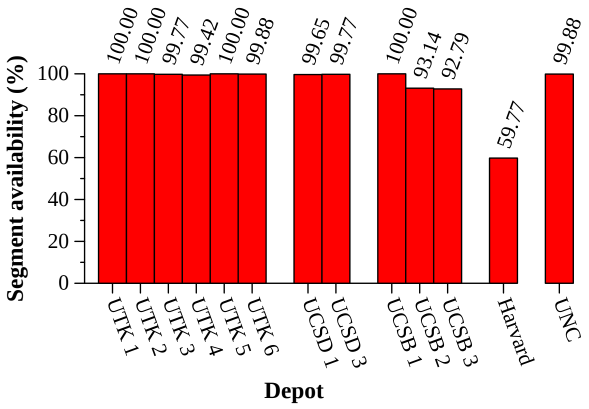

Figure 9: Test 2: Availability from UTK 1

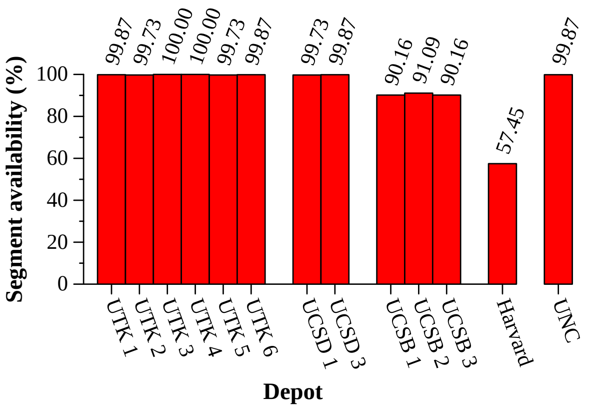

Figure 11: Test 2: Availability from Harvard

94.54% from UTK 1, 90.93% from UCSD 1, and 93.42% from Harvard.

Out of 860 downloads, UTK 1 had 100% success retrieving the file. Since the exNode had two complete copies on the UTK network, most downloads could get the entire file without leaving the UTK campus. Accordingly, UTK 1 saw downloads times between 0.82 and 18.61 seconds with an average download time of 1.29 seconds and a median time of 0.96 seconds. Figure 12 shows the most common download path from UTK 1 — the depot selected at each decision-point is shown in yellow. White lines in the middle of segments show non-obvious decision points of the download. As we would expect, this path shows all depots from Tennessee.

The results from UCSD 1 were similar to UTK 1 with somewhat slower download times.

In 857 attempts, this site also experienced a 100% success rate in downloading the file. UCSD's download times reflect the fact that most of the exNode was stored offsite. Download times ran from 1.87 to 175.38 seconds with an average of 4.38 seconds and a median of 2.63. The most common download path shows that UCSD preferred the UCSD depots, and then UCSB depots, (Figure 13) as would be expected.

Last, although Harvard had better availability numbers than UCSD, it had the worst download performance of all three test sites: A range of 14.77 to 96.87 seconds with an average download time of 28.99 and a median of 27.92 seconds. This is because the majority of its segments came from offsite.

Like the other tests, all of Harvard's downloads completed. The most common download path shows it favoring the Harvard segment for the first third of the file and UNC for the second third (Figure 14). Interestingly, it downloaded the last third from UC Santa Barbara rather than the significantly closer University of Tennessee. Looking at the bandwidth measurements at the end of the test, Harvard was seeing 0.73 Mbits/sec to UCSB and only 0.58 Mbits/sec to UTK.

Clearly, this test shows the benefits of using the exN-

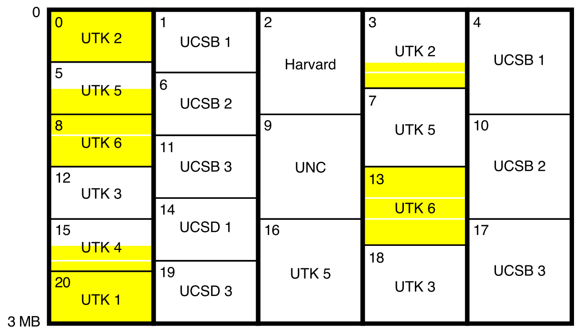

Figure 12: Test 2: Most common download path from UTK

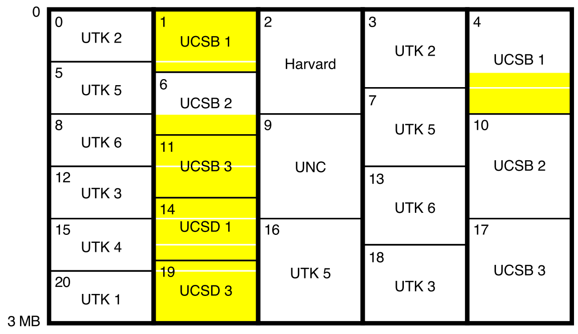

Figure 13: Test 2: Most common download path from UCSD

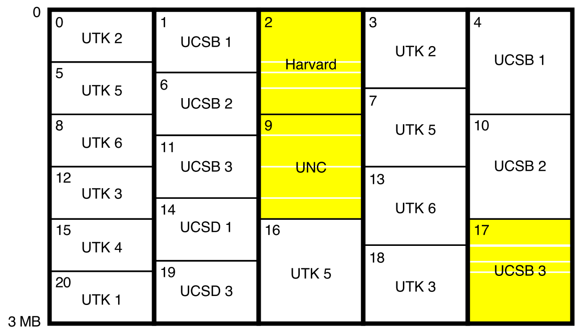

Figure 14: Test 2: Most common download path from Harvard

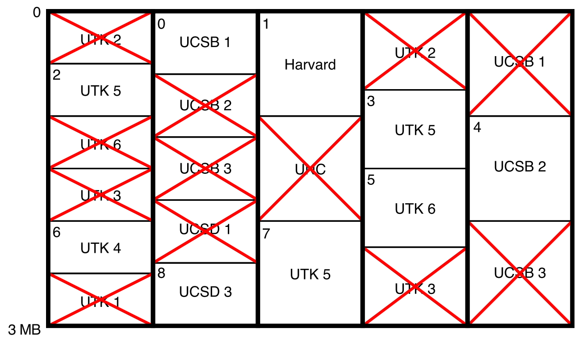

Figure 15: Test 3: The exNode. Now 12 of the 21 segments have had their IBP byte-arrays deleted.

ode to improve fault-tolerance. The Logistical Tools took advantage of the NWS bandwidth forecasts to download from the source with the highest forecasted bandwidth which maximized throughput. Although we did not measure latency directly, the differences in download times between UTK, UCSD and Harvard highlight the importance of having at least one copy of the exNode within the local network for best performance. And most importantly, in over 2,400 downloads, we could always retrieve the file.

Test 3: Simulating Network Unavailability

For the third test, we wanted to test the fault-tolerance of the exNode when we simulated high levels of unavailability. We used the exNode from test 2, but we deleted 12 of the 21 byte-arrays from their IBP depots, in order to simulate a high level of resource failure (machine or network). The resulting exNode (Figure 15) has 33% to 67% of each replica eliminated.

Even with the eliminated segments, there are always at least two possible locations available for the download tool to choose, so in the event that one should fail, even if it were the closer of the two, the other is available for a complete recovery.

From UTK 1, we checked the availability and then downloaded the file every two and half minutes over three days. Similar to test two, we saw individual fragment availability vary from 48.24% to 100%, (Figure 16). On average, Test 3 experienced 92.93% segment availability.

Using this restricted exNode, we were able to download the file 1,150 times before we experienced a download failure. Out of the 1,225 total tests, only the last 75 downloads failed. The first sixth of the file was available only from UCSB 3 and Harvard, which coincidentally had the worst availabilities (93.88% and 48.24%, respectively) of the nine segments. Accordingly, it is reasonable to expect failed downloads of this segment.

The successful download times ran from 3.85 to 36.24 seconds. The average download time was 6.49 seconds and the median time was 6.03 seconds. The longer times are due to the fact that the first sixth of the

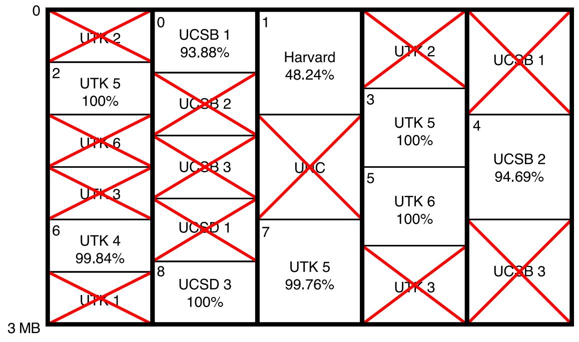

Figure 16: Test 3: Availability from UTK 1

file had to come from California. We display the most common download path in Figure 17.

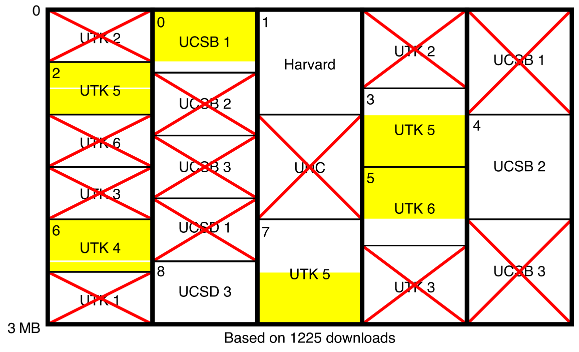

Figure 17: Test 3: Most common download path from UTK 1

This test raises the question of how much replication is enough. In Test 2, we saw that an exNode with five replicas yielded a 100% retrieval rate. Test 3 employed two replicas which allowed for almost a 93% retrieval rate. Ideally, we could always use a high number of replica, but in reality, resources are always limited. Finding the balancing point between the number of replica for greater retrievability versus conserving resources will need to be studied.

Conclusions and Future Work

In this paper, we have described our design of a Network Storage Stack, which provides abstractions and a methodology for applications to make use of storage as a network resource. One result of this design is the ability to store data in a fault-tolerant way on the wide-area network. This work is different from work in distributed file systems (e.g. Coda [SKK+90], Jade [RP93] and Bayou [TTP+95]) in its freedom from the storage's underlying operating system (IBP works on Linux, Solaris, AIX, Mac OS X, and Windows), and its crossing of administrative domains.

The software for IBP, the L-Bone, the exNode and the logistical tools is all publicly available and may be retrieved at `http://loci.cs.utk.edu`. The LoCI lab is especially interested in attracting more L-Bone participants, with the hope that the L-Bone can grow to over a petabyte of publically accessible network storage.

While the work detailed in this paper demonstrates the effectiveness of the methodology and software, it also motivates the need for research on strategies for replication and routing. For example, Test 2 shows an exNode with an excess of replication, whereas Test 3 shows one with too little replication. We view the decision-making of how to replicate, stripe, and route files as falling into the realm of scheduling research, and it is work that we will address in the future.

To further improve fault-tolerance using the Logistical Tools, we intend to investigate the addition of coding blocks as supported entities in exNodes. For example, with parity coding blocks, we can equip the exNodes with the ability to use RAID techniques [CLG+94] to perform fault-tolerant downloads without requiring full replication. To reduce storage needs further, Reed-Solomon coding may be employed as well [Pla97]. Finally, we also intend to add checksums as exNode metadata so that end-to-end guarantees may be made about the integrity of the data stored in IBP. All of these additions are future work for the LoCI lab.

Although not directly a fault-tolerance issue, we will be adding more security features to the exNode and the Logistical Tools. Currently, the data stored in IBP depots are stored in the clear. In the future, exNodes will allow multiple types of encryption so that unencrypted data does not have to travel over the network, or be stored by IBP servers.

Acknowledgements

The authors are grateful to Norman Ramsey for granting access to the Harvard machines, Rich Wolski for granting access to the Santa Barbara machines, and Henri Casanova for granting access to the San Diego machines. Additionally, the authors acknowledge Jim Ding for help in authoring the Logistical Tools, and Alex Bassi and Yong Zheng for authoring the exNode library. Finally, the authors acknowledge Rich Wolski and Jack Dongarra for their vital participation in the Logistical Internetworking and Computing project.

References

[BMP01] M. Beck, T. Moore, and J. S. Plank. Exposed vs. encapsulated approaches to grid service architecture. In *2nd International Workshop on Grid Computing*, Denver, 2001.

[CLG+94] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[PBB+01] J. S. Plank, A. Bassi, M. Beck, T. Moore, D. M. Swany, and R. Wolski. Managing data storage in the network. *IEEE Internet Computing*, 5(5):50–58, September/October 2001.

[Pla97] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[RP93] H. C. Rao and L. L. Peterson. Accessing files in an internet: The jade file system. *IEEE Transactions on Software Engineering*, 19(6), June 1993.

[RSC98] D. P. Reed, J. H. Saltzer, and D. D. Clark. Comment on active networking and end-to-end arguments. *IEEE Network*, 12(3):69–71, 1998.

[SKK+90] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems,*, 2(4):277–288, November 1984.

[TTP+95] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *15th Symposium on Operating Systems Principles*, pages 172–183. ACM, December 1995.

[WSH99] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, 1999.