

# Combining Traditional and Open Source Testing Processes for NFS Version 4

**Bryce W. Harrington**

[bryce@osdl.org](mailto:bryce@osdl.org)

Test and Performance Department  
Open Source Development Labs

## Abstract

As use of open source technology expands into the enterprise, a challenge emerges for the end user: the testing of these applications is often not as rigorous and comprehensive as is typically the case for proprietary applications. Yet despite this, many open source applications have gained a reputation of quality and robustness, using an evolutionary "release early, release often" approach.

The open source process has important implications for achieving quality, yet it is very different from traditional formal testing. This raises an interesting question: could traditional testing and open source testing be conducted in a way that taps the strengths of both and avoids the weaknesses of either, and thus better address the challenge faced by end users?

This paper presents a case study of fusing traditional testing in the Linux NFS version 4 open source community. The development and testing efforts of a number of different companies and universities are coordinated using a typical open source processes, enabling the sharing of results and the distribution of test development, testing, and analysis work. A test matrix is used to break out the tasks, identify owners, and track progress. Existing open source tests such as Connectathon, PyNFS, and Iozone are identified, enhanced for our purposes, and standardized across the community.

NFS is an interesting case study, as it is a critical technology for many businesses and has very high expectations from end users in terms of functionality, robustness, interoperability, performance, and security. Yet the Linux kernel and its NFS stack is not owned or controlled by any single company; thus no single entity has responsibility and accountability for conducting formalized testing. While for some applications, an evolutionary approach to quality may work well; for NFS the level of tolerance for issues will be much narrower.

This provides an excellent opportunity for testing to play a central role in enabling the adoption of the technology by a) identifying issues early in development so they are fixed quickly, b) providing data to potential adopters about the risk/reward they can expect to gain, and c) producing information and tools to help end users identify, report, and overcome problems they run into.

Based on our experiences, several technical, sociological, and organizational ideas for improving the overall state of testing in open source are outlined. As open source becomes increasingly more critical in the enterprise, small steps to improve testing processes today could have large benefits for the future.

## About the Author

Bryce is a Senior Performance Engineer at the Open Source Development Labs (OSDL) in Beaverton, Oregon. He graduated with a BSc in Aerospace Engineering from the University of Southern California and an MSc in Aeronautical Engineering from the California Institute of Technology in 1995. Currently Bryce is leading OSDL's NFSv4 testing project.

## INTRODUCTION

There is no single best way to test a piece of software, and testers have learned, combining multiple methodologies can result in much better coverage than using a single methodology alone. In this paper we treat the open source development process as a new testing methodology and explore how it can be combined effectively with more traditional methods like regression and performance testing.

The idea that the Free/Open Source Software (FOSS) community could produce higher quality products through use of formalized testing should be no surprise; the challenge is in gaining the community's adoption and appreciation of it.

The Open Source Development Labs (OSDL) became interested in NFSv4 due to interest in it by enterprises using Linux in their data centers. We found that a number of organizations were already involved in testing NFSv4, and that the desired role for OSDL was to facilitate, organize, and improve the quality assurance processes. OSDL exists to find ways to help enterprises and the open source community work together to address common needs. Previous efforts have included stabilization of the 2.5 Linux kernel, publication of capabilities needed by the carrier-grade industry, and facilitation of technical discussions between vendors, customers, and the community.

# BACKGROUND

## Network File System (NFS)

NFS [1] is a UNIX protocol for large scale client/server file sharing. It was originally developed by Sun Microsystems, and has become a de facto standard on all UNIX platforms via RFCs 1094, 1813, and 3530. It is analogous to the Server Message Block (SMB) and Common Internet File System (CIFS) protocols on Microsoft Windows, however it maps better to the UNIX Virtual File System (VFS) semantics, has an easy to understand protocol, and is relatively simple to implement on UNIX-based systems.

Version 1 was first introduced in 1984 but is not typically used on Linux. Versions 2 and 3 of the protocol were 'stateless', meaning the server did not keep track of which clients were using which files. Locking and caching were implemented externally to the protocol, and no specific provisions were included for security. Version 3 marked a shift to using TCP as a transport; the previous versions had used UDP exclusively, which had proved difficult to use on a wide area network (WAN).

The need for state tracking and security caused many users to shift to the Andrew File System (AFS), developed at Carnegie Mellon University (and named after Andrew Carnegie and Andrew Mellon), as a more capable alternative. AFS provided better security, performance and scalability than NFS, and has thus been adopted by a number of organizations for large scale file sharing.[2]

Influenced strongly by AFS, a new version of the NFS protocol was introduced with the publication of RFC 3010 in 2000, and RFC 3530 in 2003.

This development work has been undertaken by a number of companies, with most of the developers working as part of the Center for Information Technology Integration (CITI) at the University of Michigan [3]. However, development alone will not be sufficient to drive adoption of NFSv4. Many of the key usage models for NFSv4 involve large numbers of machines; thus even a short duration downtime can lead to extensive costs in lost productivity. Because of this, testing will be critical, both from the standpoint of eliminating defects and to give potential adopters a confidence about its capabilities.

## Need for Testing

OSDL became aware of the need for testing of NFSv4 through formal and informal discussions of testing priorities with its member companies in the latter half of 2004. It was remarkable that while NFSv4 testing was not the number one priority for very many companies, it appeared to be somewhere on everyone's top ten list. OSDL was looking for testing projects that would benefit as many companies as possible, that were in need of formal testing similar to what OSDL had done previously, and that were not already getting sufficient attention from testers. NFSv4 fit the bill.

Specifically, OSDL's goals were:

1. Ensure that the testing of NFSv4 is effective at improving the quality of NFSv4
2. Open visibility of the testing so everyone can see the current status
3. Share the effort of conducting the testing to many companies
4. Help new testers join the testing effort
5. Participate in the testing efforts by gathering, writing, and running of tests

## Testing Objectives for NFSv4

Testing is an extremely broad topic, covering standards conformance, validation of new features, performance measurements, code audits, and so on. The sheer number of ways for testing the software is overwhelming. Thus for organizational sanity OSDL, several member companies, and the NFSv4 community started by identifying the high level goals for the testing efforts to achieve.

The first goal is to assure the Linux NFS version 4 implementation be as complete and correct as any proprietary NFSv4 implementation. This helps the developers gage their progress, identify regressions, and notice where their implementation work needs further attention.

The second goal is to showcase the new NFSv4 features and benefits. We expect to be able to show significant improvements in performance and security compared to NFSv3, and want to make measurements of these changes to help prove the benefits of NFSv4 adoption. Doing this will also help identify best practices and provide users with example implementations, and help developers in maximizing the magnitude of these benefits.

The third goal is to give assurance to end-users that NFSv4 will not impose new limitations compared to what they are currently using. We want new adopters' initial experiences with NFSv4 to be satisfactory, and to meet their expectations of its quality.

## **Business Interest in NFSv4 Testing for Linux**

The next step was to identify the reasons that other companies were interested in NFSv4 testing. The purpose of this analysis was to validate the goals and to help find rationales for convincing additional companies to participate. [4]

- *Technology Consolidation* – Shift customers from AFS and other file systems to NFS to provide cost savings from simplification of migration paths and reduction in supported technologies.
- *Promote Technology Adoption* – By ensuring Linux has good NFS support, it enhances or enables other products and services the company provides.
- *Strategic Investment* – The company relies directly on Linux and wishes to ensure its NFSv4 support is on par with commercial implementations for its long term needs.
- *Internal Adoption* – The company intends to implement NFSv4 in their environment to replace NFSv3 and wish to verify it meets their immediate needs.

The companies contributed in different ways: Direct funding, equipment, tests, testers, use cases, defect reports, and documentation.

## **THE OPEN SOURCE TESTING METHODOLOGY**

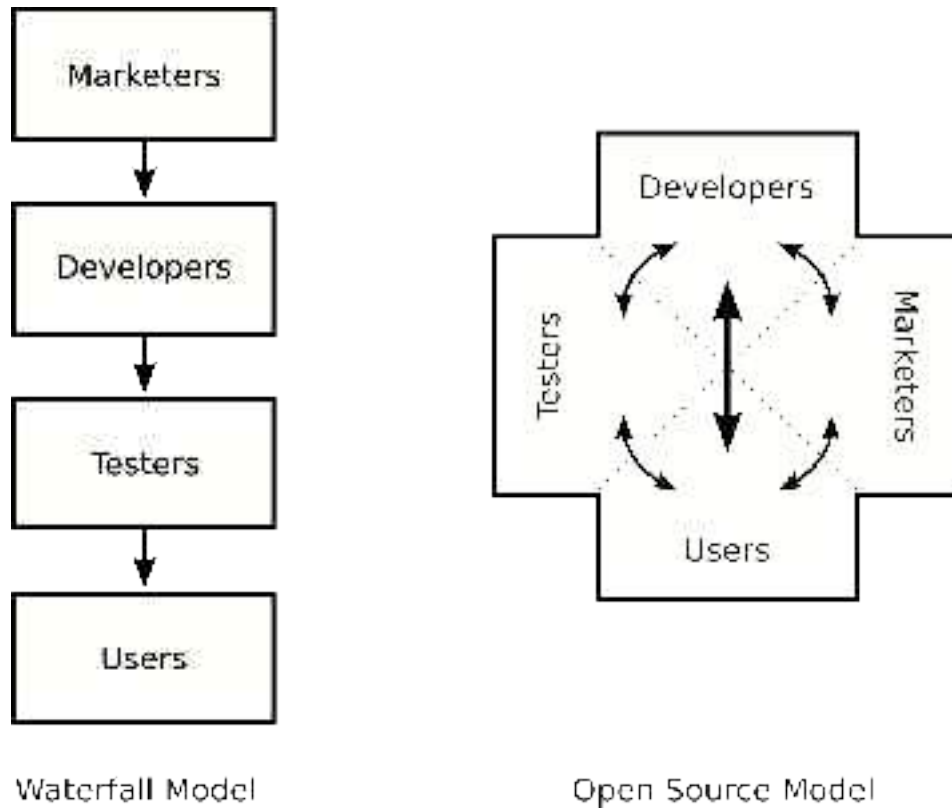
The notion of open source as a testing methodology was perhaps most famously expressed in Eric Raymond's essay "The Cathedral and the Bazaar" with the famed adage attributed to Linus Torvalds, "Given enough eyeballs, all bugs are shallow". [5] While not every project follows the approach Raymond outlines in his paper, it has served as a useful model for characterizing how open source projects can achieve high levels of quality without relying on traditional testing methods.

### **The Process in a Nutshell**

A more technical description of the methodology might characterize it as an evolutionary development approach that heavily leverages its user-base to form an integrated feedback cycle. It blurs the distinctions between users, testers, and developers, and strives to involve everyone as co-developers. It is a highly organic process, informal and rarely organized, yet tends to be able to converge on a stable product with noteworthy speed. It is good at identifying robustness, interoperability, security and portability problems, and weaker at performance and conformance testing.

The open source model is often contrasted with the more linear and compartmentalized "waterfall model," such as in Figure 1. The waterfall model is not a completely accurate model of the way software is developed today due to the proliferation of alternative development methodologies, yet due to its production-line mentality it fits with expectations of how the process "should" work, and thus from a conceptual standpoint it serves useful for distinguishing how open source is different. [6]

The testing process for open source projects typically begins with a public release of the code, often coupled with an announcement of some form to a mailing list, blog, and/or software registry. The initial release of the code is often very incomplete, but it serves a particular need. Users with an interest in that software download it and try it out, and report back on errors they encounter or ideas for features they'd like to see added. Developers work on whichever of these reports seem most interesting or necessary to them. [7]



**Figure 1:** Comparison of traditional “waterfall” model with open source

Users also get involved by writing patches for the code, fixing defects or adding features that they care about. These changes are submitted to the core developers for incorporation. The patches go through some sort of review process, which varies from project to project (some apply all patches without question, others have stringent quality requirements).

Marketing plays a very different role in open source than in traditional development processes. Traditionally, marketing plays both an internal role for defining requirements and creating specifications, and an external role for promotion of the product to support sales. In a typical open source project, marketing tends to play a reduced role, confined to external promotion of the project's work. [8]

### **Evidence of the Successes of Open Source Processes**

That this simple process can result in high quality code seems counter-intuitive. Organization in these projects often seems haphazard. Many projects do not have delivery roadmaps, task assignments, or test plans. Bugs seem to get fixed unpredictably, seemingly important features languish forever unimplemented, and designs are often left undocumented.

Despite this, the evidence is strong that open source processes can produce some of the highest quality applications in the industry. David Wheeler's paper "Why Open Source Software / Free Software" [9] summarizes published information about the reliability, performance, scalability, and security of open source applications as compared with other alternatives.

Studies of the robustness of several major open source projects have demonstrated significant advantages of the open source program over closed source alternatives.

Studies have shown the GNU/Linux OS has fewer flaws, fewer critical system failures, and fewer crashes ([10], [11], [12], [13]). Covertly [14] conducted a four-year research effort comparing the Linux kernel against the industry

average. They reported 985 defects in the 5.7 million lines of code, and compared this with data from Carnegie Mellon University that would predict 5,000 defects in a similar sized proprietary program.

A Swiss evaluation of website uptime by Syscontrol AG [15] showed that the average down-time of the Apache web server as 3.23 hours per month, compared with 11.03 for Microsoft IIS, 3.35 for Netscape, and 9.21 for other servers. A Netcraft uptime study [16] in 2001 reported that of the 50 sites with the highest uptime ratings, 92% were running Apache, and half were run on open source operating systems.

A code analysis [17] published in the "Communications of the ACM" reviewed 6 million lines of code for a number of programs over time. They reached the conclusion that open source "code quality appears to be at least equal and sometimes better than the quality of [closed source software] code implementing the same functionality."

Beyond robustness, open source applications such as Linux, Samba, MySQL have also repeatedly been found to be equal or higher performance than their proprietary counterparts, according to studies by PC Magazine [18] [19], eWeek [20] and more.

Open source also scores well on security. Informal studies show that Linux vendors have faster response rates to security issues than others [21], that GNU/Linux systems are relatively immune from outsider attacks [22], that Apache has much fewer security vulnerabilities than Microsoft's IIS [23], and that viruses are much more common on Windows than Linux [24]. It is sometimes claimed that the higher virus numbers for commercial products is due to their larger installation base, however according to Netcraft's August market share survey across all domains, Apache commands a 60.5% marketshare, compared with 20.4% for Microsoft IIS [25].

Clearly, there is mounting evidence that open source development methods are able to produce code that scores well on key quality metrics. For this reason, these methods are worth exploring so that the characteristics or mechanisms that make them work so well may be more generally applicable.

## **Open Source Processes Alone Do Not Guarantee Quality**

Despite the growing evidence showing that programs developed using open source methodologies can be very high quality, there are of course, no guarantees.

Indeed, it is relatively easy to find unusable, buggy, and ill-performing open source programs. In some cases this can be explained as due to the programs being early in their life-cycle. In other cases, lack of proper adherence to open source processes may be a cause. In still other cases the cause can be attributed to lack of sufficiently large user-bases. Yet even for programs that have active development communities, issues that would not be found in commercial software can be easily found.

There are several reasons why the open source methodology does not ensure better code more often. To name a few:

1. *Non-user facing defects.* Defects that are not known will not get attention. Since developers often rely on users to report defects as they find them, defects that don't affect users directly may remain hidden until new features reveals them. Unfortunately, time passes between the cause of the defect and its discovery, and the knowledgeable individuals are no longer available. If these latent defects are revealed earlier in development, they would stand a better chance of getting fixed swiftly and provide savings later on.
2. *Tolerable defects.* Testers and users take different mind-sets to a program. Testers are actively looking for issues and motivated to report even minor quirks. Users, on the other hand, desire to see the program work correctly, thus may overlook minor discrepancies that none-the-less are legitimate issues. The accumulation of these minor issues can give a program an appearance of being poorly tested, when in reality its core capabilities may be best of breed.
3. *Local optima.* In open source, development is often organized around the individual program level. Great attention will be placed at making a particular library, application, or server high quality, easy to use, and powerful, yet each program may be implemented in incompatible ways. Different interface standards may be employed. They may use different configuration approaches. They may have notably different look and feel designs. Viewed at a higher level, while each program may be great in and of itself, the inconsistencies from one program to the next can result in serious issues when using the system as a whole. Bugs seem to prosper in the cracks between integrated components.

4. *Slow convergence.* Programs like Apache, Linux, Samba, Sendmail, Bind, and so forth have been the result of years of attention. In some cases, proprietary alternatives will more swiftly meet customer needs in the short term, and users may expect open source projects to add and perfect features more quickly than is seen in practice. While the particular open source application may be of high quality eventually, early in its life-cycle it may not be able to fully meet all of the user's needs.
5. *Defect propagation.* A unique issue with having source code easily available for examination and/or re-use is that a defect in a piece of code that is highly copied or reused will get propagated into a number of other open source programs.

While they can present serious issues in the short term, in the long term projects find ways to work around or avoid the issues, especially when ample resources are made available and open source methods are mixed with more traditional methods.

## Fostering the Open Source Methodology

The practices in successful open source projects often seem very natural and low-key, and can lead to the misconception that the working style arises spontaneously and effortlessly. Simply making code available under open source terms is not sufficient to guarantee the project's success within the open source model. In fact, achieving viability with open source requires equal parts technical quality, sociology, timing, and luck.

The magnitude of the challenge is evident from a cursory review of the SourceForge open source project registry. As of June 2005 there are over 100,000 registered projects, operated by over a million open source developers. Only a portion of the total number of projects register through SourceForge, so the actual number of open source projects is well above 100,000. Yet the number of packages available for a typical Linux distribution is much lower. Red Hat Enterprise Linux, a corporate-oriented distribution, includes approximately 1500 packages [26]. Debian Linux, a community-oriented distribution, includes about 15,500 packages [27]. Freshmeat, a registry of released open source software, lists about 38,000 projects that have produced at least one release [28]. It is not uncommon for one project to produce multiple packages, so the package number gives a very conservative estimate for the number of viable open source projects. These numbers would suggest a rough rule of thumb that of 100 open source projects started, no more than about 10 will succeed at producing a package that is included in a Linux distro, and only around 1 will be of sufficient interest for inclusion in an enterprise distribution. Clearly, successful open source projects are the exception, not the rule.

Even among those projects that have been accepted into a Linux distro, not all would be considered as high of quality as projects like Apache or the Linux kernel. The interesting question to ask is what processes, practices, and characteristics are enabling the low-quality projects to grow into high-quality ones. In particular, what testing activities are they employing to help foster their successes?

A position paper from Carnegie Mellon University [29] outlined a study examining the publicly visible portions of successful open source projects from November 2001 to March 2002. It noted several several major characteristics that the projects shared:

1. The life-cycle of the projects studied were all in the software maintenance phase. The projects had benefited from having a good initial design and an architectural foresight to enable modular, incremental growth to be achieved without heavy risk of introducing major quality setbacks.
2. All projects used nightly builds. This ensures that the changes made to the codebase do not break the build system, as well as providing a downloadable binary for users to test. There are also tools such as Tinderbox that assist in identifying the precise change(s) that caused the breakage.
3. Publicly visible bug tracking is performed by nearly all of the projects examined. A bug tracking tool enables users to post issues and enhancement requests, and opens a dialog with the developers and other users with similar interests. The tracker also provides a vector for users to become directly involved in development, by performing traces and tests on their own system, or even creating and sharing fixes on their own. The trackers also fulfilled a project management role by allowing prioritizing defects or flagging issues for closure prior to a particular planned release.

As will be explored below, for NFSv4 the first characteristic has been followed for quite some time, the second has been recently adopted, and the third is in the process of being adopted.

The Carnegie Mellon researchers also modeled open source projects, examining the input/output flow in a "Free-Body Diagram" style. The leaders of the projects control the engineering practices inside the boundary of their project, and establish practices that simultaneously maximize the outgoing flow of information to the community while strictly limiting and controlling the flow of incoming information. This allows the project to select critical aspects of software best practice felt to be of relevance to the project, without overly burdening the external developer. They add, "Any quality-related intervention must respect this overall approach." [29]

Another aspect they noted of quality management in open source projects was the aggressive use of software tools to manage it. The leaders strive to shift policy enforcement from people to tools. This not only frees up the leaders to focus on more important development activities, but also ensures the rules are applied consistently and that they will remain in place even if the project participants change or if activity levels wax and wane. It also provides an effective way to mediate between participants who may be widely separated both in geography and time.

Based on their analysis, the authors recommend that the following criteria are required of any quality-related technology or process improvements introduced to a project:

1. Incremental model for quality investment and payoff
2. Incremental adoption of tools and methods
3. Allows for untrusted input from anywhere, but produces a trusted version of the code for universal use
4. Tool interaction style is easily adopted by practicing open source developers.

An effective way to institute better Quality Assurance practices in an open source project is to go slow, changing the processes incrementally little by little and avoid processes or tools that lay too far outside what the community is already using.

## IMPROVING TESTING METHODOLOGIES FOR NFSV4

With NFSv4, the desire is to leverage the full strengths of the open source methodology, and to couple it with some of the more formalized methods of traditional testing. This objective was attacked from a number of angles.

### Test Matrix

When OSDL first took interest in assisting with NFSv4, our first question was what form our assistance should take. To answer this, we first analyzed the existing testing efforts and the quality questions being posed by the users and developers.

In formal testing, it is traditional to lay out a test plan detailing all of the items to test. However, as described in the previous section, open source projects rarely follow such formal processes. Thus we wondered if it was possible to find a comfortable middle ground, that would give companies the confidence that the testing was well planned, yet not impose procedural hurdles that would fail to work in practice.

We also reviewed existing presentations, documents, and test plans that addressed testing needs. Within a month, we found ourselves with a comprehensive yet overwhelming list of items to test.

The community needed a way to collect and organize these disparate ideas and plans to communicate testing needs coherently. Early on, Mary Edie Meredith of OSDL, Data Center Linux (DCL) Initiative Manager, suggested the notion of a test matrix, to correlate test items with test programs and reference testing resources and staff. The form this document took was a listing of testing items in a spreadsheet that resembled a work breakdown structure (WBS). Like the typical WBS, the NFSv4 test matrix organized testing tasks into a numbered hierarchy, as seen in Figure 2.

**Figure 2:** NFSv4 test matrix

PHASE	TEST ID	DESCRIPTION	STATUS	COMPLETION
PERFORMANCE TESTING	1	Compare NFSv4 vs. NFSv3 for constant size tests	Success	In Progress
	2	Test to perform requests of various available operations	Success	In Progress
	3	Test to perform requests of cacheable read/write operations	Success	In Progress
	4	Random read/write requests from many clients to one server	Success	In Progress
	5	Randomly generated loads	Success	Not Started
	6	Test to test file write beginning to end and then random I/O	Success	New
	7	Test to measure I/O in a client to specifically test I/O	Success	New
	8	Workloads - random read/write operations	Success	New
	9	Workloads - random read/write operations	Success	New
	10	Workloads - random read/write operations	Success	New
	11	Workloads - random read/write operations	Success	New
	12	Workloads - random read/write operations	Success	New
	13	Workloads - random read/write operations	Success	New
	14	Workloads - random read/write operations	Success	New
Compare NFSv4 vs. NFSv3 on TCP vs. RAW	15	Compare NFSv4 vs. NFSv3 on TCP vs. RAW	Success	New
	16	Compare NFSv4 vs. NFSv3 on TCP vs. RAW	Success	New
	17	Compare NFSv4 vs. NFSv3 on TCP vs. RAW	Success	New
	18	Compare NFSv4 vs. NFSv3 on TCP vs. RAW	Success	New
	19	Compare NFSv4 vs. NFSv3 on TCP vs. RAW	Success	New
	20	Compare NFSv4 vs. NFSv3 on TCP vs. RAW	Success	New
	21	Compare NFSv4 vs. NFSv3 on TCP vs. RAW	Success	New
Test performance on different local filesystems	22	Test performance on different local filesystems	Success	In Progress
	23	Test performance on different local filesystems	Success	New
	24	Test performance on different local filesystems	Success	New
	25	Test performance on different local filesystems	Success	New
	26	Test performance on different local filesystems	Success	New
	27	Test performance on different local filesystems	Success	New
	28	Test performance on different local filesystems	Success	New
Test performance on different cluster filesystems	29	Test performance on different cluster filesystems	Success	New
	30	Test performance on different cluster filesystems	Success	New
	31	Test performance on different cluster filesystems	Success	New
	32	Test performance on different cluster filesystems	Success	New

The testing task ideas were divided into five areas, with objectives defined as follows:

**Functional Testing** - Ability to do what it's supposed to do. Standards compliance, regression, compatibility, static code analysis, etc.

**Interoperability Testing** - Ability to work with other versions of NFS, other operating systems and other software/filesystems/etc. Generally associated with NFS.

**Robustness Testing** - Remains stable and recovers even in extreme situations. Stability, error recovery, race conditions, etc.

**Performance Testing** - Able to perform well under real and theoretical workloads. Load, stress, destruction, scalability, etc.

**Security Testing** - Resistant to being compromised and difficult to attack.

Within each category we identified particular types or aspects of testing. For instance, in robustness testing we identified the following aspects: basic stability assessments, resource limit testing, stress load testing, scalability (robustness), recovery from problems while under light/normal/heavy loads, race conditions, and automounter robustness.

In each of these sections, the testers and developers then itemized the specific tasks that were felt to adequately cover the need. For example, with resource limit testing we identified the various resources that can be limited, such as memory, disk space, pid, inode, and swap space, and created tasks to perform a test of each situation.

Prioritization was then performed to highlight the specific testing tasks that were felt to be most important, versus ones that were less necessary or that could be postponed for a time. To establish the priorities, considerations were made for whether the given test item helped us towards achieving one of the community's objectives, as listed above.

It was important for the viability of the effort that the priorities represent a consensus between users, developers, and testers. Initially, we attempted to gain this through email discussions, but found that interest was low. We shifted to hosting weekly conferencecalls and working through the matrix. Over the course of three months we worked our way through the test matrix, prioritizing and fleshing each section out. This work was completed in May 2005 and is available online at <http://developer.osdl.org/dev/nfsv4/testmatrix/>.

In a number of cases the team found that community members were already working on testing tasks. Tracking this existing activity in the test matrix helped other testers avoid duplicating efforts. This correlation also helped to identify gaps where specific forms of testing were needed, but where the existing tests lacked the necessary coverage. This information has proven especially interesting for the test authors, giving clear direction about what to add to tests, and why.

With the help of this prioritized matrix, we have been able to group the items into sequential milestones towards achieving enterprise-readiness for NFSv4. These activities include:

- Detailing steps needed for performing the testing tasks
- Writing tests to support the testing tasks
- Setting up test cases using the tests
- Performing testing tasks
- Analyzing and reporting on results

Using a spreadsheet format was convenient for printing and for making changes to many items at once, but it was found to not be an ideal match to the community's needs. The main issue was that it was cumbersome for more than one person to update at the same time; this was particularly problematic as testers needed to update their status. It was also difficult to hyperlink items to their tests, results, or commentary.

For these reasons, the test matrix was converted from a spreadsheet to a wiki. The community members were already comfortable using wikis from other projects for collaboration. This has improved the ability to share access to the data, hyperlink to additional materials, and give the NFSv4 community ownership over it. There are some minor remaining issues but it is expected that these can be resolved in time.

## **Synthetic Testing**

Groupe Bull, an IT solutions company headquartered in France and emphasizing a focus on open environments, has been contracted to perform dedicated testing of NFSv4 on Linux. Their efforts have included focused testing on a particular characteristic of the system, and has worked with the development community to fix a number of issues that their testing uncovered.

For example, they set up a test to place as many (zero-byte) files into one directory as possible [30]. They tracked the time needed to create 100 more files in the directory. As they conducted these tests, they would find and report issues that caused NFSv4 to take excessive amounts of time to create the files. The developers took interest in these issues and quickly resolved them, enabling the testing to continue to new levels. At last count, they had achieved 1.6 million files in one directory. This level is well beyond what a typical user would expect to work, and thus would be unlikely to be reported as a problem until perhaps late in development. By conducting this artificial, synthetic testing early in the development process it identifies and eliminates defects well before any users would encounter them, thereby ensuring a better experience by future users.

The test matrix has been of particular use for Bull. It helps them to scope out and prioritize their own work and it ensures their efforts address the primary needs of the users and developers.

Bull follows a traditional testing methodology, writing test plans, conducting the tests, and creating reports from their results. However, these efforts have been adapted to fit within the open source model used by the NFSv4 project. They report their findings on the mailing lists, and work directly with developers to close the issues.

These efforts are also proving to be the most effective way at accomplishing the tasks set forth in the NFSv4 test matrix. It is hoped that additional companies can become involved in ways similar to Bull, in order to make good progress through the testing tasks.

## **Automated Build Testing of NFSv4 Patches**

The NFSv4 development community uses a patch-oriented development process similar to the Linux kernel. Code changes are published in the form of 'diffs' from a baseline kernel release. About once every week or two, the team produces a combined 'patch set' consisting of all of the individual patches in one package. Periodically, the Linux kernel maintainers integrate some or all of the patches into the kernel, allowing the NFSv4 developers to simplify the patch set.

One of the consequences of this approach is that nightly builds cannot be done, as is common in other open source projects. Despite this, it is possible to do per-patch-set builds, which tends to be equally effective at identifying and highlighting compilation issues.

Thus, an early effort was placed into adding the NFSv4 client and server patches for the Linux kernel to OSDL's Patch Lifecycle Manager (PLM). PLM is a system that watches certain web or ftp sites for new kernel patches, downloads them, and runs various cross-compile tests against them [31]. A cross-compiler compiles code for a different architecture than the compiler is running on; often, when a developer is coding for a particular architecture, errors will be introduced that are not seen by that architecture's compiler. Thus, by compiling for a variety of platforms, these sorts of issues are flagged as warnings or errors, and are noted in the compiler output.

For NFSv4, we performed these cross-compile tests on both the base kernel, and the kernel with the new NFSv4 patch applied, and compared the compiler output using diff. This revealed all warnings and errors that the NFSv4 patch would have added to the kernel code.

By collecting this information for each patch, and reporting it to the development mailing list, the issues received attention quickly, and within a month we found that the developers were able to eliminate all warnings and errors on all platforms we were testing. Now, it is rare for a warning or error to last more than a few patch releases before it gets resolved.

## **Automated Regression Testing**

OSDL has a depth of experience with development of automated testing harnesses for the main Linux kernel [32], and thus an obvious contribution was to perform automated regression testing of NFSv4.

The challenges for automating NFS testing compared with regular kernel testing were a) that as a network protocol it required a client/server arrangement rather than running tests on one system under test (SUT), b) that it involved testing of code beyond the kernel, including authentication libraries and network utilities, and c) that it required the ability to vary network conditions. This required creating a new harness able to do network testing, but elements from previous automation harnesses were reused.

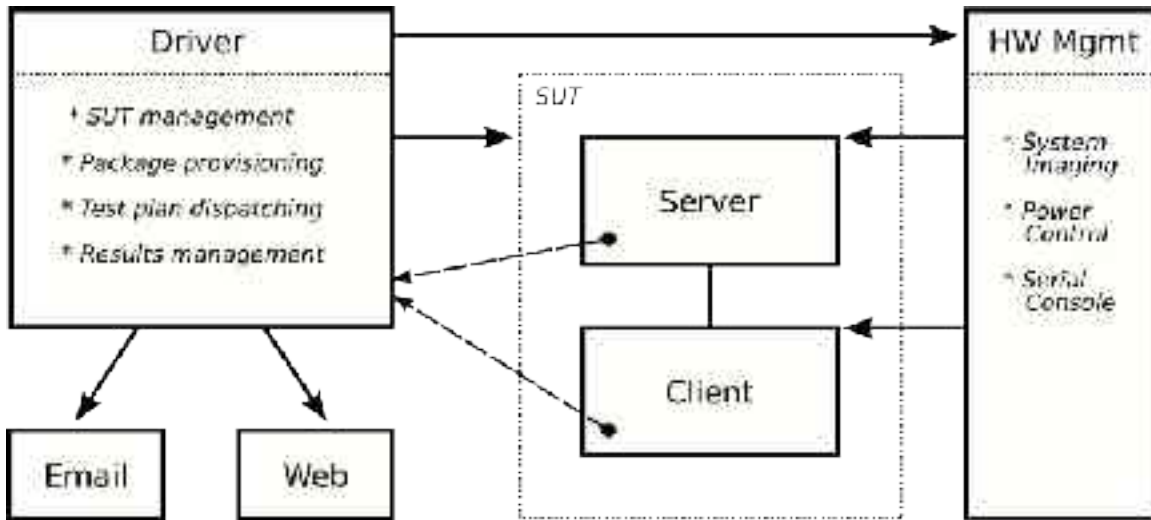


Figure 3: NFSv4 regression test harness

The automated testing system's architecture, as illustrated in Figure 3, involves five different types of machines:

1. SUT Client - This machine acts as an NFS client for testing purposes. In general it is configured with the same software complement as the server, but does not run the NFS daemons.
2. SUT Server - This machine acts as the NFS server, running the `rpc.gssd`, `rpc.svcgssd`, `rpc.idmapd`, `rpc.mountd`, and `rpc.nfsd` daemons. The server currently only has one physical client connection, however in theory it could be extended to multiple client and/or multiple server machines if needed.
3. Test Driver - This machine is used to direct the SUTs and oversee the running of the test processes. It also provides an NFSv3 mount point for the SUTs to store their logs to and to share software packages and test scripts.
4. Hardware Manager - This machine provides serial console logging, remote power control, remote imaging, and other hardware-level features. It is kept separate from the Test Driver for security purposes, since this machine also provides these services to other projects in the Lab.
5. Web/Email Server - This machine sits on the OSDL extranet and provides world-readable access to test reports, test logs, tests, and other materials. It also hosts a bug tracker and a wiki.

The NFSv3 mount provided by the Test Driver deserves additional discussion. This mount is not part of the testing process, but rather is used as the primary means of data communication between the Test Driver and the SUTs. The advantage of this design is that it keeps the Driver-SUT interaction simple and straightforward. Since test logs are written directly to the mounted file system, they can be watched remotely even if the SUT is running under heavy load. This also ensures that the SUT gets into a failure state, the logs up to the failure will be captured, and does not depend on the machines continued functionality after its failed.

An obvious risk is that if an issue is introduced with the NFSv4 code that affects NFSv3 behavior, this could introduce a failure into the testing process itself. However, from a testing perspective, this would be a good thing since our goal is to uncover such problems and fits the 'scratching your own itch' aphorism. In practice we have not seen this as a problem, but it forces the tester to give extra consideration to the NFS overhead, performance, and reliability.

The automated testing system is currently used only for performing regression testing. Four test suites are currently being run: Connectathon, PyNFS, LTP, and Iozone. The Connectathon test suite [33] is a standard NFSv3 test developed by Sun for their annual integration testing event held in San Jose. PyNFS [34] is a test suite supporting NFSv4-specific features maintained by the Center for Information Technology Integration (CITI) at the University of Michigan. The Linux Test Project (LTP) [35] is a large test suite collecting many different regression test cases for Linux; for the NFSv4 testing only the NFS-specific tests are executed. Iozone [36] is a commonly used file system stress test. Other tests are planned to be added to the testing framework in the future.



One of the first requests OSDL received from its member companies when becoming active with the NFSv4 testing community was to help in establishing a bug tracking capability. Our initial investigation showed some resistance to the idea, due to various reasons:

- Risk of complicating reporting processes
- Risk of extra overhead imposed on developers
- Unfamiliarity with bug tracking best practices
- Previous bad experiences with bug trackers in corporate development settings

Most tellingly, the NFSv4 project had already tried two bug tracking systems previously, and neither "caught on", thus creating a large amount of skepticism about bug tracking in general.

Due to these reservations, our initial attempt at establishing bug tracking began to meet resistance. Rather than push it, and alienate the community, we opted instead to focus on using the approaches the processes the community was already comfortable with. Prior to the bug tracker discussions, defect reports were simply sent to the mailing list for discussion and archiving; we acknowledged that this was an acceptable process for current needs.

But as the months passed, testing efforts began revealing a number of issues, many of which were important but not immediately solvable. The question of bug tracking was raised once again, and this time the development community was much more open to the idea. Even among those who had been skeptical before, there was a willingness to give it a try.

The bug tracker was implemented in June of 2005 and in its first two months of use has seen an increasing amount of activity. It is too early to call it a success, but the level of adoption by the core developers suggests that it has strong potential.

## **Building a Community-owned Testing Methodology**

There are a number of challenges to the community approach. Unlike traditional testing, where a single company owns the process and employs the staff to perform it, in wide, community-driven testing processes it can be difficult to get every area filled.

Also, with open source the distinction between developers and testers is much more blurred. This can sometimes result in more emphasis placed on development than on testing. For NFSv4, a balance must be struck that includes strong emphasis on both testing and development.

A third challenge is the sheer complexity of the NFS code stack. In addition to the NFS client and server code in the Linux kernel, there is a surrounding layer of utilities, administrative tools, underlying file systems, add-ons like automounter and cachefs, and authentication services. Most of these components are maintained by people outside the NFSv4 community. Interactions among these pieces and NFS need thorough testing, but with the different versions and configuration settings, have huge numbers of permutations. Opening participation in testing to a wider user community will help distribute the effort and help identify the areas where complexity is an inhibitor.

However, these challenges dovetail with the strengths of corporate-backed testing efforts. Since many of the areas needing assistance happen to be areas that corporate users have a vested interest, those testing efforts will be easily justifiable as priorities for them. Companies are accustomed to dedicating employees to testing and have developed procedures for organizing large scale test teams. They can scale their contribution to match their business needs, thereby providing an effective way to address complexities - if a given company needs a particular set of interactions tested thoroughly, then the business case will exist to justify funding a testing effort to do so. The Bull contributions to testing NFS is an example of this practice.

Establishing a clear, well-organized, structured testing effort in the open NFSv4 community will enable these organizations to better participate in conducting the testing; they can focus on their own priorities. By encouraging them to share their results openly, NFSv4 as a whole will be improved.

## IMPROVING OPEN SOURCE TESTING

Like documentation, testing is an activity that the open source community as a whole continually wrestles with. For example, at the 2005 Ottawa Linux Symposium, David Jones of Red Hat gave a keynote that focused exclusively on the need for more testing and better testing tools. Andrew Morton has also spoken extensively about the need for more testing of the Linux kernel.

The experience seen at the outset of the NFSv4 testing effort seems to echo the situation in many other projects. People recognize that more “testing” is needed, however beyond this it is often unclear what needs to be done. Based on the test needs analysis done for NFSv4, several issues we identified will be relevant to open source testing in general.

### Improving Tests

Automated test frameworks are only as good as the tests that they run. Unfortunately, while a variety of general purpose tests exist, many are not actively maintained, or tend to test code that isn't changing very frequently. Jones pointed out that the Linux Test Project (LTP) in particular has not proven to be as useful as it could be, because its tests cover areas of the Linux kernel that no longer change much.

Test automation should not be viewed as something that is completed once the framework has been built; rather, creating the harness is merely the starting point. New test cases and scenarios that exercise new areas of the code must be added with regularity in order to identify new opportunities for improvement. The effort required to create these tests is worth it when the test will be run repeatedly on all future versions of the source code.

### Improving Bug Trackers

Jones also pointed out the need for better intercommunication between bug trackers. Many projects do a good job at tracking defects in the project's code, but often a defect will be found to belong to another project. Currently, procedures to transfer or share a defect with another project are very ad hoc [37]. One idea for improving this situation would be to adapt all of the major bug trackers to implement something analogous to Rich Site Summary (RSS) feeds that would provide a publishing/subscription mechanism. This would allow a Linux distribution and an individual open source application to share the defect reports of mutual relevance, or for an application project to migrate defect reports they've isolated to an upstream library to that library's maintainer.

An even more ideal system might encapsulate individual defects in such a way that they could be easily distributed to and shared by any number of projects. A given person should be able to review defects from several sources aggregated into one single bug tracking application, much as one reads multiple mailing lists from one mail client. A company should be able to participate directly and efficiently in all the bug repositories of relevance to their products or services, and track additional information on top of it (such as contact information of clients experiencing the issues).

Another area of potential improvement in bug tracking is in allowing better collaboration on the description of the defect. Oftentimes, a non-trivial defect will accumulate many, many replies, some of which are irrelevant or are superseded by later discoveries. A useful system might allow the description section of the defect report to be freely edited in a wiki-like fashion, to enable multiple people to keep the top level description up to date and correct, so that as new participants join the bug discussion, they will be able to quickly and accurately learn the status, before reviewing the discussion thread.

### Improving Testing Tools

Open source testing could also be greatly improved by the creation of better testing tools. There are a number of static analysis tools such as lint, oprofile, gprof, sparse, or valgrind. In many cases these tools need further work to enable them to remain relevant for new programming language features. There are also a number of dynamic analysis tools such as ethereal for network communication analysis, yet there are many aspects of testing for which no dynamic analysis tools yet exist in open source [38]. For example, security testing is an area where tools are particularly desperately needed; most of the good security tools are proprietary and not easily available to open source developers.

## Improving Project Metrics

In addition to testing tools, project metrics can be valuable ways to identify problems and to quantify quality improvements in an open source project. A variety of tools exist for performing line count analysis, measuring defect fixing progress, and so forth. Unfortunately, these tools tend not to be available in an integrated fashion, and the overhead of finding the tools and setting them up is often higher than can be afforded by resource-limited projects. Ideally, these tools would be included as part of project hosting services such as SourceForge, or integrated into the project's version control system so they can be run automatically, with results (and graphs) posted on a regular cycle.

## Improving Community Involvement in Testing

Of course, the best tests and tools in the world will be no good if they are not used. Thus a critically important need is for increased involvement from the community in performing synthetic testing. As is being seen with NFSv4, synthetic testing is an excellent way to identify classes of problems that users would be unlikely to report. This testing, such as going through an application and trying out every option, or attempting to run the program on a random collection of input files, or pushing the program to its limits, is certainly within the skill of ordinary users, it simply requires some dedicated time to do. As has been found with commercial software development, it can be difficult to recruit people to fill these roles [39].

The need for more testers is quite serious and deserves much more attention than it gets currently. Jonathan Corbet pointed this out in his keynote at the Ottawa Linux Symposium (OLS) 2005, remarking that the past issues of patch management in the Linux kernel have been solved through tools like Bitkeeper and git, enabling an extremely high rate of change in the codebase, and that as a result testing this profusion of code changes is the bottleneck today. Efforts to attract and retain testers, test developers, and toolsmiths are vital for the continued growth of that project, and should serve as important lessons to other open source projects.

## Improve “Ecosystem”-level Testing

As mentioned earlier, bugs live in cracks between applications, in the form of interoperability issues, file format incompatibilities, differences in optimization approaches, API changes, and so forth. Many open source projects barely have the resources to test for issues within their codebase, let alone testing against other code “in the wild”. Fortunately, the style of open source user testing is effective at quickly finding these sorts of issues as the software is distributed, however this can often be a frustrating experience for users who expect the program to at least install and run within their environment.

At OSDL, we have experimented with automated systems to test a given program against versions of other software, other distros, and so forth. This tends to be a challenging task, since it requires care in managing a variety of configurations, and in that each new “degree of freedom” added ends up multiplying the system complexity.

It may be possible that recent developments in automated provisioning and virtualization such as Xen may give better ways of managing this complexity. More research into using technologies such as these for (semi-)automated testing is needed.

In addition, encouragement of cross-project communication can help mitigate the problems. For instance, two GUI applications that users commonly use together could collaborate on a review of their keyboard shortcuts or menu layout, and identify areas where consistency could be improved. Or upstream/downstream projects, such as an application and its dependent libraries, could arrange automated mechanisms for sharing defect reports or to coordinate API changes.

## Improving Testing for Small Projects

From a testing point of view, a systemic problem exists in the open source community in the reality that many projects are far too small and too limited of resources to be able to implement any but the most basic testing practices.

A number of open source programs are maintained by individual developers. Many more do not have an active maintainer. Yet often these small projects fulfill critically important areas in software stacks. This situation can arise due to a wide variety of causes, but they impose a weakness on the open source community because defects fail to get reported and/or fixed, sometimes to the degree that it inhibits the software stack as a whole; a chain is only as strong as its weakest link.

An approach to help address this problem is to provide services that can be quickly and easily applied by small projects to allow them to gain easy test results. For example, a user could paste in a URL of a random piece of software, and the system would download the code, identify and install dependencies, compile the code, run a battery of general purpose tests against it, and then generate and email a summary report. By highly automating this process, it reduces the burden of testing and thus encourages its use.

Another idea for addressing this issue is to encourage larger, more active projects to share access to their testing frameworks with smaller projects that develop tools and libraries that the application depends on.

## CONCLUSION

From a testing perspective, open source methodologies are considerably different from traditional, formal methods, yet have been shown to be effective in multiple instances. Even so, they do not obviate the value of performing formal testing as well.

The testing for the new version 4 of NFS is proving to be a great opportunity to tap both open source and traditional styles of testing to strengthen its quality [40]. The use of per-patch cross compile builds and a bug tracker in conjunction with the NFS user community shows the strengths of the open source method for achieving quality improvements. Coupled to that are dedicated testing efforts by Bull and automated testing efforts by OSDL that employ more traditional methods such as direct regression, performance, and robustness test runs against the codebase.

The growing adoption of open source technologies by companies and the need for better tested software suggests that some of the things learned in the NFSv4 effort may have broad applicability for other, similar efforts. By making testing easier and more rewarding to do, we all can benefit from having better software to rely on long into the future.

## ACKNOWLEDGMENTS

Special thanks to reviewers Richard Vireday, Kees Cook, Craig Thomas, and Mary Edie Meredith, and Eric Schnellman. This work was supported by the Open Source Development Labs.

## REFERENCES

1. <<http://nfs.sourceforge.net>>
2. Thurlow, Rob, et al. "AFS/DFS – Migrating to NFSv4," *NFS Industry Conference*, Sep 2003. <<http://www.nfsconf.com/pres03/thurlow.pdf>>
3. <<http://www.citi.umich.edu/projects/nfsv4/linux/>>
4. "Open Source: Open for Business," *Leading Edge Forum Report*, 2004. <<http://www.csc.com/features/2004/48.shtm>>
5. Raymond, Eric. "The Cathedral and the Bazaar," 2000 <<http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>>
6. Massey, Bart. "Why OSS Folks Think SE Folks Are Clue-Impaired," *Proc. Workshop on Open-Source Software Engineering, 2003 International Conference on Software Engineering*, Portland, OR, May 2003. <<http://www.cs.pdx.edu/~bart/papers/icse-osse.pdf>>
7. Schmidt, Douglas C., and Porter, Adam. "Leveraging Open Source Communities to Improve the Quality and Performance of Open Source Software," *1st Workshop on Open Source Software Engineering*, ICSE, Toronto, Canada, May 2001. <<http://www.cs.wustl.edu/~schmidt/PDF/skoll.pdf>>
8. Erenkrantz, Justin R. "Release Management within Open Source Projects," Institute for Software Research.
9. Wheeler, David A. "Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!", Mar 2005. <[http://www.dwheeler.com/oss\\_fs\\_why.html](http://www.dwheeler.com/oss_fs_why.html)>
10. Miller, Barton P., Fredricksen, Lars, So, Brian. "An Empirical Study of the Robustness of UNIX Utilities," National Science Foundation Technical Report, 1990. <<http://www.cs.wisc.edu/~bart/fuzz/fuzz.html>>
11. Ge, Li, Scott, Linda, and VanderWiele, Mark, "Putting Linux reliability to the test" Linux Technology Center Technical Report, Dec 2003. <<http://www-106.ibm.com/developerworks/linux/library/l-rel/>>
12. Vaughan-Nichols, Steven J. "Can you Trust this Penguin? What's Wrong (And Right) With LINUX" ZDNet, Nov 1999. <<http://web.archive.org/web/20010606035231/http://www.zdnet.com/sp/stories/issue/0.4537.2387282.00.html>>
13. Godden, Frans. "How do Linux and Windows NT measure up in real life?" ID-side, Jan 2000. <<http://gnet.dhs.org/stories/bloor.php3>>
14. Lemos, Robert. "Security research suggests Linux has fewer flaws," CNET News.com Dec 2004. <[http://news.com.com/Security+research+suggests+Linux+has+fewer+flaws/2100-1002\\_3-5489804.html](http://news.com.com/Security+research+suggests+Linux+has+fewer+flaws/2100-1002_3-5489804.html)>
15. "Sites utilizing Microsoft Webserver-Software turn in inferior reliability results in study of Swiss websites," SysControl AG press release, Feb 2000. <<http://web.archive.org/web/20011011215009/http://www.syscontrol.ch/e/news/Serversoftware.html>>
16. <<http://www.dwheeler.com/frozen/top.avg.2001aug3.html>>
17. Samoladas, Ioannis, et al. "Open source software development should strive for even greater maintainability", *Communications of the ACM* Volume 47, Number 10 <<http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=240>>

18. Kaven, Oliver. "Performance Tests: File Server Throughput and Response Times," PC Magazine, Nov 2001. <<http://www.pcmag.com/article/0,2997,s%253D25068%2526a%253D16554,00.asp>>
19. Howorth, Roger and Stevens, Alan. "Samba runs rings around Win2000," VUNet.com, Apr 2002. <<http://www.vnunet.com/News/1131114>>
20. Dyck, Timothy. "Server Databases Clash," Eweek, Feb 2002. <<http://www.eweek.com/article2/0,3959,293,00.asp>>
21. Reavis, Jim. "Linux vs. Microsoft: Who Solves Security Problems Faster?" <<http://web.archive.org/web/20010608142954/http://securityportal.com/cover/coverstory20000117.html>>
22. "New Evans Data Survey Reports Security Breaches Rare in Linux Environment," Evans Data Corp. Press Release, April 2002. <[http://www.businesswire.com/cgi-bin/f\\_headline.cgi?bw.040802/220982285](http://www.businesswire.com/cgi-bin/f_headline.cgi?bw.040802/220982285)>
23. Pescatore, John. "Commentary: Another worm, more patches," Gartner Viewpoint, 2001. <<http://news.cnet.com/news/0-1003-201-7239473-0.html?tag=nbs>>
24. Leyden, John. "Zombie PCs spew out 80% of spam," *The Register*, June 2004. <[http://www.theregister.co.uk/2004/06/04/trojan\\_spam\\_study/](http://www.theregister.co.uk/2004/06/04/trojan_spam_study/)>
25. <[http://news.netcraft.com/archives/2005/08/01/web\\_server\\_survey\\_turns\\_10\\_finds\\_70\\_million\\_sites.html](http://news.netcraft.com/archives/2005/08/01/web_server_survey_turns_10_finds_70_million_sites.html)>
26. <<http://www.redhat.com/software/rhel/details/>>, Aug 2005.
27. <<http://www.debian.org/>>, Aug 2005.
28. <<http://www.freshmeat.net/stats/>>, Aug, 2005.
29. Halloran and Scherlis "High Quality and Open Source Software Practices," School of Computer Science, Carnegie Mellon University.
30. <<http://nfsv4.bullopensource.org/tools/tests/page24.php>>
31. Lebzelter, Judith. "Open Source Testing Framework for Handling a Multiple Product Stack," *Pacific Northwest Software Quality Conference*, 2004.
32. Dabney, Nathan. "The Scalable Test Platform," *Linux Journal*, Nov 2001.
33. Connectathon test suite. <<http://www.connectathon.org/nfstests.html>>
34. PyNFS test suite. <<http://www.citi.umich.edu/projects/nfsv4/pynfs/>>
35. LTP test suite <<http://ltp.sourceforge.net/>>
36. Iozone <<http://www.iozone.org>>
37. "Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community" Sandusky, et al, University of Illinois at Urbana-Champaign.
38. "Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools" Jason Robbins, University of California, Irvine.
39. "Modeling Recruitment and Role Migration Processes in OSSD Projects" Chris Jensen and Walt Scacchi, Institute for Software Research, Bren School of Information and Computer Sciences, University of California, Irvine.
40. De la Torre, Lynne and Harrington, Bryce. "NFSv4 Testing: Preparing for Enterprise Deployment," *LinuxWorld Magazine*, May 2005. <<http://linux.sys-con.com/read/86018.htm>>

## APPENDIX A: NEW FEATURES IN NFS VERSION 4

- Tracks file state. Unlike prior versions of NFS, in NFSv4 file state (locking, reading, writing) is tracked between the client and server
- Permits lease-based locking. Allows the client to take ownership of a file for a period of time; it must contact the server to extend this lease
- Allows file delegation. NFSv4 servers can allow NFSv4 clients to modify cached files without contact to the server, until the server notes that another client needs it and issues a 'callback'
- Implements compound RPCs (Remote Procedure Calls). Multiple NFS operations (LOOKUP, OPEN, READ, etc.) can be combined into a single RPC request, thereby minimizing network round trips and thus improving latency
- Supports security flavors. A number of sophisticated security mechanisms including Kerberos 5 and SPKM3 are implemented, and APIs are available for adding new security mechanisms down the road
- Supports ACLs. On POSIX systems and Windows, NFSv4 standardizes how ACLs are used. Named attributes are also added, allowing user and group names to be accessed as strings, not just numeric IDs.
- Combines several distinct NFS protocols (stat, NLM, mount, ACL, and NFS) into a single protocol specification to allow better compatibility with network firewalls
- Supports file migration and replication