A basic framework and overview of a network-based RAID-like distributed back-up system: NetRAID

by

Christopher Allan Colvin

A thesis submitted to the graduate faculty in partial

fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Assurance

Program of Study Committee:
Anthony Townsend, Major Professor
Kevin Scheibe
Thomas Daniels

Iowa State University

Ames, Iowa

2005

Graduate College
Iowa State University

This is to certify that the master's thesis of
Christopher Allan Colvin
has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

Table of Contents

LIST OF FIGURES

ABSTRACT

NetRAID is a framework for a simple, open, and free system to allow end-users to have the capacity to create a geographically distributed, secure, redundant system that will provide end-users with the capacity to back up important data. NetRAID is designed to be lightweight, cross-platform, low cost, extendable, and simple.

As more important data becomes digitalized it is critical for even average home computer users to be able to ensure that their data is secure. Even for people with DVD burners that back up their data weekly, if the back ups and their sources are kept in the same physical location the value of the back-up is greatly diminished. NetRAID can offer a more comprehensive end-user back-up.

NetRAID version 1 has some limitations with the types and speeds of networks it can run on; however, it provides a building block for the future extension to almost any sort of TCP/IP network. NetRAID also has the potential capability to use a wide variety of encryption and data verification schemes to make sure that data is secure in transmission and storage. The NetRAID virtual file system, sockets, and program core are written in Visual Basic.NET 2003, and should be portable to a wide variety of operating systems and languages in the future.

1 NetRAID Background

1.1 Design Goal

The design goal for the NetRAID system is to create a framework for a simple, open, and free system to allow end-users to have the capacity to create a geographically distributed, secure, redundant system that will provide end-users with the capacity to back up important data. Unlike AFS [3], InterMezzo [20], Microsoft DFS, or other distributed file systems [25] NetRAID is designed to be lightweight, cross-platform, low cost (free), extendable, and relatively simple. NetRAID trades speed for the ability of wide distribution, and normal random access for reliability. NetRAID is not intended to compete directly with other distributed systems since it is a back-up medium only.

1.1.1 Why Distributed Back-up?

Even in well written and comprehensive books on disaster and data recovery like Cougias [12] and Toigo [45] there is a lack of material relevant to average computer users and small businesses. These books focus on how large organizations manage and leverage services provided by other large organizations such as EMC and IBM. They discuss ways to store petabytes of data in remote tape libraries and in other large buildings. These methods and hardware systems are not relevant or affordable to the vast majority of computer users on the internet even if they had the advanced training to operate such systems.

As society becomes increasingly dependent on computers and their data files, the loss of critical information such as tax information, email, photos, and any other important documents can cause serious financial or emotional damage. For several years now large and even medium sized businesses and organizations have had the resources to maintain

complex, sometimes geographically diverse data back-ups for the preceding several years. However, easy back-ups of this type are still beyond the means of most of NetRAID's target end users.

1.1.2 NetRAID's End-users

Free market competition has helped to create an environment where writable CD and DVD drives are common in all new PCs, and inexpensive to acquire for existing systems. Their cheap, fast back-ups make it possible for end-users to back up files at little or no cost. If the back-ups and their sources are kept in the same physical location, however, the value of the back up is greatly diminished. NetRAID can help fill in this deficiency with a more comprehensive end-user back up.

NetRAID's target user is a computer user at home or with a small business. These users maintain one or two computers and are probably primarily Windows or Mac OS X users. Common software packages include a word processor, email, spreadsheet, light-weight database, basic accounting software, and some graphics software. The files they need to back up and secure are typically in the range of 1KB to 15MB in size. They are connected to the internet via DSL, Cable, or possibly a T1 or higher LAN. NetRAID can, in theory, also work over modems.

This profile has been further refined through an informal survey of selected users from among Iowa State University undergraduates and alumni. Those that had data for which NetRAID would be an appropriate back-up system had about 100-200MB of data in roughly 1-50MB chunks. Several of the respondents said they did not trust computers to preserve important information, and kept important data on paper only. If these respondents

were to encode their information with Portable Document Format (PDF) and archive it, the resulting electronic files would fall into the same range of data sizes as the respondents with files to back up.

1.2 Visual Basic.NET 2003 Development Environment

When NetRAID was still in its planning stage, it became necessary to decide regarding what language to write it in. The goal was a cross-platform system. In the end this did not happen, but a port to UNIX (Mac OS X, Linux, BSD, and Sun's Solaris) operating systems should be fairly easy to accomplish. For cross platform ease, Java, C, C++ or now even Microsoft's C# (via Mono [26]) would have been more ideal languages to use. GCC, Java, and .NET all allow for relatively easy porting between several operating environments.

Visual Basic.NET was in the end was selected for several reasons. VB.NET was the easiest to code given the skills of the programmer. The availability of a wide variety of sample code and excellent guide and documentation books, *Microsoft Visual Basic.NET Step by Step* by Halvorson [17], and Kurniawan and Neward's *VB.NET Core Classes in a Nutshell: A Desktop Reference* [22] were also major factors. Visual Basic and C# share much of the same syntax and the .NET runtime is language neutral. A future conversion from Visual Basic .NET to C# should be straight forward. Currently there are also several active efforts to extend Mono and other packages to support Visual Basic compatible languages, so conversion may not even be an issue. Also greater weight has been given to proving that NetRAID can work, than to developing optimized or even efficient code.

One of Microsoft's goals with .NET was to make computing more secure, and while their efforts on a broad scale are debatable, the newer cryptography and hash generation APIs

are much easier to program with and have broader support than the older ones. A comparison of Bondi's *Cryptography for Visual Basic: A Programmer's Guide to the Microsoft CryptioAPI* [8] which is based on Visual Basic 6, with LaMacchia's *.NET Framework Security* [23], based on the .NET languages demonstrates with its examples the relative simplification of cryptography programming from the old to the new environment.

1.3 Trademark Information

The term 'NetRAID' resulted from several discussions with fellow students. The participants in these discussions were unaware that Hewlett Packard Corporation had been using the NetRAID trademark on a series of high end server SCSI RAID adapters during the 1990s and into the early 2000s. After the initial NetRAID paper was written, "NetRAID" was Googled to see if the search engine had picked up on the paper. This led to the discovery of HP's product. NetRAID's website was modified to clarify the fact that HP apparently held the trademark on the term, but that the paper was not going to re-written unless HP requested it.

Once NetRAID was decided on as a thesis project, HP's Intellectual Property Licensing Department was contacted via the HP website. A description of the project was submitted and a request was made to determine if a license could be acquired for the name, or if HP would protect its trademark. A week later HP replied that it was no longer using the term and it had no opposition to the term's use. The reply text is located in the appendices in section A2.

1.3.1 NetRAID licensing Information

NetRAID will be released to the public under some sort of open source license. The exact license will be determined after the thesis defense. Open source development is still a bit like the Wild West in the software development world, but it is growing. Pavlicek's book [32] is an excellent guide to the broader question raised by how to develop open source software. Open source development [27] using the .NET platform is growing, and NetRAID will be on the front end of that trend.

2 Review of Past Distributed File Systems and Back-up Methods

2.1 Distributed Systems

Distributed systems date back to the early days of computer networks. The basic idea behind a distributed system is to spread the data or computations across many machines on either a local or wide area network. Most distributed systems tend to focus on distributed processing. Beowulf [38] and Seti@home [5] are two of the best current examples of low cost, distributed computing. Distributed systems can be used for data storage as well. With the emergence of the internet, the parts of a distributed file system can be located almost anywhere in the world.

2.2 Distributed Computation

2.2.1 Databases

One of the most common uses for distributed systems is databases. While distributed databases [21] are not quite the same as a true distributed file system, they are common, and share many of the same properties. As computer systems proliferated, it became easier for organizations to compile data for analyses (sales, personnel, payroll, etc) on a local server and they all have the local computer interact with a mainframe or cluster at a central data repository. The DataBase Management System (DBMS) has to maintain the integrity and transparency of data across the internet. The DBMS must also keep the data synchronized across the database nodes. XML has been revolutionizing distributed data bases over the last several years.

2.2.2 Distributed Operating Systems

There are even a few distributed operating systems [37]. These use distributed file systems and often shared memory and CPU operations. Amoeba [4] was started in 1981 by Andrew Tannenbaum as a research project. It is a "pure" distributed system, where there is very little concept of a local machine. V-System is another distributed operating system made up of Sun and VAX systems. It is no longer being developed or in use, but a project using V developed the W windowing system. W was one of the main sources of influence and code for early versions of the modern X Windows system. Distributed operating systems are extremely complex systems, much more difficult to program and maintain than either distributed file systems or operating systems.

2.3 Distributed File Systems

Levy and Silberschatz [25] wrote an excellent definition of a distributed file system. NetRAID fulfills all of the requirements they specify:

> A DFS is a file system, whose clients, servers, and storage devices are dispersed among the machines of a distributed system. Accordingly, service activity has to be carried out across the network, and instead of a single centralized data repository there are multiple and independent storage devices. As will become evident, the concrete configuration and implementation of a DFS may vary. There are configurations where servers run on dedicated machines, as well as configurations where a machine can be both a server and a client. A DFS can be implemented as part of a distributed operating system or, alternatively, by a software layer whose task is to manage the

communication between conventional operating systems and file systems. The distinctive features of a DFS are the multiplicity and autonomy of clients and servers in the system.

2.4 Similar Systems to NetRAID

William J. Bolosky and John R. Douceur and David Ely and Marvin Theimer's [7] 2000 paper, Feasibility of a Sever-less Distributed File System Deployed on an Existing Set of Desktop PCs describes a system that has several similarities to NetRAID. It is decentralized, provides for redundancy, and uses encryption for privacy. The authors worked for Microsoft, and there is very little information on how they wrote the system, let alone the full source as NetRAID provides to the public. Their system took advantage of disk free space on Windows 98 and NT workstations. Their system was designed only for LAN use, and their paper never discussed what types of data they wanted their system to store.

2.5 Andrew File System

The Andrew File System is a distributed file system that Carnegie-Mellon University started developing in 1983. AFS is designed to work on a large scale. Deployments can involve thousands of computers. The file system is divided into two parts, the shared name space and the local name space. Each workstation has a local name space for it to hold programs and local settings. All systems are connected by LAN. The shared space is maintained by servers and uses a location-transparent file hierarchy. The physical location of data in the AFS system can change even while the information is in use. AFS allows any system user to access any file from any workstation. There are even provisions for providing transparent back-up by creating read-only copies of AFS cells. These are very useful in large

networks and for systems of computer labs. AFS is a scalable, mature, and robust file system.

The Andrew File System is a very large, complex system, and requires integrated Kerberos authentication. AFS is designed for a much, much heaver load than NetRAID, and allows for random access. While these traits make AFS an excellent distributed file system, they also make AFS too difficult for an average computer user or administer to set up and maintain. NetRAID is designed to be simple and work well with small groups.

2.6 Other Distributed File Systems

Frangipani [44] is a DFS for UNIX that uses a virtual disk system based on Petal to provide for a large single storage area to multiple users. Petal provides for lower level redundancy and file storage and the Frangipani server allows for many Petal servers to be tied together. Frangipani is designed for use on a trusted net cluster environment. Frangipani is designed to look like any other device in UNIX. xFS and Zebra [6] used RAID striping to provide a distributed file system that is similar to NetRAID in their use of RAID logic. Petal [24] provides similar services in abstracting the network logic from the storage logic to these distributed file systems as NetRAID's virtual file system provides to NetRAID. These systems were designed with LANs and other high speed networks. NetRAID is targeted towards slower internet based WANs.

2.7 Back-up Systems

The earliest use of distributed file systems was when researchers and system administrators would take their back-up tapes home with them and store them in their basement, or in other buildings on their campus. This approach has little in common with a

modern DFS. It was a very low tech and inflexible way to distribute data to protect against the failure of one geographical location. Cooper and Garcia-Molina [11] go into great detail about data replication and the best methods for distribution and replication.

2.7.1 Large System Back-ups

Large computer networks face very complex back up problems. A modern corporate network may contain several terabytes of storage. Due to state and federal regulations suck as Sarbanes-Oxley many organizations have to keep detailed backups of data for many years in case of investigation or auditing. Stringfellow's book [40] about Sun based Network backups in good example. While the systems discussed are a few years old, the systems involved are similar. The best long term backup for large amounts of data is tape. One of the biggest problems with tape is the bandwidth needed to get the data across the network to the tape machine, and then the speed of writing the tape. For online back-up, a solution such as SAN or large RAIDs is becoming more common. IBM and EMC make high quality storage systems that can store many dozens of terabytes. As clustering technology improves, these systems will continue to scale up to meet large companies with ample cash's needs.

Storage Area Networks are also an excellent technology to implement large storage systems inside a corporate network. They can be implemented using a variety of hardware and protocols. SANs only work on local networks, and they require a large amount of network bandwidth. They can be very expensive to setup and require a knowledgeable staff to maintain.

2.7.2 Home System Back-ups

Quality back ups for home users is the niche where NetRAID can make a difference. Tape drives have tended to be too expensive or complex for most home users. The best method for an average home user was to implement a RAID. Until recently this was expensive a slow. Hughes and Murray's [19] paper goes into detail about modern serial ATA drives and RAID. Home RAIDs are still relatively expensive, and do not offer off-site portability. Currently the best all around technology for home back-ups are CD and DVDs. Double layer DVDs hold over 8 GB and with compression they can hold significantly more actual data. These drives and media have come down considerably in price relative to storage size, and the software to run them has become easier to use and more functional. DVDs or CDs allow the user to send them in the mail to trusted individuals. They also allow for easy the creation of duplicates for further redundancy.

3 NetRAID v1 Framework Overview

3.1 NetRAID is not RAID

Patterson, Gibson, and Katz [31] in their "A Case for Redundant Arrays of Inexpensive Disks" explained how to use local disks to improve a computer's ability to store more data and keep it safe. NetRAID builds on RAID's reliability, but not its striping capability. NetRAID's real constraint is bandwidth.

NetRAID nodes communicate over the internet, not a local bus, so the bandwidth can be very low. Even users with good DSL or cable connections can only rely on a few hundred kilobits per second. NetRAID users do not yet have the speed (1 – 3 Mb/s) that hard drives in the late 1980s had when Patterson, Gibson, and Katz wrote their paper. The internet also provides an unstable, best effort bus instead of the dedicated mainframe storage they were working with. Furthermore current drives based on ATA, SCSI, USB, or FireWire can transfer hundreds or even thousands of megabits per second. For higher efficiency NetRAID needs to minimize the amount of data transferred and have all transfers to be independent of each other. If the NetRAID cluster is implemented on a LAN instead of the internet, the higher bandwidth will allow for a much more rapid transfer of data.

3.2 What is NetRAID?

NetRAID is a peer-to-peer system. Each member of a cluster has two functions: client and server. NetRAID's client-side code is a collection of widgets and tools implemented with a GUI that allows each user to control their files and set their preferences.

The NetRAID server is the code that runs NetRAID's network sockets and manages the communication between nodes. The server also controls access to the storage areas in the

cluster. The storage components are a virtual file system that stores the information. Because the system is modular these components are easier to maintain and improve.

The virtual file system may or may not reside on the same computer as the node, but all network communications run though one point. The nodes are numbered, so if one is offline, the other nodes can move on to the next box. This allows for the loss of a node without the long-term loss any of the data or functionality of the storage nodes. The only delay is the time needed for the clients to detect the absence of the primary node and revert to a secondary. Given the information duplication of N, the cluster can lose up to N-1 nodes without losing data. If the information is duplicated three times, two nodes can be lost and the information can still be regenerated. The original vision of NetRAID designated separate storage and server nodes, but as research and coding began, it became apparent that the original proposition that all nodes provide both server and storage would be fairly easy to implement.

The storage functionality of the server in a NetRAID cluster does exactly what its name suggests: store data. Since, in theory, NetRAID works independently of the operating system, the details of the implementation of the storage on each client are left up to the specifics of that environment and file system. Currently NetRAID only exists in the .NET environment, so no variant has been developed. In the version one model, each node allocates a set amount of space on the local drive. A virtual file system, NRDFS (Net RaiD File System or "nerd-fs") is created and a local database is built. This allows for the reconstruction of a node based on information in each of the other nodes. The local file system knows what it contains, and how to retrieve it, but not necessarily how to read it. A determined local user might possibly access the local nrdfs node, but should be unable to

read individual files. Since each node is maintained by a user who is trusted to have access to the cluster's nrdfs, this problem is minimal.

3.3 Example Use of NetRAID #1

Consider a NetRAID cluster set up for Alice, Bob, Carol, Dave, and Eve (Figure 1). All systems are personal computers using Windows with .NET 1.1 runtime or higher. All users have administrator access to their own boxes and trust each other to store data, but do not want the others to read it. Each node is 1 GB, and no files larger than 20MB need to be backed up across the system.

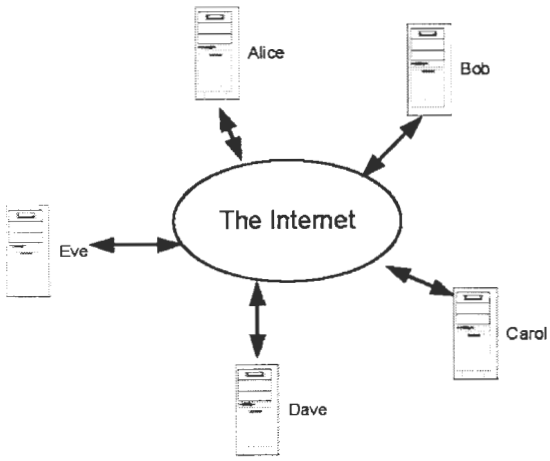

Figure 1. A sample NetRAID cluster

3.3.1 Adding a file to the cluster

Alice backs up her important files every day to CD and flash drive. She decides that she also needs to back up a term paper to the NetRAID cluster she created with her friends in order to make sure that she does not lose all of her work. The first version of NetRAID does

not support uploading multiple files as part of a single job or the use of folders. Because of this, Alice will use an archiving program with support for such formats as ZIP, RAR, or TAR to place all data and their relative hierarchies into a single file. To help conserve space, she uses compression [18] [35]. Even though NetRAID uses AES encryption for transmission, she further secures her information with her own favorite encryption method using the archiving program or stand-alone encryption software.

Alice then opens her node control program and adds the file into the system. The local node records the file's vital information, generates a hash for later integrity verification and then breaks the file up into chunks. The computer selects three other nodes, and then transmits. The receiving nodes then compare hashes and place the file into their virtual file systems. A message is passed back to the source node verifying that all tasks have completed correctly. Since everyone in Alice's cluster is online and they all have high speed DSL, this takes only a couple of minutes to complete. Confident that her paper cannot be lost, Alice gets back to work.

3.3.2 Deleting a File.

Now that the file is in the system there are two options for Alice: she can delete it or retrieve it. Deletion is simple. Alice opens up her client software, finds her file in her client interface, and deletes it. The client knows where the copies of the files are being stored and tells the appropriate nodes to remove their copies by sending a file deletion message. Once this is done, the clusters will send back an OK message to Alice.

3.3.3 Retrieving Files

File retrieval is also is also relatively simple. First, Alice will open her file control menu and select the list of files she has stored in the system. Her node then selects a machine to retrieve the data from and then requests the file. After Alice's node has received an end-of-file message it compares the information it has retrieved to the hash she generated when she stored the file. If they match, the retrieved file is then stored in her local file system in the NetRAID program folder. Alice can then decrypt and decompress her data using her own software.

3.3.4 Updating a File

Alice can update her file by uploading a new copy and then deleting the old one. This might be considered a waste of bandwidth, but it is simple and straight forward.

3.3.5 Repairing the System

Suppose that one night Alice's house suffers a small natural gas explosion that destroys her computer room. Her computers are destroyed, and all of her local back-ups melt in the resulting fire. The next day Alice buys a new computer and plugs it into her broadband connection. She calls up Bob who reads her the group information she needs to rewrite her configuration file using a simple text editor. This only takes a minute or two. She then opens NetRAID, has it build the local virtual file system, and then clicks on "rebuild node." The regeneration happens in two phases.

Just as when the cluster was initialized, the local client code creates the virtual disk and formats it. This takes a couple of minutes. Every cluster then generates a list of the files to be repaired that were on Alice's node and the location of other copies of the files. This list

gets parsed and her client code begins to download and rebuild her node. This makes sure that the data that everyone else had stored on Alice's computer is rebuilt in case a different node is somehow lost. The time this takes will depend on the internet connections that the cluster has at its disposal. Once the node is rebuilt, the cluster can begin to work normally again.

Alice also now has access to all of the files that she stored in the cluster and can retrieve them as quickly as the network connections can move the data. She will have all of her documents, photographs, financial data, and the term paper she backed up the night before.

3.4 Example Use of NetRAID #2

A small company, ACME Corporation, has five retail locations in a region. There is one small central office at the first store with several workstations for use by the product designers and a secretary. All five locations have broadband internet connections. The owner bought an extra four older computers inexpensively at a .com liquidation sale. He built a NetRAID cluster out of them and placed his primary server at the original store. He runs a little script that backs up his most important data files to his cluster every night.

One morning a nasty worm comes in through an email and infects all three computers in the main store office. The ISP immediately detects the worm and quickly shuts off the connection. The server uses mirrored hard drives for back-up, but the consultant brought in to help fix the problem does not realize this and formats both disks. ACME's DVD back-ups are worthless because the burner broke two weeks ago and ACME's staff was waiting for the manufacturer to send them the warranty mandated replacement.

ACME's owner looks up at his "Don't Panic" poster and realizes that while it hurt to lose everything on the office server, all he needed to do was boot up a PC, install NetRAID, type in the configuration file, and tell it to rebuild and retrieve his files. NetRAID's distributed nature prevented the loss of the company's most important data.

3.5 Example Use of NetRAID #3

NetRAID doesn't necessary have to be used for wide-area distributed back-up. With a little modification it could be extended to act much like a current RAID system across several inexpensive, basic machines. A good example would be a small media company or organization that produces videos, music, or print ads and needs a large amount of short term back-up space for projects its employees are working on before the finished products are archived.

This small media firm buys five beige-box machines, each with a one terabyte RAID 5 setup. The firm has a tight budget so it has to compromise a bit on quality to get the desired size. Linux is installed and NetRAID is configured across the boxes. When a single disk goes bad the administer can then replace it and regenerate the local RAID. When a box dies, the administrator can regenerate the NetRAID cluster. By sacrificing NetRAID's wide area capability and then increasing its local bandwidth, the firm now has a large online back-up cluster with only a small cash layout.

4 NetRAID Implementation

NetRAID is made up of four primary components: the NetRAID file system, the network layer, the program core, and the user interface. These different threads are held together by the network layer and a shared message stack. The virtual file system is composed of four files: the virtual disk, the local FAT, the user FAT, and the index. Each has a specific job and structure.

NetRAID is best implemented using groups of individuals that can trust each other to store data, but not necessarily to read it. NetRAID framework version 1, as described in this paper, uses a few assumptions to simplify its use. The first is that all nodes are using a fast network connection that is always on and reliable (i.e., not a modem). The second is that the machines are always on. Furthermore, the NetRAID framework should be flexible enough to allow for the users to plug in many different types of clients, encryption, and network interfaces. As security theory and practice evolves, the ability to adapt quickly will make NetRAID much more useful.

While NetRAID's redundancy level can be set by its users, the default level in version one is a five node cluster with three copies of the file in the system. If N equals the number of copies of information in the system, NetRAID can lose N-1 nodes without losing any information. In version one, two nodes can be lost and then rebuilt without losing any information in the system. Just as with classic RAIDs, redundancy comes with a loss of speed and space.

4.1 NetRAID Virtual File System (NRDFS)

The virtual file system, nrdfs, is meant to be very simple and is designed for speed and security. The initial drive is allocated by the client software. How this is implemented depends on the local operating system. All nrdfs requires is a file ID, source node, file size, date and time, and what blocks the file uses. This allows the system to retrieve and send files around the system, even though local user lacks the ability to read them. The local user's nrdfs also maintains a directory structure of the local user's files. The table needs file ID, file name, node1, node2, node3, size, and hash information.

The first version of nrdfs is based on a simple FAT type of file system. This system is very simple and can be inefficient, but it does meet the essential requirements of an early implementation. During testing the reading and writing from the data store was very quick on the 500MHz laptop that most of NetRAID was written on. The primary data bottleneck in NetRAID is the network bandwidth, not the local system I/O capability. The Visual Basic .NET code for implementing and running nrdfs one is relatively simple and should be easy to maintain and optimize. Since NetRAID is designed to be modular, it should be easy to swap in a different nrdfs system that conforms to the same APIs and be transparent to the network layer. Different implementations of NetRAID should even be able to use different node file systems, as long as the network layer remains compatible.

4.1.1 Why a virtual file system?

NetRAID is designed to be a simple, lightweight, robust, open, and cross platform system. Implementing a virtual file system helps to accomplish all of those goals. .NET, GTK+ [46], Sun's Java, and other toolkits allow for more portable code. By not using OS

specific calls to the local data repository that are file system specific, NetRAID remains more portable. By using .NET's file classes, NetRAID is able to run on any platform that .NET supports with no platform specific changes. In some environments NetRAID might itself be on a RAID or a network share. During testing it was even implemented on removable Jazz disk. The Virtual File System also allows the host operating system to manage and optimize its placement on the native file system. The primary references for writing Nrdfs was Tanenbaum's chapter [42] on file systems, Grosshans' File Systems [16] and Dennis M. Ritchie and Ken Thompson [34].

Just as with other parts of NetRAID, there is no need to implement extra complexity in NetRAID if low level functionally can be offloaded to the expert programmers at Microsoft, Sun, or an open group. They have worked hard to write their software, so there is no reason for NetRAID to reinvent the wheel. This reliance on a specific toolkit also has a software life cycle justification. The developers of these tool kits are supposed to maintain and improve their code. During the initial NetRAID development the environment was changed from .NET 1.0sp2 to 1.1sp1. Since the code was written using standard .NET functions and APIs, NetRAID was able to benefit from Microsoft's work on performance and security without modifying a single line of NetRAID code. Relying on such a specific, widely used and well maintained development framework should improve portability and development. There is no reason that the NetRAID framework cannot be implemented with different toolkits on different environments.

4.1.2 NRDFS Virtual Disk

The virtual disk is programmed as a very simple binary stream object in .NET. The .NET framework runtime does most of the work. The node is created by writing binary 0s to a file that is of size CLUSTER_SIZE * CLUSTERS. These constants are determined by the user's configuration settings. The size can be manipulated by changing a constant in the code, or eventually a number in the configuration file. There are no complex structures. To find a location in the file, the binary reader or writer is directed to go to the CLUSTER_SIZE * ClusterID and then read or write CLUSTER_SIZE amount of data. The data is then read into or written from a binary array, and those are dumped into a stack for movement around the program.

4.1.3 Directory Support

The first version of the NetRAID framework does not support a hierarchal folder system. This would add extra complexity to the system and is not needed at this time. End users should really use a compression/archiving program to wrap up their folders and make them as small as possible. A good example of this is backing up NetRAID's source folders. When it is archived, it is a lot smaller, and just one file as opposed to lots of text and binary files in quite a few folders. Most archiving software support some sort of encryption to provide extra security, or a single file can easily be encrypted by a wide variety of tools, free or not, that can encrypt files.

Why not just put them in a specific folder? For the first version, this would seem be a faster and simpler plan. However, there are reasons to implement a virtual disk this early in development.

The virtual disk helps to hide the data from the local node's owner. One of the core premises of NetRAID is that Alice can trust Bob to store her data, but not to read it. If Bob can look in a NetRAID\Stored folder and see files, it is very tempting for him to open them up. If he has to write his own binary reader and Local FAT information parser before he can get to the data, it will slow him down. Allocating all of the drive space at once should reduce host file system fragmentation. VMWare also has an option to either dynamically allocate space for virtual disks, or allocation all at once.

4.1.4 Local FAT Table

The local virtual file allocation table (Figure 2) is implemented in VB.NET as a structure. It is fairly simple, but does hold more data than a typical FAT on a disk such as the SHA512 [33] hash value. The data is all either strings or integers and can be read or written very easily using a binary stream object.

```
Public Structure LocalFAT
    '//The File ID
    Public FileID As Int16
    '//The first Cluster allocated
    Public FirstCluster As Int32
    '//who's file it is
    Public Source As Byte
    Public FileSize As Int32
    Public FileName As String
    Public DateAndTime As String
    Public Hash As String

End Structure
```

Figure 2. Local Virtual File Allocation Table Code

- The File ID is based on the combination of the source ID and a random number to mitigate the problem of conflicting file numbers in the system. First, NetRAID generates a random number, and then multiplies it by 1000. To attach the source, it

then multiplies the number by ten and adds the source number. In version one the file ID is 16 bytes. This limits NetRAID to only hold about 64,000 files. This might seem like a cap on the system, but it shouldn't be. NetRAID is not designed it back up thousands of little files, it is designed to work with a few hundred larger files. The FileID could be expanded to 32 bytes at a future date with out too much trouble if it is needed. The current limiting factor is the regeneration code, version one can only regenerate about 300 files.

- The FileName is the name of the file input into NetRAID.
- The FileSize is the integer size of the file in bytes. This is needed to truncate the extra junk at the end of the cluster of the last segment of the file. The first cluster is the cluster where the file starts. When the file is being read back from the virtual file system there is a function that will make figure out the next cluster given the first cluster until the end of the file is reached. If the file allocation system is looked at as a database, this is where the FAT table is linked to the Index table.
- The hash is the SHA512 hash of the file to verify integrity.
- Source identifies the file origin.
- DateAndTime is the timestamp of when the file was added into the NetRAID system.
- The file name is recorded just incase the original source node needs to be rebuilt

4.1.5 User FAT Table

The User table (Figure 3) is very similar to the local FAT table. The difference is the lack of a start cluster since the UserFAT does not address disk space, but instead the network locations. The hosts are the three network locations of the file. These are implemented as

data type byte since a byte takes up the least amount of room in memory. NetRAID is not designed to have more than the 256 computers that the data type supports.

```
Public Structure UserFAT

    Public FileID As Int16
    Public FileName As String
    '//size in bytes
    Public FileSize As Int32
    '//SHA2 Hash
    Public Hash As String
    Public DateandTime As String
    Public Host1 As Byte
    Public Host2 As Byte
    Public Host3 As Byte

End Structure
```

FIGURE 3. USER VIRTUAL FILE ALLOCATION TABLE CODE

4.1.6 FILE INDEX

The File Index (Figure 4) is very simple. It gives the cluster number, if it is in use, and if it is in use then the address to the next cluster. If there is no next cluster, there is a 0 value.

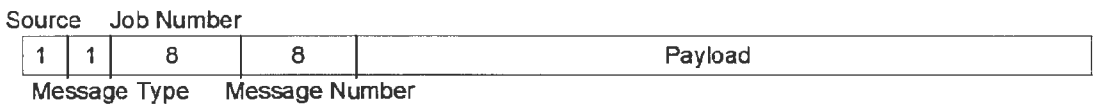

Figure 4. A NetRAID Message, Sizes in Bytes

4.1.7 POTENTIAL PROBLEMS

A potential problem in the current virtual file system is the possibility of data corruption in the various files. The fix for a corrupted file is to delete it and regenerate the node. As the NetRAID software improves, the efficiency and speed of the virtual file system should improve.

4.2 Network Transmission

For communication over a network or the internet, NetRAID uses a standard message and directly calls .NET's TCP socket APIs for communication. Version one uses TCP socket 1978; however, users can set the socket number in the configuration file. Socket 1978 is registered to UniSQL; however, Google could not even find a home page for UniSQL so it should be satisfactory for most home users. Allowing each cluster to set its socket number may slightly improve security since an eavesdropper would have to know the exactly which socket to listen on.

4.2.1 Socket Listener

As soon as NetRAID is launched it creates a thread to listen for incoming messages. Visual Basic sockets and socket manipulation is very similar to standard UNIX C/C++ code. The code is similar to the examples provided by Stevens [39], but with some Visual Basic specific key words instead of C. When the messages are received they are decrypted and each pushed into a stack for the program core to analyze.

4.2.2 Socket Sender

When NetRAID needs to send information a sender object is declared that contains the ability to open the socket for writing. This is done most often by the core thread; however, the primary thread does send a few messages. The encryption function is called by the sender object directly.

4.2.3 NetRAID Message Structure

The structure of a NetRAID message (Figure 4) and code (Figure 5) is fairly simple. They contain Source, Type, Job Number, Message Number, and then Payload. The structure of the Payload is based on the message type.

```
Public Structure NetRAIDMessage
    Dim Source As Byte
    Dim MessageType As Byte
    Dim JobNumber As Int64
    Dim MessageNumber As Int64
    Dim Payload() As Byte
    Dim RawData() As Byte
End Structure
```

Figure 5. NetRAID Message Data Structure

NetRAID contains several functions and utilities that can either generate or parse the RawData to and from the other variables. The Payload size is set by a constant in the software, and in future versions it will be possible for it to be set from the configuration file. In version one the payload is two thousand bytes. The RawData array is set from a constant that is derived from the payload size plus the size of the header. This constant then also sets the network buffer size.

4.2.4 Job Number Generation

Each unique task that is executed by NetRAID is assigned a unique ID. By having a unique number each node can juggle several tasks. The number is generated by multiplying a random number by ten million, and then that is multiplied by the current millisecond. To associate the job number to a specific origin node, the number is then multiplied by ten and the node is added, so that the last digit is always the node. During testing it was found that job numbers conflicted roughly 1.5 times per ten thousand attempts. Since NetRAID is

likely to have only a few dozen transactions per day and each node's numbers will always be different than the numbers from the other nodes, the chances of a collision should be remote. Since the milliseconds are a seed, over the course of a day this element will keep changing. At the moment there is no security function assigned to job numbers; however, this could change in the future. The number generation algorithm could be improved so that a collision would be even more unlikely; for now however, this method seems to be adequate.

4.2.5 Message Number Generation

The message numbering is done sequentially to ensure that messages can be reassembled in order. Just as with the job number, this message numbering has no current security feature, but that could change in future versions of NetRAID.

4.2.6 Message Types

The message type tells the NetRAID core how to handle the data (Figure 6). As NetRAID grows, more types can be added. Since the information is being carried as a byte, up to 255 commands can be held before the message structure will have to be revisited. At the current time only 22 (9.7%) of the commands are used, so there is plenty of growth potential in the name space.

Number	Description
3	Testing Ping
10	Get node free space
11	Amount of free space
20	Delete a File
21	Deletion Confirmation
30	File Data
31	End of File and file information
32	File received OK
40	Request a file, file information
41	File Data
42	End of File
50	Begin Regeneration
51	File list token message
52	Files to Send list
53	Rebuild node file payload
54	End of file and file info
55	End of node regeneration
60	Begin User FAT rebuild
61	User FAT payload data
62	User FAT end of data
63	User FAT rebuilt correctly
70	Text message

Figure 6. Listing of NetRAID Message Types

4.2.7 Determining Destination Nodes

NetRAID, version one, uses random number generators to determine the three destination nodes. Ideally, the nodes would make a rough determination about what nodes have free space and better network connections. This is why there is support for determining free space; however, this algorithm has not been implemented.

4.3 NetRAID Core

Most of NetRAID's work is done by a thread launched immediately after the listener is launched. This program core checks to see if there are any incoming messages to analyze. If there are not, it sleeps for a fraction of a second before checking again. If information is present in the listening queue, the main function passes it off to a sub-routine that parses the message and calls whatever functions it needs on the basis of the message type. If the code calls for the data to be held until a different type of message comes in, the core has its own internal stack for temporary storage. For example, if the node starts getting in type 30 messages, it stores them in the internal stack and then checks the next message. It does this until it gets a type 31, then in sorts though the message stack and picks up on any data with the same job number. These messages are removed from the stack and the file is processed.

There is probably a better way to write the system than a large case statement that calls subroutines as needed; however, at the moment this method works accurately and seems to be stable. The more threads that are introduced, the more complex the memory management will be.

4.4 NetRAID User Interface

The NetRAID GUI is very simple. There is a main form (Figure 7), a local node control (Figure 8), and a network interface GUI (Figure 9). The main form launches the listener and core threads and is the launching point for the other two forms. The GUIs are primitive and spartan, but they allow a user to manipulate the NetRAID cluster's basic functionality.

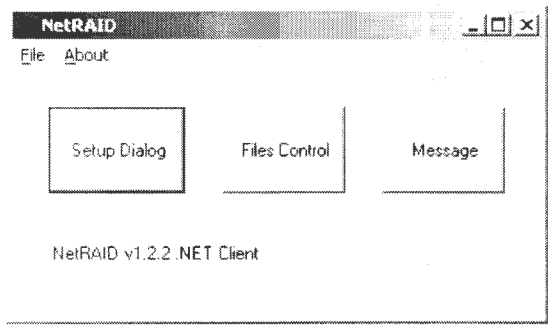

Figure 7. NetRAID Main Form

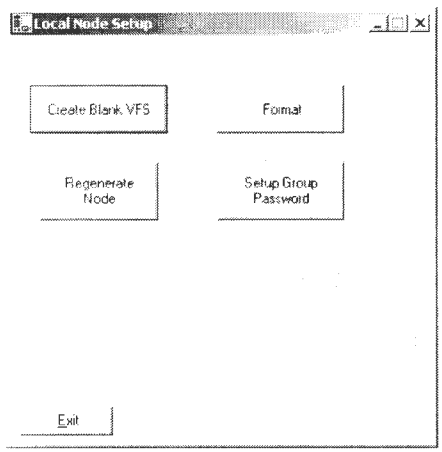

Figure 8. NetRAID Local Control Form

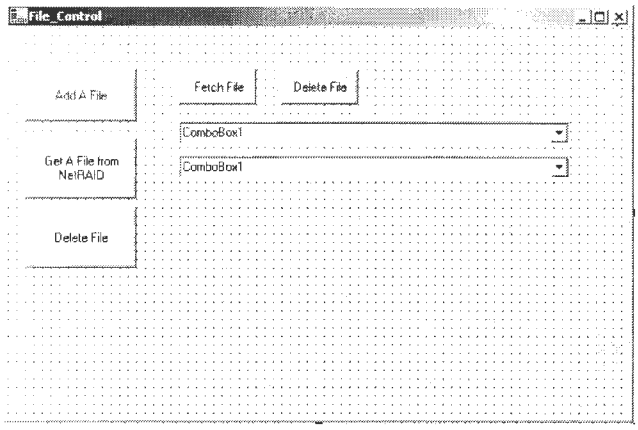

Figure 9. NetRAID Cluster Interface Form

4.5 Other Implementation Information

4.5.1 Threading

True threading only became supported in Visual Basic recently, so there are not many good samples and tutorials in wide circulation. Ardestani et al.'s book [2] is an excellent resource on Visual Basic, expanding on specific resource to expand on Tanenbaum's [43] more general Minux examples.

There are potential problems with synchronization and other multi-thread related programs. As NetRAID matures, these problems will be resolved more thoroughly than they have been in the first version. In Version One, the only cross thread variable is the stack that the listener places received data in to by analyzed by the program core. Another potential problem is a conflict that occurs when two threads try to access the same files in NRDFS. During testing this has not been a problem, but more work needs to be done to make sure Version One is truly thread safe.

4.5.2 The Configuration File

NetRAID uses an XML-like configuration file to set many of its variables. By providing an easy-to-edit text file to a user, NetRAID gains quite a bit of flexibility. Furthermore, debugging a simple text file is much easier than constantly having to change hard-coded variables during testing. For example, the line to set the local nrdfs directory is <DIR>C:\NETRAID</DIR>. Each line in the file is parsed by a module that runs at the program startup. The module knows many of the defaults and performs error checking on the data elements. The lines do not have to be in any specific order, and comments can begin with a "#" character.

4.5.3 Text Messages

The ability to send text messages from one node to another was developed initially as a debugging tool. Instead of sending files across an entire cluster for testing, the developer could send a simple text message from the source node to a designated destination node. This allowed for the testing of the encryption and network communications without having to wait for the complex code and protocols involving nrdfs and the constant formatting of all the nodes between each test. This functionality was exposed to the end user with version 1.1. This is not a replacement for a system such as AOL's Instant Messenger, but it does allow for secure direct text messaging from one node to another. For members of the same cluster, this allows for communication without worries of being snooped on or monitored.

The Text Message System uses two dialog boxes, one for an initial message (Figure 10) and one for replies. In the future this could be accomplished with one more complex form.

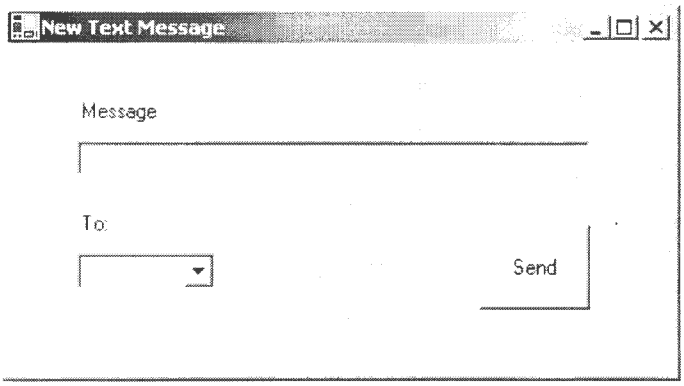

Figure 10. NetRAID New Text Message Dialog

5 NetRAID Security

NetRAID is designed to provide a satisfactory level of security that is consistent with convenience and reasonably quick performance. As the program is further developed, more security elements and techniques can be integrated into the framework. However, the cluster users should be able to decide what level of security they need. The more encryption used, the slower the system is. Each group needs to find a balance between speed and simplicity and security beyond a certain baseline. The best security is usually the most transparent (for both CPU and user interaction), and all efforts need to be made to build the system correctly in the first place. By planning every aspect of the client and the framework with security in mind, and then making everything as modular as possible, NetRAID should be safe and accurate.

5.1 File Security

All systems have security problems, and NetRAID is probably no different. However, the NetRAID framework is built upon .NET with its support for widely used and well understood protocols and systems. Many of the security components are semi-independent of NetRAID, and as problems arise they can be patched with minimal disruptions to the functioning of the cluster. TCP/IP, SSH, WebDAV [48], SSL all have had security flaws either inherent in them or in their implementation; however, they should provide NetRAID with adequate encryption and security for transmission across the public internet. As new flaws and fixes are discovered, a modular NetRAID and be fixed and updated just like larger projects such as the Firefox [14] browser or the family of Apache [13] web server projects.

5.1.1 User Responsibilities

Each user is responsible for his or her own file encryption. While NetRAID uses AES encryption for transmission, the more encryption the longer it takes an attacker to break the file.

5.1.2 Group Password

NetRAID has support for a group password. This needs to be the same for all nodes. This is created by the input a string of text. An SHA512 hash is generated from the key, and then a second hash is generated from the hash and stored. This second hash is used to verify that the original password is correct. By using the hash of the text a long string of pseudo-random information is used. A dictionary attack on the group password would require the attacker to first generate the SHA hash of the original string and then comparing the hash of that to the stored file. The extra computations involved will slow down the attack. The group key can be used as a source of the encryption key or for digital signatures.

5.1.3 Personal Password

Ideally the user when starting NetRAID should input a password. This functionality has not been implemented yet. This would facilitate better key generation and individual authentication.

5.2 Transmission Security

NetRAID is just like any other system that sends information over the internet. Is the data going to be secure, are the senders who they claim to be, and is the data integrity intact? Version One has support for AES [29] using the Rijndael [28] algorithm. The key is generated from the first part of the SHA512 Hash. The initialization vector is based on the

destination address of the node. If Node 1 is sending to Node 4, then the IV is a derivative of "4". The IV does not need to be transmitted since the source and destination both know it. An attacker would need to know the Node number of the destination to create the IV. Even is the key is compromised each computer still needs a unique IV. An attacker would have to use all of the possible IVs, this wouldn't take long, but would slow someone down.

The password may come from three sources. The worst method is a hard coded default key embedded in the code. This is very nice for debugging, but should be removed from future versions so end users aren't tempted to use it. The second is a text file in the \NetRAID folder that stores data. A hash is created based on the contents of this file and the key generated from the hash. This is better since a group can set the password to be what ever they'd like. However, if the text file can be read, the encryption is then useless. The best method is a password that is input and a hash generated from that. It is then checked against a hash of the hash derived from the key. The key is then not stored in either the code or on the disk, only the hash of the hash of the key. It would take quite a bit of CPU power to reverse the SHA512 hash on the disk. There are better ways to attack a PC than reversing SHA512 hashes.

5.2.1 Use of Asymmetric Keys

One potential source of encryption for NetRAID is the use of short term or one time asymmetric keys. The .NET cryptography has support for several types of encryption systems. RSA would be an easy and well researched system to use. Key pairs can be generated using random and pseudorandom information and distributed to the cluster. Each node can keep a key ring for the rest of the cluster. This would require more CPU power,

and a few more kilobytes of traffic, but these costs would be worth the extra security. A key ring would need to be implemented to support this encryption structure.

5.2.2 Use of Symmetric Keys

Alternatively or in conjunction with asymmetric keys, NetRAID could also use several different types of symmetric keys. .NET supports both Triple DES, and AES. NetRAID Version One already has AES encryption for transmission. With symmetric keys there is a problem of distribution and management. Keys could be exchanged via external communication or asymmetric distribution.

With the first method of exchange the initial passwords could be set via a phone call or encrypted email or other communication between the users setting it up. If the client software is encoded with good password generation rules, then weak passwords and key problems should be resolved before they can happen. Voice phone lines can be bugged, but there should be a reasonable expectation of the privacy of voice phone lines. For the first version of NetRAID, there is transmission security; however, the key is not random and constantly changing. If the users encrypt their files before adding them to the cluster than even if someone sat and intercepted all messages, this person would still need to work through the individual file encryption put in place by the users.

The other method by which symmetric keys might be exchanged requires the use of asymmetric keys for key distribution. This method would bypass potential person-to-person problems since the nodes would be communicating directly. However, this method would require even more CPU power for key generation. The NetRAID code would have to allow for a second key ring to support both sets of keys. This should not be too difficult and would

be worth while in terms of improved security. This is similar to a java based system described by Parnerkar, Guster, and Herath [30].

5.2.3 Use of One Time Key Systems

NetRAID should be a flexible system that can conform to the security needs of each specific cluster. The most security conscious user could implement a system of one time asymmetric and symmetric keys. With each new job the origin computer would generate a new key pair based on a hash created from several types of data including a random number and the time. This key would then allow for the distribution of a symmetric key based on similar, non-predictable information. After each job is completed, each node would toss out all key information. This would require a significant increase in CPU power, and a few kilobytes of extra data per job, but if the cluster only handles a few jobs each day, there is no reason that a user would find this security measure affecting the performance of any modern computer significantly. If the user group values the security of its data that much, then it should be able to protect this data with as many layers of encryption as it seems necessary.

5.2.4 Man in the Middle Attacks

A man in the middle problem can arise. Since messages are sent out on the public internet they could be grabbed and cracked. One method of preventing data corruption is the storage of the initial SHA512 hash on the origin server. Each client would check against it to help mitigate man-in-the-middle corruption. If there is someone with enough skill and CPU power to crack and spoof hashes attacking a cluster, then corrupted data is probably going to be the least of the user's data security problems. Digital signatures could be introduced into

the system, but that is yet another layer of complexity that would have to be added and maintained.

5.2.5 Authentication

NetRAID currently has no authentication. A first step in fixing this problem would be to have the listener verify the sending address. Addresses can be spoofed, but verification would make a potential malicious user have to go through a bit of extra work to set up the spoof. A better method would be to enforce some sort of signature or authentication message. NetRAID has support for the generation of a group password that could help authenticate machines. If someone tried to spoof being a node, they would also need to know the group password or their data would be gibberish and ignored by the system.

5.3 Hashes

NetRAID uses .NET's SHA512 hash generation API to create hashes of files for integrity verification. SHA512 was chosen because it was the strongest hashing algorithm that .NET 1.1 supported.

6 System Performance

6.1 Network Load

NetRAID's impact on a network and on a workstation should be minimal. The example network in figure one has five nodes. If each node backs up five files of five megabytes each per day, that is 25MB per node. Since each file goes to three machines that is 75 MB upload per day. With a 256kb/s upload rate, that is about 39 minutes of upload time per day. The extra overhead is only a few kilobytes per day. Each node would also be downloading (upload in MB / (# of machines – 1) * 3 * # of machines) or about 281 MB per day. With a 2mb/s DSL/Cable connection, the total download time is under 10 minutes depending on network speed. The addition of queuing and better network load balancing to NetRAID could dramatically decrease the bandwidth needed. Built in scheduling support could allow transfers to run at times when the internet has more available bandwidth. The implantation of a fast compression scheme like gzip [49] could also significantly improve bandwidth. However, if the files that are being sent have already had ZIP, RAR, or GZIP used on them, then the built in GZIP module would be redundant and only serve to waste CPU power.

6.1.1 Network Assumptions for Version One

The first version of NetRAID works on any network connection that can have a static IP. Firewalls might be a problem, but if the designated port is open, it should work. NATs should also work as long as the addresses and ports are forwarded correctly. Version 1.x development occurred on a small LAN running at 100Mb behind a NAT home router and firewall.

6.2 Modem Connections

Regular V.90 dial-up modems present an interesting challenge to the NetRAID infrastructure. They are slow and unreliable. Backing up a 25MB file on a DSL connection should only take a couple of minutes. However, on a 56Kbs modem it might take ten times longer. Given the importance of keeping accurate data, few people would find this slower speed worth it.

There are also a variety of techniques such as Microsoft BITS [1] to try to only use background bandwidth in a connection to make the data load more transparent. If several programs are using that same background bandwidth then the system becomes slow anyway. There should be a way to set up a 'proxy' node on one of the high-speed connections that would act as a proxy for the modem-based node and then synchronized with it whenever the modem connection is on. This method could be applied to satellite phone equipped laptops or other low bandwidth connections. The storage node could even be de-coupled from the server half of the application. Given the declining use of modem connections, this problem should be resolved with newer technologies. By keeping the NetRAID code open source, if a modem user wanted to extend the framework or client to better accommodate lower speed connections, then they could.

6.3 Encryption Load

The first version of NetRAID use AES for transmission security using [15] as the primary guide. Users should also encrypt the files that they post using any tool that they chose based on their own preferences. During testing, there was no noticeable difference in speed with encryption on or off.

The CPU load of encrypting and decrypting files could drag on node performance as much as the network's load. In a test using a 140bit AES key on an AMD XP 1.5GHz CPU, it took less than a minute to encrypt a 5MB file. As personal computers continue to increase in speed, problems of this type will become even less noticeable. For an average user NetRAID will only use a few CPU minutes per day spread across the day. The CPU and network cost is worth the return of having secure, distributed back-ups.

6.4 Host System Resources

6.4.1 Installer

The installer package is about 3.5 megabytes and is in the Microsoft System Installer format. It is built using Visual Studio 2003. If the user does not have the .NET runtime, he or she needs to download and install it from Microsoft.

6.4.2 Memory Footprint

When the program is at idle, NetRAID 1.0 uses roughly 18MB of memory. When NetRAID is processing a job this can dramatically increase based on the file size and the operations that need to be performed. This memory requirement is in addition to any resources the .NET runtime is using.

6.4.3 NRDFS Footprint

In the default 25,000 cluster mode, the disk footprint for Nrdfs is about 50 megabytes. In real world use this would need to expand. The number of clusters is a constant that can be manipulated by the configuration file.

6.5 NetRAID as Distributed File System

6.5.1 Naming and Transparency

A file name is really just a logical pointer to a block of data. According to Levy, this should remain transparent. There should be no need to have a different file name or to change file names based on the physical location of the file. The system user cannot tell the location of the file from the name or ID. In NetRAID the file ID is generated once, and the locations of the file remain completely transparent to the human user of the system. The user will click on a file name to back up, retrieve, or delete, and NetRAID uses the information it has stored in the system to do its task.

6.5.2 Caching

NetRAID does not need to implement caching for fast access. Since NetRAID is not designed for random access, but as a back-up system, caching is not necessary.

6.5.3 Remote Access

NetRAID provides for the transmission and execution of file system operations by exchanging messages using TCP/IP. It has a set list of commands for inter-node communication.

6.5.4 Fault Tolerance

A file system that allows for data to become corrupt is useless for the accurate storage of information. To maintain fault tolerance NetRAID implements hashing to maintain high level data integrity. TCP/IP also has its own set of data integrity tools that NetRAID takes advantages of. NetRAID meets both of Svobodova's [41] requirements that data be recoverable and robust. The file is recoverable since it can revert back to the saved state.

The user can grab the backed up file from the cluster. The file is robust since it can survive a storage medium crash. The goal of implementing the RAID-like system is to make sure if one virtual disk was destroyed, it could be regenerated from the other virtual disks.

7 NetRAID's Future

Future NetRAID versions should be able to implement sharing of files between clients and node, scalability beyond storage nodes for larger networks, and server based queuing for dealing with modems and other non reliable connections. NetRAID should also be able to support network load balancing and proxies.

7.1 Node Upgrades

The node could be made a lot smaller by implementing some sort of built in ZIP or dynamic allocation. This could lead to more fragmentation and could actually slow the system down.

A possible future extension to NetRAID is to allow sharing between users of files held in nrdfs. To implement this functionality, the server will need to be able to analyze what might be the best place store a file based on who in the cluster can see it.

7.2 Beyond the Peer-to-Peer Structure

NetRAID does not necessary need to be a peer-to-peer application for which everyone provides storage. The original concept of NetRAID decoupled the virtual file system from the members. If a large group of people wanted to set up a cluster and some of them had very little room on their machines, while others had large RAIDs or expensive and reliable storage or high bandwidth, it would make more sense for everyone to store data on the more powerful machines. But to do this it would be necessary to have a client that would allow the users to manipulate their own data on the storage nodes. This would be ideal for

low bandwidth and less reliable client connections. All participants must understand and agree to an arrangement by which they are sharing bandwidth resources unequally.

7.3 Improvements to the File Update System

Initially NetRAID can only perform file updates by using brute force to send a new copy of the file. This could be better accomplished with an rsync [36] type of transmission method by which only information that has changed is sent. Rsync [47] could cause less data to be transmitted, but would introduce another protocol and new potential security flaws into the NetRAID framework.

The first version of NetRAID has not included rsync due to the extra complexity. Burns and Long [9] published some research in delta compression in distributed systems that could also be applicable. They also examined only transmitting the changes in the file. This could be implemented by sending something similar to a patch file through the network instead of comparing data blocks.

For now, the user will need to manually delete the old file and add the new one. Since NetRAID uses file numbers not file names for identification, there is no problem with having more than one file with the same name in the system.

7.4 Improvements to the File retrieve System

NetRAID could be easily extended to use a system similar to Bittorrent [10] to allow several nodes to transmit information at the same time to speed up the transfer process. On most systems there is more download bandwidth than upload, so several computers can send to the same machine to optimize the data flow.

7.5 Other RAID Levels

At the current time NetRAID is capable of a data structure similar to RAID 3 or 5. NetRAID should also be able to emulate disk mirroring. This would allow for two people to set up a cluster and have duplicate information. A client node could even be configured to support having two mirrored virtual drives for even more reliability. Each drive could be on a different disk or computer.

8 CONCLUSIONS

The NetRAID framework has the potential to bring easy, inexpensive, and distributed back-up to the average computer user. Encryption helps make transmission and storage secure, and redundancy allows NetRAID to survive the lost of storage or server nodes without losing information. NetRAID has modular characteristics that allow for flexibility in encryption and transmission protocols, and to allow rapid updates as new flaws or enhancements to different dependencies are released. The framework is intended to be flexible to meet new user needs as they arise. As the capacities of the internet and CPUs increase, NetRAID's fingerprint will decrease and its usefulness will increase.

APPENDIX

A1 GLOSSARY OF TERMS:

Cluster: A set of NetRAID nodes acting as the virtual RAID

Node: A NetRAID client. It has both server and client functions. This is the equivalent to a single disk that is part of a RAID.

A2 EMAIL FROM HEWLETT PACKARD

Dear Chris,

Thank you for contacting HP. After some review, we found that we are no longer using this mark and that there is no problem with you using this term.

Regards,

IPL

REFERENCES

1. Background Intelligent Transfer Service (BITS) WWW.MICROSOFT.COM/BITS (Accessed: 15 March 2005)

2. Adestani, K., Ferracchiati, F.C., Gopikrishna, S., Redkar, T., Sivakumar, S. and Titus, T. *Visual Basic.NET Threading Handbook.* Wrox, Birmingham, 2002.

3. Andrew File System WWW.OPENAFS.ORG (Accessed: 10 March 2005)

4. Amoeba HTTP://WWW.CS.VU.NL/PUB/AMOEBA/ (Accessed: 20 April 2005)

5. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M. and Werthimer, D. SETI@home: an experiment in public-resource computing. *Commun. ACM, 45* (11). 56-61. 2002

6. Anderson, T.E., Dahlin, M.D., Neefe, J.M., Patterson, D.A., Roselli, D.S. and Wang, R.Y. Serverless network file systems. *ACM Trans. Comput. Syst., 14* (1). 41-79. 1996

7. Bolosky, W.J., Douceur, J.R., Ely, D. and Theimer, M., Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. in *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, (Santa Clara, California, United States, 2000), ACM Press, 34-43.

8. Bondi, R. *Cryptography for Visual Basic: a programmer's guide to the Microsoft CryptoAPI.* John Wiley, New York, 2000.

9. Burns, R.C. and Long, D.D.E., Efficient distributed backup with delta compression. in *Proceedings of the fifth workshop on I/O in parallel and distributed systems*, (San Jose, California, United States, 1997), ACM Press, 27-36.

10. Cohn, B. Incentives Build Robustness in BitTorrent, 2003.

11. Cooper, B.F. and Garcia-Molina, H. Peer-to-peer data trading to preserve information. *ACM Trans. Inf. Syst. Secur., 20* (2). 133--170. 2002

12. Cougias, D.J., Heiberger, E.L. and Koop, K. *The backup book: disaster recovery from desktop to data center*. Schaser-Vartan Books, Lecanto, FL, 2003.

13. Apache.org WWW.APACHE.ORG (Accessed: 12 April 2005)

14. Mozilla.org WWW.MOZILLA.ORG (Accessed: 11 April 2005)

15. Freeman, A. and Jones, A. *Programming.NET Security*. O'Reilly, Sebastopl, CA, 2003.

16. Grosshans, D. *File systems: design and implementation*. Prentice-Hall, Englewood Cliffs, N.J., 1986.

17. Halvorson, M. *Microsoft Visual Basic.Net step by step*. Microsoft Press, Redmond, Wash., 2002.

18. Held, G. Personal computer file compression: a guide to shareware, DOS, and commercial compression programs, Van Nostrand Reinhold, New York, N.Y., 1994.

19. Hughes, G.F. and Murray, J.F. Reliability and security of RAID storage systems and D2D archives using SATA disk drives. *Trans. Storage, 1* (1). 95-107. 2005

20. Intermezzo WWW.INTER-MEZZO.ORG (Accessed: 11 March 2005)

21. Kroenke, D. *Database processing: fundamentals, design, & implementation.* Prentice Hall, Upper Saddle River, NJ, 2000.

22. Kurniawan, B. and Neward, T. *VB.NET core classes in a nutshell: a desktop quick reference*. O'Reilly, Sebastopol, CA, 2002.

23. LaMacchia, B.A.*NET Framework Security*. Addison-Wesley, Boston; London, 2002.

24. Lee, E.K. and Thekkath, C.A., Petal: distributed virtual disks. in *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, (Cambridge, Massachusetts, United States, 1996), ACM Press, 84-92.

25. Levy, E. and Silberschatz, A. Distributed file systems: concepts and examples. *ACM Comput. Surv., 22* (4). 321--374. 1990

26. The Mono Project WWW.MONO-PROJECT.COM (Accessed: 30 March 2005)

27. Nantz, B. *Open source.NET development*. Addison-Wesley, Boston, 2005.

28. Nichols, R.K. and Lekkas, P.C. *Wireless security: models, threats, and solutions*. McGraw-Hill, New York; London, 2002.

29. AES Algorithm (Rijndael) Information HTTP://CSRC.NIST.GOV/CRYPTOTOOLKIT/AES/RIJNDAEL/ (Accessed: April 20, 2005)

30. Parnerkar, A., Guster, D. and Herath, J. Secret key distribution protocol using public key cryptography. *J. Comput. Small Coll., 19* (1). 182--193. 2003

31. Patterson, D.A., Gibson, G. and Katz, R.H., A case for redundant arrays of inexpensive disks (RAID). in *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, (Chicago, Illinois, United States, 1988), 109--116.

32. Pavlicek, R.C. *Embracing insanity: open source software development*. Sams, Indianapolis, Ind., 2000.

33. Pfleeger, C.P. *Security in computing*. Prentice Hall PTR, Upper Saddle River, NJ, 1997.

34. Ritchie, D.M. and Thompson, K. The UNIX time-sharing system. *Commun. ACM, 17* (7). 365-375. 1974

35. Salomon, D. *Data compression: the complete reference*. Springer, New York, 1998.

36. Samba.org WWW.SAMBA.ORG (Accessed: 13 March 2005)

37. Sinha, P.K. *Distributed operating systems: concepts and design*. IEEE Press, New York, 1997.

38. Sterling, T.L. *Beowulf cluster computing with Windows*. MIT Press, Cambridge, Mass., 2002.

39. Stevens, W.R. *UNIX network programming*. Prentice Hall PTR, Upper Saddle River, NJ, 1998.

40. Stringfellow, S., Klivansky, M. and Barto, M. *Backup and restore practices for Sun Enterprise servers*. Sun Microsystems Press, Palo Alto, CA, 2000.

41. Svobodova, L. File servers for network-based distributed systems. *ACM Comput. Surv.*, *16* (4). 353--398. 1984

42. Tanenbaum, A.S. *Modern Operating Systems -- 2nd Edition*. Prentice Hall, Upper Saddle River, N.J., 2001.

43. Tanenbaum, A.S. *Operating Systems: Design and Implementation*. Prentice Hall, Englewood Cliffs, NJ, 1987.

44. Thekkath, C.A., Mann, T. and Lee, E.K., Frangipani: a scalable distributed file system. in *Proceedings of the sixteenth ACM symposium on Operating systems principles*, (Saint Malo, France, 1997), ACM Press, 224-237.

45. Toigo, J.W. *Disaster recovery planning: preparing for the unthinkable*. Prentice Hall, Upper Saddle River, NJ, 2003.

46. The Gimp Toolkit WWW.GTK.ORG (Accessed: 15 March 2005)

47. Tridgell, A. Efficient Algorithms for Sorting and Synchronization, The Australian national University, 1999.

48. Web-based Distributed Authoring and Versioning (WebDAV) WWW.WEBDAV.ORG (Accessed: 17 March 2005)

49. GNU Zip WWW.GZIP.ORG (Accessed: 15 March 2005)