A comparison of system monitoring methods, passive network monitoring and kernel instrumentation

A. W. Moore[*], A. J. McGregor[†] & J. W. Breen[‡]

Abstract

This paper presents the comparison of two methods of system monitoring, passive network monitoring and kernel instrumentation. The comparison is made on the basis of passive network monitoring being used as a replacement for kernel instrumentation in some situations. Despite the fact that the passive network monitoring technique is shown to perform poorly as a direct replacement for kernel instrumentation, this paper indicates the areas where passive network monitoring could be used to the greatest advantage and presents methods by which the discrepancies between results of the two techniques could be minimised.

1 Introduction

The use of file system monitoring in general, and comprehensive kernel monitoring techniques in particular, have laid the critical groundwork for the development and refinement for many operating systems. Kernel instrumentation has the potential to give an exact record of what occurred in the kernel of a system and, as a result, is commonly used when high precision is required.

It has been used, in studies such as Ousterhout et al. [20], Smith [28], Mummert and Satyanarayanan [17, 29] and Baker et al. [2], to record information about an operating system and its file systems. The results and conclusions of these studies have then been used for studies of topics such as cache issues and simulation models [32, 7, 8, 33], and in the design process of new systems [11, 19, 13, 10, 21].

However, while there is a large variety of systems in common use, a similarly wide variety of comprehensive studies is not evident, which can be attributed to the difficulties in performing such studies. Comprehensive studies using kernel instrumentation have a number of drawbacks, as seen in the following list (adapted from Mogul et al. [16]):

- code which is to reside in the kernel is difficult to write and debug,
- kernel source-code is not always available,
- the kernel must be recompiled and the machine rebooted each time an error is found,
- errors in the kernel code are likely to cause system crashes,
- functionally-independent kernel modules may have complex interactions over shared resources,
- kernel-code debugging cannot be done during normal machine operation; specific development time must be scheduled, resulting in inconvenience for users sharing the system and odd work hours for system programmers,
- commonly additional load is introduced onto the monitored system,
- sophisticated debugging and monitoring facilities such as those available for developing user-level programs may not be available for kernel code.

Kernel instrumentation for file-system monitoring takes the form of code inserted at the system-call interface or at the internal interface between

[*]Department of Robotics and Digital Technology, Monash University, Clayton, Victoria 3168, Australia (andrew.moore@rdt.monash.edu.au).

[†]Department of Computer Science, Waikato University, Private Bag 3105, Hamilton, New Zealand (T.McGregor@cs.waikato.ac.nz).

[‡]Department of Robotics and Digital Technology, Monash University, Clayton, Victoria 3168, Australia (j.breen@rdt.monash.edu.au).

system-call and file-system operations. Such methods suggest an alternative technique for monitoring a computer's file-system activities where the communications channel between a machine and its disk drives, and in particular between a diskless client and its disk server, is passively monitored. Blaze used this technique with his `snooper/rpcspy` software [3], passively monitoring traffic between Network File System (NFS) [24] clients and servers, and predicting the operations the clients performed to cause those operations.

Full kernel instrumentation is used commonly in system monitoring but, by definition, it involves the modification of the operating-system source-code for the machine in question. Passive network monitoring can be a preferred choice over kernel instrumentation for certain system-monitoring work, particularly if the source-code is not available. Passive network monitoring also has other advantages, including:

- results from the machines being monitored can be collected independently of those machines,
- no modifications are required to the operation of the monitored systems,
- the collection of data with passive network monitoring does not impact on the machines being monitored, and
- the ability exists to monitor multiple machines simultaneously on a network.

This final point is important because distributed systems are growing in popularity and a significant number of computers in common use are part of a distributed system, if only through the distribution and sharing of files. Comprehensive studies of distributed systems in general and distributed file systems in particular are relatively rare. The main reason for this is that the complexity of collecting data is greatly exacerbated by the need to collect it simultaneously from a large number of machines. Kernel instrumentation would require modification of any number of different computers with different operating systems running on different hardware. There would also be the issues of the load imposed in the actual collection of data on or from each client and the immense task of post-processing the data from the different machines.

These disadvantages do not exist when using passive network monitoring of the data channel between clients and a server. Data about all active clients can be collected simultaneously and, if a distributed file systems such as NFS is in use, the data collected from the network is independent of the operating system or machine architecture [15, 14].

2 `rpcspy/nfstrace`

If `rpcspy/nfstrace` are to be used as a replacement for kernel instrumentation, the technique must be able to deliver to a researcher data similar to that generated by the kernel instrumentation techniques. `rpcspy/nfstrace` has two distinct components for achieving this.

`rpcspy` interacts with the ethernet-interface facilities of the monitoring machine and collects packets traversing the network to which it is connected. The packets are converted into the request or reply part of an NFS Remote Procedure Call (RPC). Each request and reply is then matched together, data of interest are extracted and a transaction record is made along with a time stamp of when the transaction was completed.

The second component, `nfstrace`, uses an heuristic based on the operation of NFS to make an estimation of the duration of a file's open-close session (the time between when a file is first opened, read from and/or written to and then closed) which caused the NFS transactions recorded by `rpcspy` to occur. `nfstrace` creates records of file open-close sessions it estimates have occurred (and, thus, to have generated the NFS transactions seen). This estimation relies, in part, on consistency in NFS implementations.

For example, for every `open` system call (independent of whether the file is to be read to and/or written from or just accessed) an NFS `getattr` transaction is generated. However, open-close sessions to `read` or `write` data handle the actual data in significantly different ways although it should be noted that the `write` transaction case is easier to handle because the cache does not have as dominant effect on the `write` operations. As a result, much of the special-case handling `nfstrace` must do applies only to NFS `read` transactions.

`rpcspy/nfstrace` together generate an estimation of the open-close sessions that a kernel instrumentation system could record directly. Figure 1

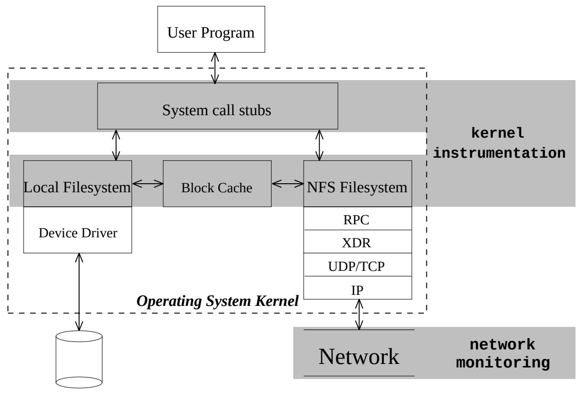


Figure 1: The data flow between a user program and an NFS file system. Instrumentation points for kernel instrumentation (`snooper`) and network monitoring (`rpcspy`) are indicated. This diagram compares the difference in the information available to each system. In particular, one instrumentation point, `snooper`, is before the cache and the other, `rpcspy`, is after the cache.

shows how the instrumentation points for each of the two systems differ in the components of the system available to them and, thus, the information accessible to them.

Kernel instrumentation is able to record events such as the system calls occurring on the computer, while passive network monitoring must interpret the transactions between client and server to estimate which system operations have occurred. `rpcspy /nfstrace` results record what happened on the server side of the client cache - the traffic that occurred between client and server.

3 Previous work using `rpcspy /nfstrace`

The `rpcspy/nfstrace` implementation by Blaze has been used in several different works to aid in the development of new systems and as an aid to the configuration of existing systems. Anderson [1] used `rpcspy/nfstrace` to analyse client-server file system traffic and made use of information about the traffic to better utilise local disks in the clients themselves. Regularly-read, static, read-only files were moved to the local disk, on a partially automatic basis, taking into account each file's utilisation by a particular workstation. Blaze uses results collected using `rpcspy/nfstrace` in a number of works to justify the design of a wide area file system [4, 5]. Finally, Dahlin et al. [9] uses `rpcspy` and a partially-modified version of `nfstrace` to collect results used in a paper to justify a particular file system design [34].

Each of these studies has been made with the assumption that complications introduced by the results of `rpcspy/nfstrace` are negligible. This is valid for studies based upon the traffic between a client and server which take into account the effects of the client cache such as Dahlin et al. [9]. However, other studies can be at risk for assuming that `rpcspy/nfstrace` is such a *perfect* replacement for traditional techniques.

Blaze [4] notes in his description of the `rpcspy` software that the effects of packet loss should be quantified. Additionally, he notes in his description of the `rpcspy/nfstrace` software that peculiarities of the heuristics of nfstrace (and its original implementation) need to be evaluated more completely.

Previously, the use of `rpcspy/nfstrace` has been without any hard data on the accuracy of the implementation and only a passing appreciation of areas where the implementation is inaccurate and the reasons for those inaccuracies. The following sections present a comparison of two systems recording data from the same source and then discuss the results.

4 Systematic error in `rpcspy`

The `rpcspy/nfstrace` tools depend heavily on the ability of the network interface of the machine on which they are being run to capture all traffic passing through the network. Packet-loss by the network interface does not have a linear relationship with network utilisation. The network interface will not lose data when utilisation is low but data loss will increase as utilisation increases to a point beyond which it will be unable to accept any further increase in the data-transfer rate. The amount of data it can process will flatten out no matter what the utilisation beyond that point.

A study was performed to quantify the potential data loss of `rpcspy` and to calibrate the network interface `rpcspy` uses. To perform these tests satisfactorily, a network analyser capable of full-utilisation measurements on Ethernet was required. A Hewlett Packard Internet Advisor Model J2522As was used both to make measurements and to generate artificial loads on the network. The `packetfilter` mechanism used was in a DECstation 3100 running Ultrix 4.3a and the `NIT` mechanism used was in a Solbourne SC2000 (a machine compatible with the Sun Sparcstation 2) running a SunOS 4.1.2 compatible operating system.

Tests of `rpcspy`, where the network was loaded artificially, used the traffic breakdown in Table 1 which was based on an analysis of the network's regular traffic content collected over several 24-hour periods.

The `packetfilter` facility of Ultrix offers some configuration options. In particular, the size of the packet buffer, where packets processed by `packet filter` are placed for collection by the user process, can be set. The `NIT` mechanism in SunOS does not offer this configurability. The default configuration and an optimum (largest configurable buffer size) for `packetfilter` in addition to the `NIT` mechanisms are compared in Figure 2. This figure shows the percentage of unprocessed Ethernet packets versus Ethernet network utilisation. It is apparent that not only are the characteristics of the `NIT` mechanism poor beyond 10% utilisation but that the `packet filter` mechanism did not demonstrate the same level of loss until utilisation was close to 50%. The `packetfilter` mechanism showed no loss until over 15% utilisation, a stage by which `NIT` mechanism loss was close to 25%.

A significant issue in `rpcspy` is the combination of processing overhead on the client, which is imposed by the need of `rpcspy` to match RPC transactions, and the packet-loss characteristics of the Ethernet interface which `rpcspy` is using. Figure 3 shows the number of NFS transactions versus Ethernet utilisation. The Ethernet utilisation in these tests is almost purely NFS traffic. By using NFS traffic exclusively we are able to establish the maximum number of NFS transactions each `rpcspy` system is able to process in a given time period. The Hewlett Packard test equipment recorded the actual number of NFS transactions that occurred over this time. For this test the `packetfilter` was left in the default configuration.

The test shows that each system has a maximum number of packets it can process. The `NIT`-

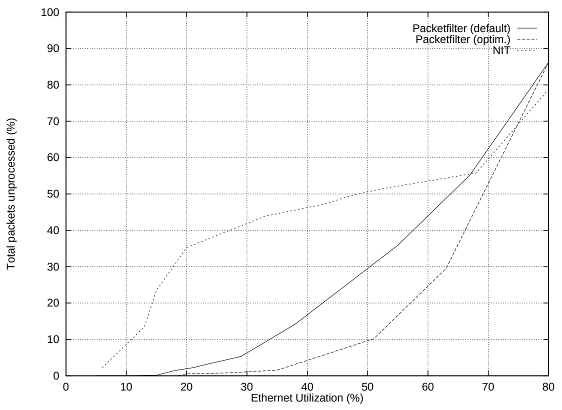


Figure 2: A comparison of Ethernet utilisation versus packet loss for various workstation Ethernet interfaces. `packetfilter` default and optim(um) are two configurations of the Ethernet packet capture facility of the Digital DECstation, `NIT` is the Ethernet capture facility in Sun Microsystem's SunOS.

SunOS system is limited to processing about 175 NFS transactions per second. The default configuration `packetfilter`-Ultrix combination appears to be limited to processing approximately 260 NFS transactions per second. It is important to note this was a stress-testing of `rpcspy` and that such NFS loads were not a characteristic of the network to which these machines were connected. From the figures in Table 1 we can see that 36% of the total Ethernet traffic is from NFS. However, it would not be true to say of this 36% that half the number of NFS Ethernet packets is an approximate count of complete NFS transactions. Such a simplification would not allow for there being incomplete NFS transactions (the loss of the request or reply in a transaction), nor would it allow for NFS transactions that required more than one pair of network packets (transactions where the data payload required two or more Ethernet packets). In each of these cases `rpcspy` does not need as much processor time as if it had had a complete NFS transaction. As a result, the test network operating at 12% utilisation could mean less than 72 transactions per second in a mixed load with a variety of NFS traffic rather than the 200 transactions per second that the Figure 3 stress-test indicates.

The exact cause of such data loss is not known for certain but it could result from limitations in the hardware of the network interface and/or in the

Protocol type		Sub-protocol		Types of packet	Packet size (avg.)	%
Internet Protocol (IP)	67	UDP	36.9	NFS	155	24.7
					1500	12.2
		TCP	30.1	(all)	80	15.1
					192	9.0
					1272	6.0
Novell Netware (IPX)	33	-		-	155	19.8
					768	13.2

Table 1: A breakdown of the traffic mixture used for testing `rpcspy` response to Ethernet utilisation

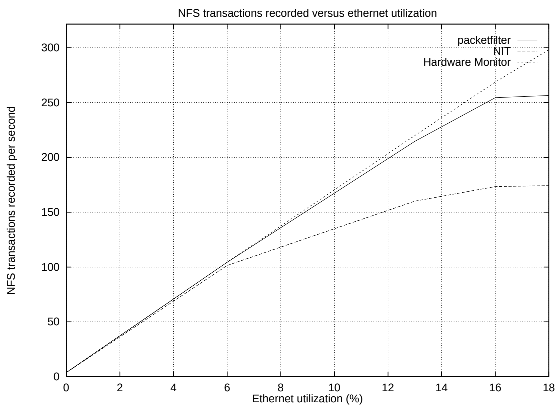


Figure 3: The number of NFS transactions versus Ethernet utilisation for the `NIT` and `packetfilter` capture mechanisms. Results from a network analyser recording no packet loss is also given.

software of the packet collection and filtering mechanism. This characteristic is unfortunate. It is during the time when the network is busiest that utilisation across a distributed file system will potentially be highest. Because there is potential for `rpcspy` based tools to lose data about transactions at busy times, studies such as file sharing, a situation that would be more likely to occur at busier times, would be affected adversely.

Such drawbacks could be overcome by the use of faster workstations with faster hardware network interfaces. However, this may not be solved as easily if the problem is due principally to poor software implementation performance in either the network packet capture mechanism (`NIT/packetfilter`) or `rpcspy`.

While this characteristic loss does exist, it is significant only above about 10% utilisation for the `packetfilter` mechanism. Boggs et al. [6] comment that most Ethernet loads are well below 50% (and are actually close to just 5% of the network capacity) and the network on which measurements were taken supports this observation with a maximum load over 24 hours of no greater than 18% and an average utilisation over 24 hours of closer to 1.1%.

5 Comparison

The following results present a comparison between `rpcspy/nfstrace` and a kernel instrumentation technique. In these results there has been an assumption that `rpcspy/nfstrace` would be used as a substitute for kernel instrumentation.

The kernel instrumentation this paper used is the `snooper` package. It was implemented originally by Siebenmann and Zhou [27] for Ultrix version 3.3. `Snooper` is a set of kernel instrumentation routines for recording information about a number of kernel functions including logical file operations, physical-block operations, process execution and termination, etc. The `snooper` package is based upon the package of the same name described in Zhou et al. [35] which, in turn, shares its ancestry with the package used by Ousterhout et al. [20] to perform their study of the UNIX 4.2 BSD file system.

The results from each system were processed into open-close sessions, i.e. sets of file transactions bounded by an open and a close system call. `nfstrace` performs that function for `rpcspy/nfstrace` but additional post-processing code was required for the data produced by `snooper`

The comparisons of `snooper` and `rpcspy/nfstrace` have been done by using simultaneous traces of a single machine over a 24-hour period. The trace of this machine was performed from 11:00 a.m. Monday, 12th of December, 1994, until 11:00 a.m. the following day. The machine traced was a Digital DECstation 3100 configured with 20Mbytes of memory, running Ultrix 4.3a. This machine was configured with a local disk for virtual memory swap activities. The `rpcspy` trace was recorded to an additional local disk so as not to perturb the results with extraneous network activity. During the 24-hour period, a loss of 1.5% of total Ethernet traffic was recorded. Based on the graphs of packet loss versus utilization (figure 2) and upon the average Ethernet packet utilization (table 1) this gives an approximate loss of 0.6% of NFS transactions from the total recorded trace.

The `nfstrace` post-processing tool uses a heuristic which incorporates a timeout to determine how long an open-close session will last. The value is user-selectable but the default value of 135 seconds was used throughout the analysis described herein.

5.1 Excluded data

All transactions associated with the reading of executable files recorded by either the `snooper` or `rpc spy` systems were removed from the trace data before processing. This was done to avoid problems associated with a shortcoming in the `snooper` instrumentation, not the `rpcspy/nfstrace` system. Records pertaining to the `snooper` trace file itself were removed from the output records during the processing stage.

While the removal of all execution transactions may seem to change the results presented, the remaining data still permit a satisfactory comparison of the two monitoring systems. The amount of potential comparison-error which would be introduced due to the inclusion of incomplete execution records by `snooper` was not justified. Additionally, file system traffic resulting from the loading of executable files was excluded from previous studies such as Ousterhout et al. [20] and Baker et al. [2] due to similar problems in the logging of executable file traffic.

5.2 System traffic

The characteristics of the total file-system communications traffic are commonly-used measurements. In the case of diskless workstations, the measurements are important for insuring that the networks have adequate transport capacity and that the servers of diskless workstations have adequate service capacity. In any sort of workstation such values define the required capacity for disk interfaces, as well as being used in cache and bus design [35, 21, 20, 2, 23].

A comparison of communications traffic to and from the file system at the logical level and of the communications traffic at the `rpcspy` network level are not strictly comparable because each set of measurements was made on a different side of the cache. However, one of the objectives of `nfstrace` was to estimate operations that occurred at the user level by analysing the data communications traffic between client and server and the transactions used by the client to ensure the contents of the cache are up to date. As a result, while `rpcspy/nfstrace` cannot generate information on exactly what data were transferred between the user programs and the file system (including the NFS file-system routines and the block cache), it can calculate the exact amount of data transferred by the NFS file system between NFS client and server.

Table 2 gives a summary of results for the comparison period. It is apparent immediately that there is a major difference in the value `nfstrace` estimates for the total data transferred when compared with `snooper`. They differ by a factor of 1.7. From these results it is equally apparent that over the course of a long-term analysis (24 hours) the results for peak values and `write` data are comparable for the two systems.

Peak values display this characteristic because they typically involve amounts of data that are too large or too volatile to be suitable for long term storage in the cache [20, 2, 28] - this characteristic is independent of the particular load a machine is under [18]. As a result, the similarity between transferred data, particularly peak values, would remain across any sample taken. In comparison, values for the total quantity of data transferred over time is not similar. The difference between `snooper` read averages and `nfstrace` values is not surprising. The client cache will eliminate successive NFS transactions for reading data from the NFS server and, as a result, `nfstrace` cannot record the data transfer that occurred at the logical level.

Figure 4 shows plots of data transferred over time as recorded by `snooper` and `rpcspy/nfstrace`. Higher levels of data transfer, particularly significant writing activity, between 7 a.m. and 11 a.m.

Particular measurement	interval length	`snooper` (bytes)	`nfstrace` (bytes)
Total data transferred		86,644,530	46,967,724
Average data transferred	10 seconds	10,028	5,436
Peak data transferred		5,120,000	5,048,320
Average data read		7,468	2,590
Peak data read		5,120,000	3,914,935
Average data written		2,560	2,846
Peak data written		5,120,000	5,048,320
Average data transferred	10 minutes	601,698	326,165
Peak data transferred		19,028,550	17,015,414
Average data read		448,103	155,387
Peak data read		10,427,845	7,144,164
Average data written		153,595	170,777
Peak data written		8,600,705	9,289,091

Table 2: The total data transferred for the system. Peak and average values for 10 second and 10 minute intervals are also given.

is due to the testing of image encoding algorithms (by another researcher) on this machine requiring the reading and writing of large image files.

The graph of read-data shows an example of the difference between data gained from `snooper` instrumentation and that available to `nfstrace`. Periodic accesses by automatic jobs account for the regular communications traffic logged during the 19:00 to 07:00 period. Because this communications traffic involves the regular execution of programs, commonly with little other file-system activity, the cache of the client holds all the necessary software and associated data files. The result is that approximately 300 Kbytes of logical data are read each 30 minutes at the `snooper` level but `rpcspy` records negligible read-activity between client and server over the same period.

The reason `nfstrace` is not as accurate for records of raw data transfer is because NFS transactions do not contain significant information about blocks read from the cache of the client. The only specific `read` data available to `rpcspy` about data transfers that occur is when data are read by the client from the server's disks.

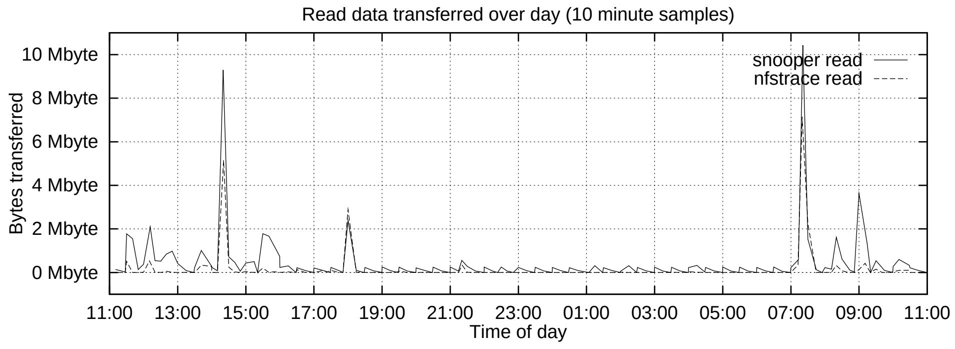
5.3 File system transactions

As with most unix systems, each file system is used typically for a particular purpose. For example, one file system contains the users' directories, another file system contains executable files for the system, etc. The DECstation analysed in this study did not have any local file systems, apart from that used to store trace data locally, and a local, swap disk. Table 3 lists the different file systems the client accessed over the trace period and the tasks each file system served.

A breakdown of the type of data transferred to and from each file system can be used to assist in making file-system-configuration decisions. Such decisions can include which file systems generate so much server traffic that it would be better for them to be attached locally to the machine and how widely a particular file system is used. A breakdown of each file system's communications traffic is given in Table 4.

It is important to note that at the system-call level, as recorded by `snooper`, there is a characteristic breakdown of these transactions. Of particular note is a very large percentage of operations associated with the `/` partition. The large number of transactions on this partition will have been potentially compounded because the `/tmp` and `/var/tmp` directories resided on the `/` file systems. `/tmp` and `/var/tmp` can potentially carry a large percentage of operations because temporary files are traditionally created in this directory structure [31, 22].

Table 4 shows a moderate similarity between the results from the two monitoring methods. Notable exceptions are traffic involving the `/` partition and



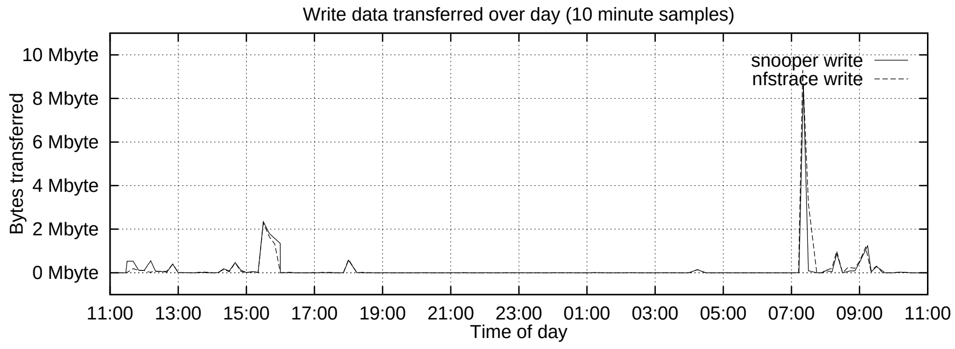


Figure 4: Read and Write transfers as recorded by kernel instrumentation (`snooper`) and network monitoring (`nfstrace`). A quiescent system from 19:00 until 7:00, the machine is busy during the daylight hours.

File System	Function and Contents
`/`	`root` file system, also includes `/var` and `/tmp`. Top-level file system containing temporary directories and logging directories.
`/usr`	contains *standard* software distribution, in addition to libraries and include files for the current system.
`/var/spool/mail`	contains each users' mail file.
`/usr/local`	contains locally installed software.
`/usr2`	home directories for a group of users.
`/packages`	contains commercial software packages and collections of project specific data (in this case image data).

Table 3: A breakdown of the file systems of the study and their respective functions.

File System	snooper		rpcspy/nfstrace	
total				
/	31,736,478	(36.63)	5,863,351	(11.59)
/usr	2,941,480	(03.39)	1,446,003	(02.86)
/var/spool/mail	4,385,788	(05.06)	3,142,239	(06.21)
/usr/local	1,455,692	(01.68)	965,364	(01.91)
/usr2	38,660,513	(44.62)	35,251,413	(69.66)
/packages	7,464,579	(08.62)	3,934,663	(07.78)
read				
/	27,267,823	(42.26)	2,853,302	(12.73)
/usr	2,941,480	(04.56)	1,446,003	(06.45)
/var/spool/mail	3,836,074	(05.94)	2,311,208	(10.32)
/usr/local	1,455,692	(02.26)	965,364	(04.31)
/usr2	21,561,247	(33.41)	10,895,621	(48.63)
/packages	7,464,579	(11.57)	3,934,663	(17.56)
write				
/	4,468,655	(20.20)	3,010,049	(10.68)
/var/spool/mail	549,714	(02.49)	831,031	(02.95)
/usr2	17,099,266	(77.31)	24,355,792	(86.38)

Table 4: Total data, read data and write data transferred per file system as measured by `snooper` and `rpcspy/nfstrace`.

read-traffic in general. While differences between values for read between `snooper` measurements and those of `rpcspy` can be explained as resulting from the cache mechanism filtering read requests between client and server, the read traffic for the / partition is particularly pronounced. This difference is likely to result from a high usage of system files located in the `/etc` directory being accessed, resulting in the corresponding cache entries always being valid. Examples of such files include `/etc/passwd`: the list of users able to use a system, `/etc/hosts`: a static table of the systems known to this client and `/etc/mount`: a file listing the file systems that should be mounted on this client.

The notable difference in the recorded quantities of read and write data for `/usr2` is a reflection of the volatile nature of files on this file system. In particular, software for image encoding was being developed and a cycle of

1. edit program
2. compile program
3. run program

existed. This development cycle, during stage 1, resulted in source-code files being written to the NFS server (and seen by `rpcspy`) but not necessarily read from the NFS server. During stage 2, in addition to the source-code files, libraries will be read only once from the server and then may remain in the local cache while being used repeatedly. Finally, during stage 3, while file transactions relating to the loading of the executable file itself have been removed, this program takes as input a raw image stream and outputs an encoded image stream. On consecutive runs the raw image stream could have remained in local cache.

It should be noted that the ratio of read-to-write traffic already greatly favours the write-traffic for `/usr2` as measured with the `snooper` system but the cache activities, filtering traffic, increase this ratio.

Significant differences between the amount of write traffic recorded by each monitoring system for both the `/usr2` and `/var/spool/mail` file systems can be attributed to the block cache needing to transfer data to and from the file system in block-sized pieces. The result of this is that a modification of one byte in a file will result in the writing of a whole block (8 Kbytes for these file systems).

From this breakdown it is clear that, while activities on the / file system are responsible for a large percentage of logical file traffic, block caching seems to reduce the quantity of data transferred by a factor of up to 6. By comparison, the /usr2 file system is responsible for a higher quantity of data transfer and, in the development and balancing of file systems, it would be important to establish whether this is a transient condition or a regular trend for communications traffic for that particular file system.

5.4 System users

Table 5 presents several values related to the number of active users on the system and the amount of traffic generated by them. Such tabulations have been made in a number of previous studies and are useful in the estimation of the load a user may impose on a system as well as the worst-case scenarios for this load.

The differences in Table 5 for the number of users are most likely the result of `snooper` recording the real User ID (UID) associated with each logical operation and `rpcspy` recording the effective User ID associated each NFS transaction. This difference comes about because programs such as `inetd` (the internet service daemon) perform operations as one user and spawn programs that will run as another user. The result is that counts of active users made through `rpcspy/nfstrace` usually differ by a value of one when compared with the active user count from `snooper`.

Average-data-utilised per user indicates that cache-hit rates are, once again, absorbing a substantial quantity of communications that would have occurred between each user and the file system. It is interesting to note that the maximum values recorded by each system are almost identical. This is most likely due to the transfer of large amounts of data, causing the client's cache to be quickly overrun with new data. As a result, only a minimal amount of data is cached during this time.

5.5 Files

As files are the common unit of data accessed on a file system, information about the range of files accessed, as well as the working size of those files, enables developers to determine the necessary size of file caches, to establish common working-set sizes and to quantify other related measurements.

As has been mentioned earlier, the difference in the average file size for the / file system was predictable. This will principally be a result of a large number of small, system-related files not requiring access from the NFS system. The differences in other values will have resulted from the caching of, and repeated accesses to, active files (even if these files were active for only a short period of time). In this context, an active file is one which is accessed one or more times.

Table 7 lists the number of different files recorded at the `snooper`, `rpcspy` and `nfstrace` levels. At the `rpcspy` level, this is a count of every file that had a read or write NFS operation performed on it. The filtering characteristic of the cache is obvious when comparing the number of files that had logical operations performed on them at the `snooper` level with the number of files for which data was read from or written to at the `rpcspy` level. Larger differences for the / file system will have been as a result of accesses to the large number of regularly-accessed system files located there. These files would be accessed often and be modified infrequently and would, as a result, have a long cache life.

The results in this table show an area where the estimation method used by `nfstrace` can generate discrepancies. `nfstrace` must estimate traffic to and from files that have not caused any `rpcspy` `read` or `write` transactions. With the exception of /var/spool/mail, `nfstrace` must estimate additional operations for files on each of the five file systems. `nfstrace` has estimated extraneous operations on files of /usr2 and underestimated these operations for the other file systems, in particular the / file system.

The rule base under which `nfstrace` operates estimates operations on files from a combination of NFS `read`, `write`, `setattr` and `getattr` transactions. The estimates of files which did not involve NFS `read` or `write` transactions would have resulted from `setattr` or `getattr` operations. By using `getattr` transactions alone, there is potential for `nfstrace` to confuse `getattr` transactions caused by such operations as getting a directory listing with those transactions being used to validate the contents of the client cache.

In comparison, the graph of Figure 5, a normalised cumulative distribution of the number of files of each size, shows that the estimation calculated by

	interval length	`snooper`	`rpcspy`
Number of active users			
Maximum	10 minute	4	4
Average		1.6	2.2
Maximum	10 second	3	1
Average		1.0	1.0
Total bytes transferred per active user			
Maximum	10 minute	6,342,850	5,477,752
Average		263,535	109,820
Maximum	10 second	5,120,000	5,048,320
Average		11,422	18,404

Table 5: The maximum and average number of active users over given intervals and the total quantity of data transferred per active user in those intervals.

File system	`snooper`	`nfstrace`
`/`	43,378	227,880
`/usr`	437,123	287,006
`/var/spool/mail`	267,887	201,417
`/usr/local`	10,226	12,310
`/usr2`	42,713	46,067
`/packages`	1,316,180	440,371

Table 6: A comparison of the average size for files accessed on each particular file system.

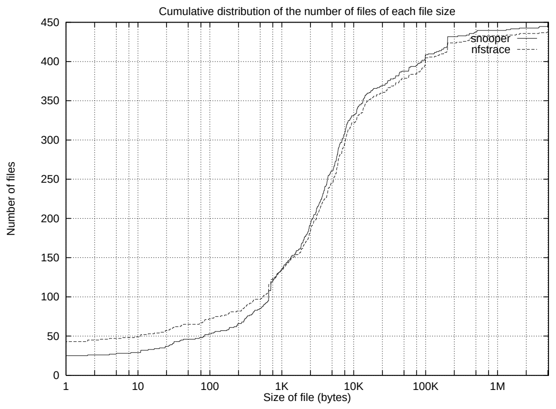


Figure 5: Cumulative distribution of number of different files accessed versus file size. From this graph we can deduce the number of times different files less than a given size have been accessed. For example both techniques suggest that over 150 of the files accessed are 1 kbytes in size or smaller. *Note:* the file size axis is logarithmic.

`nfstrace` compares well with the results of `snooper`. The two significant differences between the results of `nfstrace` and those of `snooper` which lead to disparities in the graph are for the number of zero-length files and the number of files which were approximately 700 bytes in length.

In the first case, `nfstrace` is not able to generate accurate estimations of accesses to various zero-length files and creates records of many more accesses than actually happened. This may most likely be due to `nfstrace` being unable to differentiate between `getattr` transactions for directories and those resulting from the opening of a zero-length file. In the second, related case, `nfstrace` has underestimated the number of accesses to various files which were approximately 700 bytes in length. In addition to the reasons above, it is possible that `nfstrace` evaluates many of the 700-byte file accesses as being zero-byte files accesses because of the block cache absorbing the small-file transactions.

Files with a short life-span can also present a problem to `nfstrace`. This is because given a short enough life-span between file creation, the writing and reading of data, and file deletion, no NFS `read` or `write` transactions may occur during the open-close session. As a result `nfstrace` is not easily able to record data transfer operations on files with a short life-span.

The following table, 7, gives a breakdown of the number of different files accessed by the system during the measurement period. These values are consistent with the hypothesis that `nfstrace` was unable to evaluate correctly the number of accesses to zero length files. The average file size for / would strongly confirm this, although the /packages results run counter to this. This strong counter-example could be due to the unusual nature of files on that particular file system. We note also that nfstrace results count one less file for that file system; a single large file would have modified this average considerably.

While there are notable differences in each of Tables 6 and 7, the results from them, in addition to those of Figure 5 show that nfstrace was able to give results broadly comparable with those of snooper.

5.6 File open-close sessions

The open-close session of a particular file is a concept around which a number of measurements are based. A number of studies have used such measurements; examples include file sharing, file utilisation and various cache studies [20, 2, 12, 13, 25].

Such open-close session measurements include the length of time a particular file is open, the amount of data accessed in that time, the amount of data *potentially* accessed (the size of the file opened), what sort of open-close session was involved, whether the file was opened for read and/or write operations, etc.

The number of open-close sessions as well as a breakdown of the relative types, are tabulated in Table 8. The implementation of NFS under Ultrix includes the synchronous writing of modified data blocks to the file system at the close of a file. This means that nfstrace can potentially miss write operations on files that ultimately leave the file with zero length, for example some sort of temporary file.

nfstrace will not be able to generate results for reads on files that occur in close succession (where the cache contents are still valid). Additionally, nfstrace may not correctly interpret getattr NFS transactions used to validate the cache. The result is nfstrace will either miss some open-close sessions altogether, incorrectly interpret NFS transactions as not being an open-close session, or incorrectly consider that the NFS transactions from two or more separate open-close session are from the same open-close session.

The larger number of writes recorded by nfstrace will certainly include the read-write operations snooper recorded. nfstrace is unable to detect read-write sessions and would consider each of such operations as a separate read and write session. Null open-close sessions, where no data are transferred and the file is simply closed, would not be able to be detected by nfstrace. Instead, nfstrace interprets any file open, if that were the only operation on a particular file, to be a reading of an unknown amount of data from the client cache.

Because the borders between read and write operations cannot be determined accurately, nfstrace will tend to collect successive open-close sessions together, interpreting them as one, longer, open-close session. As a result of this, the average duration of the open-close sessions reported by nfstrace may be higher than the durations reported by snooper.

Tables 9 and 10, record the open-close sessions on a type of open-close operation per file system basis, and by file system per operation. Firstly, Table 9 shows the full effect of the cache filtering, combined with nfstrace incorrectly interpreting information available, causing open-close sessions to be removed. This is especially the case for the / file system. The results for /var/spool/mail are a good example of where nfstrace has misinterpreted the NFS getattr transactions as open-close sessions because mail files are often checked for new mail resulting in getattr transactions. By way of comparison, a better result is given for the /packages file system. Files from this file system are unlikely to be able to be kept in cache for long periods. The result is that nfstrace is able to give a better result for open-close sessions because the NFS transactions for this file system were more complete.

Because the cache is removing the need for a large number of the read operations to result in NFS transactions, the read:write ratio is closer to unity for the results of nfstrace than the results of snooper. While this ratio is expected, even desirable, for the measurements of data transferred, these values are incorrect for open-close sessions resulting in higher figures for average data transferred per session and incorrect information about the characteristics of the sessions.

However, while the ratios of the various types of open-close sessions produced by nfstrace are not particularly close to those of recorded by snooper, adding the figures for null sessions to the read open-close sessions improves the comparison for all file systems except for /.

For Table 10, all write values are increased by

File System	snooper		rpcspy		nfstrace	
/	111	(24.89)	68	(17.13)	98	(22.37)
/usr	10	(02.24)	8	(02.02)	8	(01.83)
/var/spool/mail	3	(00.67)	3	(00.76)	3	(00.68)
/usr/local	49	(10.99)	46	(11.59)	48	(10.96)
/usr2	269	(60.31)	269	(67,76)	278	(63.47)
/packages	4	(00.90)	3	(00.76)	3	(00.68)
Total	446		397		438	

Table 7: A breakdown per file-system of the total number of different files accessed during the trace period. The values in parentheses are each count as a percentage of the total number of files.

	snooper		nfstrace	
read entries	7442	(88.07)	1749	(68.51)
write entries	557	(06.59)	804	(31.49)
read-write entries	35	(00.41)	-	-
null entries	416	(04.92)	-	-
Total	8450		2553	

Table 8: The count of open-close sessions each monitoring system interprets. Additionally, a breakdown of these open-close sessions into read-only, write-only, read-write and null open-close sessions is shown. A null session is where no data are read from or written to the file (although the file was opened). Values in parentheses are the percentage of the total number of files each type represents.

File System	session type	snooper		nfstrace	
/	read	6415	(90.33)	818	(63.21)
	write	354	(04.98)	476	(36.79)
	read-write	35	(00.49)	-	-
	null	298	(04.20)	-	-
/usr	read	123	(73.21)	61	(100.00)
	null	45	(26.79)	-	-
/var/spool/mail	read	18	(40.91)	75	(91.46)
	write	4	(09.09)	7	(08.54)
	null	22	(50.00)	-	-
/usr/local	read	146	(100.00)	100	(100.00)
/usr2	read	731	(74.52)	686	(68.12)
	write	199	(20.29)	321	(31.88)
	null	51	(05.20)	-	-
/packages	read	9	(100.00)	9	(100.00)

Table 9: A breakdown of the open-close sessions on each file system by type of open-close session. Values in parentheses are each type of operation as a percentage of the open-close sessions on that file system.

nfstrace, particularly in the case of /usr2. This error will partly be because nfstrace interprets the creation of any file and any subsequent writing to that file as two separate write events. Additionally, nfstrace can incorrectly interpret multiple writes to the same file as consecutive open-close sessions. Because nfstrace interprets an access to the first byte of a file as the start of a new open-close session, nfstrace can interpret multiple writes into the same location in a file as multiple open-close sessions on that file. As an example, this situation can arise with the vi editor. vi uses log files that check-point the edit operations as they occur on the file, so vi can be continually writing small changes to the log file. These collections of small writes will result in blocks being written to the server and if there are a number of writes made to the first block, the first block may be written to the server several times. Each time the first block is written nfstrace could potentially misinterpret the writing of data as separate open-close sessions on the log file. It is worth noting that the actual number of extra sessions is quite small and, in comparison with values for all open-close sessions, will be overwhelmed by the quantity of other open-close sessions (read sessions in particular). However, for open-close sessions writing to a file, these extra open-close sessions can be significant.

Some of these problems are a result of the algorithms used by nfstrace. While some assumptions have been made by nfstrace so as to produce an open-close session record, this particular situation may be resolved with a more sophisticated nfstrace algorithm.

The duration of an open-close session is important in determining the amount of time a particular file is in use. This, in turn, is important in calculating the amount of time files are shared between users and, in a distributed file system, between systems. Figure 7 shows that duration of open-close sessions recorded by rpcspy will be longer than those recorded by snooper. The longer open-close sessions that cause the differences in average durations are likely to be a result of transactions that are part of separate open-close sessions being interpreted as part of the same open-close session.

Additionally, the calculation of duration from NFS traffic means that lead and lag times (times in which the file is open but no operation occurs) will be different from the average length of the open-close session. These situations are represented graphically

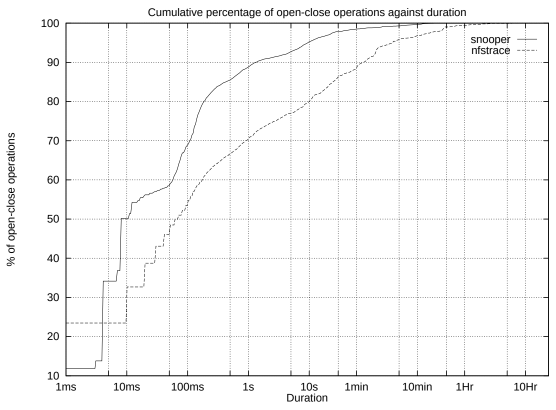


Figure 7: Normalised cumulative distribution of the number of open-close sessions versus the duration. From this graph we can deduce the longest of the open-close sessions for a given number of those sessions. For example, the snooper technique records that 70% of the sessions have a duration of about 100 milliseconds or less. *Note:* the duration axis is logarithmic.

in Figure 6. The figures show that the block operations upon which nfstrace's record will be based may not correspond with the logical open and close operations in an open-close session.

Figure 8 graphes a comparison of the data-transfer rate as measured by snooper, as per Figure 4, with the amount of data nfstrace estimates was potentially available to the system (the sum of the sizes of files accessed). While not directly comparable, it is worth noting that the sum of the sizes of files is able to give enough information to estimate with fair accuracy the trends of data transfer between client and server.

Figure 9 shows a cumulative distribution of open-close sessions versus the amount of data transferred. It is important to note that one reason that nfstrace differs so significantly with snooper is that nfstrace was unable to detect the large percentage of open-close sessions during which approximately 1 Kbyte was transferred. Additionally, snooper results estimate that fewer than 500 of the open-close sessions transferred one or zero bytes, whereas nfstrace results estimate those circumstances existed for more than 1,000 of the sessions it recorded.

A primary reason nfstrace does not record the large number of sessions transferring approximately 80, 750, 900 and 1,100 bytes is because those files

File System	session type	snooper		nfstrace	
read	/	6415	(86.20)	818	(46.77)
	/usr	123	(01.65)	61	(03.49)
	/var/spool/mail	18	(00.24)	75	(04.29)
	/usr/local	146	(01.96)	100	(05.72)
	/usr2	731	(09.82)	686	(39.22)
	/packages	9	(00.12)	9	(00.51)
write	/	354	(63.55)	476	(59.20)
	/var/spool/mail	4	(00.72)	7	(00.87)
	/usr2	199	(35.73)	321	(39.93)
read-write	/	35	(100.00)	-	-
null	/	298	(71.63)	-	-
	/usr	45	(10.82)	-	-
	/var/spool/mail	22	(05.29)	-	-
	/usr2	51	(12.26)	-	-

Table 10: A breakdown of the open-close sessions of each type, breakdown is by the file system of the file. Values in parentheses are each file system's operations as a percentage of the open-close sessions of that type.

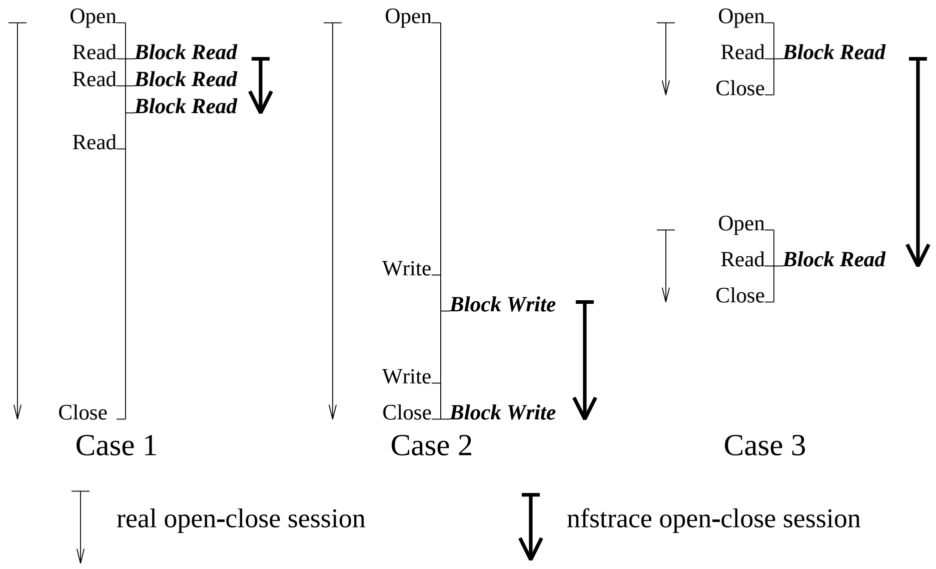
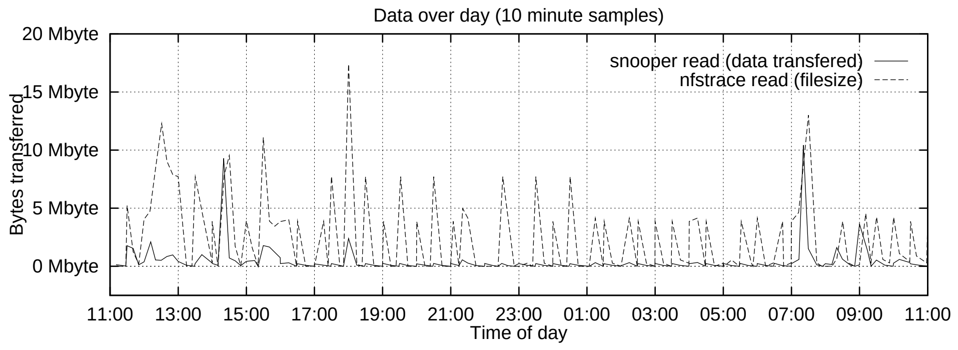


Figure 6: Several open-close sessions as generated by `nfstrace` are compared with the actual open-close session that occurred. The open-close session generated by `nfstrace` depends heavily on the type of NFS transaction each block access will invoke.



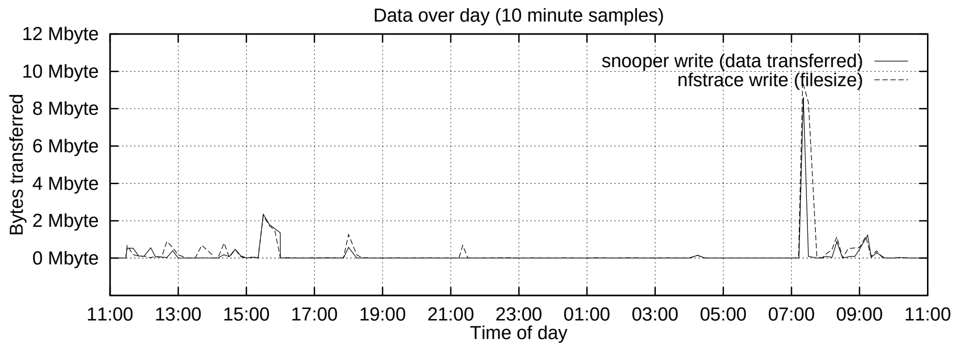


Figure 8: These graphs compare the transfer rate measured with `snooper`, to the total amount of data `nfstrace` has calculated the client has had access to in each file from which it has read data. As a low-order approximation, these values are comparable giving the same characteristics for data utilisation over time of the trace.

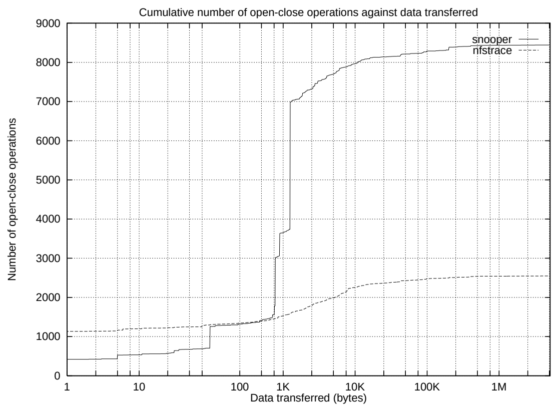


Figure 9: Cumulative distribution of the number of open-close sessions versus the data transferred for each open-close session. From this graph we can deduce the amount of data transferred per open-close session for a given number of those sessions. For example, the `snooper` technique records that over 7,000 sessions transfer about 1,100 bytes of data. *Note:* the data transferred axis is logarithmic.

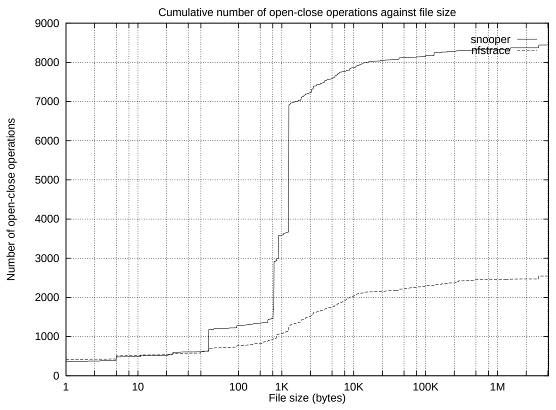


Figure 10: Cumulative distribution of the number of open-close sessions versus the size of the file accessed in each open-close session. From this graph we can deduce the maximum size of files opened for each open-close session for a given number of those sessions. For example, the `snooper` technique records that over 7,000 of the sessions access files containing less than 1,100 bytes of data. *Note:* the file size axis is logarithmic.

are in the cache and no data are transferred between server and client. This reasoning is strengthened by the fact that `nfstrace` gives trends similar to those of `snooper` for other transfer values (even if the actual number of sessions is greatly reduced).

The differences between `snooper` and `nfstrace` in Figure 10 have resulted from `nfstrace` being unable to interpret frequent accesses to files of a certain length, in particular, files which are 80, 750, 900 and 1,100 bytes in size. Accesses of such files account for a large percentage of the overall open-close sessions for regularly-accessed files but `nfstrace` is not recording an open-close session for them. This results in an exaggeration in the graphs for the number of open-close sessions for common data-transfer and file-size values. This situation is probably exacerbated by the inability of `nfstrace` to record many of the open-close sessions in which no data transfer is made.

5.7 Losses due to `rpcspy`

During this study, the recording of all Ethernet traffic by the `rpcspy` machine was not possible (a loss of 1.5% was recorded). This implies a loss of 0.6% of the total NFS transactions from the recorded trace, if we assume a ratio of NFS to non-NFS traffic at the same ratio as was recorded during the testing of `rpcspy` network packet capture mechanism. While a source of potential error, this data loss is overshadowed by the errors introduced by certain aspects of the operation of `nfstrace`. While this error should not be discounted, it can be considered to have low overall significance in the results.

6 Comparison Summary

The preceding results show that, while the two sets of results are not directly comparable, `nfstrace` is able to make a first order approximation of a number of values traditionally measured by systems such as `snooper` such as the total I/O transferred by a machine or the quantity of data written. Additionally, other estimated values, while estimated imprecisely by `nfstrace` in the current version, could potentially give accurate enough results to be able to replace systems such as `snooper` outright in a number of circumstances including measuring the number of active users per machine or the distribution of file size compared with files accessed. Most discrepancies in the interpretation by `nfstrace` when compared with results from `snooper` relate to the identification of open-close sessions. Minimisation of these errors would improve the estimation of both open-close session duration and data-size results.

A number of the results collected by nfstrace are not comparable with those collected by snooper, e.g. the amount of data transferred. While values for the maximum data transferred and write operations can be comparable, values affected by significant caching (e.g. reading of data, particularly small amounts repeatedly from the same file) will differ significantly.

In addition to measurements which can be compared, the unique nature of both snooper and rpcspy means that each has access to different types of information. Snooper is ideally suited to recording information about processes, an area from which network monitors are unable to retrieve information. On the other hand, nfstrace is ideally suited to collecting information about all machines on a particular network including, for example, all the traffic for a particular server. These differences mean each technique has a role to fulfil but there is certainly potential for network monitoring to be able to make measurements for which kernel instrumentation has traditionally been used in the past.

Additionally, it is worth pointing out that the information rpcspy generates and that nfstrace in turn uses, is not in error. The differences between nfstrace output and that of snooper occur because nfstrace attempts to estimate the operations on the user side of the cache from the operations that occur on the file system side of the cache. Improvements in the performance of nfstrace would come about from improvements in this estimation process.

7 rpcspy/nfstrace problems

For nfstrace to be a more useful tool, the accuracy of its estimations needs to be improved. There are a number of areas where nfstrace either makes errors or does not have enough information with which to work.

nfstrace problems to be addressed:

1. nfstrace treats the creation of a file as two separate open-close sessions.
2. Underestimation of the number of open-close sessions. This also means nfstrace can overestimate the data transferred per open-close session, particularly in the case of writes.
3. nfstrace is unable to observe logical data transfer.
4. nfstrace has no record of open-close sessions that transfer no data at the logical level.
5. nfstrace has no record of open-close sessions that both read and write data.
6. The nfstrace method used for summation of read operations and write operations can result in transferred data not being counted.
7. The method used for estimating the purpose of an NFS getattr transaction is simplistic.
8. nfstrace does not estimate the contents of a client cache. As a result nfstrace will assume files in cache are being accessed when this is not the case.
9. nfstrace is unable to detect short open-close sessions.

To a large extent these problems are also a result of NFS not making enough information available for nfstrace to be able to estimate the operations that are occurring. The lack of data supplied by NFS also means nfstrace acts as a filter removing short, consecutive, open-close read sessions. Such operations are absorbed by the cache and as a result fine-grain sporadic operations are missed.

8 Improving rpcspy and nfstrace

Improvements to rpcspy will be achieved by using a high-speed machine with a high-speed, low-loss network interface to be dedicated to the task of data collection. The improvements to nfstrace can not be stated quite as concisely. Smaller changes to nfstrace include:

- adding the ability to interpret other significant NFS transactions such as create,
- using a simple ratio multiplier to obtain an estimate of data transfers at the logical level,

- modification of `nfstrace` to keep information about file truncation giving the ability to interpret file re-write events
- separately recording data read from and written to the server for all open-close sessions,
- recording information on which blocks of a file have been accessed, and
- interpreting NFS `getattr` transactions that immediately follow a file being read or written as another open-close session.

While some of these changes, such as the last item listed, would need to be tested to ensure the resulting extra records were correct, others in the list would give immediate improvement in the abilities of `nfstrace`.

More significant changes to `nfstrace` include

- pre-loading information about programs that cause `stat` system calls such as `ls`,
- build a block cache simulator into `nfstrace`

In order to pre-load information about commonly-used programs that cause `stat` system calls, it may be necessary to profile the system prior to any significant tracing activity. In most systems, commonly-used programs such as `ls` could be expected to generate potential problems and could be added by default. However, the need to do a profiling operation would not only increase the complexity of passive network monitoring but might also negate any advantage of network monitoring by potentially requiring access to the machine being monitored. Another alternative, or addition, to pre-loaded configuration information is for `nfstrace` to characterise programs such as `ls` as it processes the NFS-transaction data. `nfstrace` would locate `ls` type programs by noting programs which, once executed, caused clusters of NFS `lookup` and `getattr` transactions, typically for files sharing the same subdirectory. In this way, `nfstrace` would be simultaneously processing the data and gaining enough information to locate programs causing extraneous NFS `getattr` transactions thus improving the prediction of `ls` type programs during the course of the run.

The incorporation of a block-cache simulator into `nfstrace` offers the best potential for increasing the accuracy of `nfstrace`. Unfortunately, several significant items of information would be needed to recreate accurately the block cache of a client. These would include the cache size on the client, the number of cache entries and the size of the data blocks being transferred between client and server. Additionally, the programming and testing of a cache simulator is not a simple task and because of resources used (memory, etc.) would potentially mean `nfstrace` could not be run simultaneously with `rpcspy` which is a preferred operating mode (in order to reduce output data).

The addition of the simulator would mean that `nfstrace` would be attempting to model a particular type of block cache. While there is a common ancestry for the method used by block caches in UNIX and its derivatives, there are notable differences. The introduction of such facilities as the demand-paging of executables as well as subtle changes in the cache system means the behaviour of the caches of systems being monitored will differ, sometimes dramatically. The result is that `nfstrace` may be required to incorporate models for several different block-cache systems. While this would add to the complexity of `nfstrace`, the common ancestry of block caches means much of the code used in each simulator would be common to all. It is conceivable that such an `nfstrace` could read a configuration file containing information on which cache method each client was using. Without appropriate configuration information, `nfstrace` could assume a particular model, perhaps the most common cache method used or the worst-case simulator model.

Such a pre-loaded configuration file would also contain information about NFS parameters such as cache and attribute timeouts, thereby increasing the accuracy of the simulator. This information, on a file-system by file-system basis, could also give information about the characteristics of access to a file system, e.g. mail file systems can potentially cause open-close sessions to be generated when none was, and so on.

A block-cache simulator would increase the accuracy of the open-close session predictions `nfstrace` makes and allow `nfstrace` to be used for other purposes. `nfstrace` has the potential to simultaneously simulate the caches of all the machines on a network so it could be used to study interactions between the caches of different machines. For example, such a facility would enable a comprehensive study of block sharing among NFS clients.

An extension to `nfstrace` would enable it to keep track of information about the directory systems in a distributed file system. Modifications to directory information are written synchronously back to the server as the modifications take place but the directory information itself is cached on the clients. Because changes are written synchronously, it is possible for `nfstrace` to maintain an accurate simulation of the state of the file system. Additionally, `nfstrace` could incorporate a directory-name cache simulator in the same style as a block-cache simulator and be able to simulate the contents of this cache among many clients. As in the case of a block-cache simulator, a directory-name cache simulator would enable `nfstrace` to be used to study interactions between the caches of the clients and track the history of changes to the file system. The use of such a modification may enable a follow-up study to Shirriff and Ousterhout's work on name and attribute caching ([26]).

Many of the limitations in `nfstrace`, indeed, the very need for `nfstrace` to have to *estimate* open-close sessions, are caused by the fact that this information about open or close is not transmitted in the NFS protocol. Other distributed-system protocols, such as Sprite [19] and the Andrew File System [12], transmit information related to the state of files in the distributed file system. If `nfstrace` was modified to work with such a state-orientated distributed system, the accuracy of `nfstrace` output could potentially be as high as a full kernel instrumentation trace. The potential for accurate `rpcspy/nfstrace` analysis of distributed systems should also hold true for any distributed file system that transmits enough state information across the network. This method even has the potential to work on theoretical distributed file systems, such as xFS [36, 9], which depart from a central file server model completely. It is conceivable that during the development of such monitoring systems, methods based on the passive monitoring of network traffic would become a primary tool for assisting in the development and ultimately the management of such systems.

Another technique for increasing the accuracy of `nfstrace` is to add *simulated* state operations to NFS. This would involve modifying the kernel of each client to output extra NFS transactions for system calls such as `open`, `close` and `seek`. It would not be necessary for the server to act on or even acknowledge these calls but the transmission of the extra information through the network would potentially give `nfstrace` enough information to be able to establish when files were opened and closed. Of course, such modifications are contrary to many of the concepts of passive network monitoring, requiring modifications to perhaps many client machines. However, this technique would maintain the benefit that the collection of the trace data would be independent of the server and clients. It would impose no extra workload directly upon them. This method of adding additional information to the communications traffic between client and server, for the purposes of monitoring, was used in Baker et al. [2] as one of a number of modifications they made to collect data for their work.

Distributed computer systems do not consist solely of distributed file systems. Systems in the style of Sprite [19] and Amoeba [30] enable the migration of processes among CPU elements (typically a CPU element is a computer workstation). A monitoring method for such a system might involve monitoring the network's interconnecting processing elements and tracking the movement of the processes in the same way that `nfstrace` monitors the movement of file data among workstations. In this way, passive network monitoring has possible applications in areas other than just the monitoring of distributed file systems.

9 Summary comments

System monitoring has a significant role in the development of computer systems.

A common method of monitoring systems is to use full kernel instrumentation, involving the modification of the source-code for the operating system of the machine. Passive network monitoring can be a preferred choice over kernel instrumentation for certain system monitoring work, particularly where the source-code of the operating system is not available. Additionally, other advantages of passive network monitoring include:

- an independence of the collection of results from the machines being monitored on the network,
- the ability to monitor simultaneously multiple machines on a network; the passive network

monitoring system requires no modifications to the operation of the monitored systems, and

- the collection of data with passive network monitoring does not impact on the machines being monitored.

Through the comparison of these two techniques, it is shown that passive network monitoring is satisfactory as a partial replacement for full kernel instrumentation.

In addition to this, passive network monitoring is non-invasive, platform-independent and has the ability to simultaneously monitor many network users. This gives it the potential for use in many systems studies using a broader cross-section of machines. Only through such a broad analysis can new systems be built based on information gained from more than just test systems and theories.

10 Future work

Ideally, future work would broaden the base over which the comparison of the two systems was made. The improvements could encompass both the inclusion of all traffic types, instead of the restriction to only non-executable file traffic, and the performing of the comparison on machines in a variety of operating circumstance. By comparing over a variety of systems, any peculiarities of the load the test system was placed under would be highlighted or, at least, minimised.

With the current system, a further study establishing the accuracy of the `rpcspy/nfstrace` system for the recording of block traffic communicated between client and server would prove useful.

Using a more accurate `nfstrace`, a comprehensive analysis in the style of Ousterhout et al. [20], Baker et al. [2], Howard et al. [12] and Spasojevic and Satyanarayanan [29] could be possible. Such an analysis would not only form an interesting comparison and contrast with those studies but also enable data to be collected from a variety of systems rather than the traditional limitation to academic or research installations.

A comparison of `nfstrace` with a similarly-designed RPC transaction processor analysing other distributed file systems based upon RPC communications would give an interesting point of comparison between NFS and those systems.

The incorporation of a cache simulator into `nfstrace` would offer the potential for an increase in the accuracy of `nfstrace` estimations and the possibility for `nfstrace` to be used to perform other system studies directly without the need for any extensive results processing. Such a study could cover performance issues, while another study could be made into the sharing of files and blocks among clients. In the case of a performance study, the cache simulator could be used to establish relationships between block lifetimes and cache effectiveness with the size of caches and timeout characteristics of the NFS system.

An extension of this work could take the form of a study which would also be possible with a suitably-enhanced `nfstrace` system into the utilisation of files and sub-directories, including lifetimes, usage distribution, etc. By combining such a modified `nfstrace` system with data about the file system before and after the trace period, it would be possible for `nfstrace` to accurately simulate and track operations on the directories of the file system. Such a facility would allow studies into file-naming structures and the caching of those structures in the style of Shirriff and Ousterhout [26].

Acknowledgements

The authors would like to thank Matt Blaze for making the original suite of `rpcspy/nfstrace` software available. The authors are equally grateful to Songian Zhou and Chris Siebenmann for the access to the original `snooper` code. Thanks are extended to Cameron Blackwood and Ralphe Neill for commenting on drafts of this paper. A. W. Moore was the recipient of a Monash University Postgraduate Writing-Up Award.

References

[1] Anderson, P. Effective Use of Local Workstation Disks in an NFS Network. In *USENIX LISA VI October 19-23, 1992* (October 1992), pp. 1–8.

[2] Baker, M. G., Hartman, J., Kupfer, M., Shirriff, K., and Ousterhout, J. Measure-

ments of a Distributed File System. In *Proceedings of the 13th Symposium on Operating System Principles* (Pacific Grove, CA, October 1991), ACM, pp. 198–212.

[3] Blaze, M. NFS Tracing by Passive Network Monitoring. In *USENIX Conference Proceedings, Winter 1992* (San Francisco, CA, January 1992), USENIX, pp. 333–344. Also available as a Technical Report with the Department of Computer Science, Princeton University.

[4] Blaze, M. *Caching in Large-Scale distributed file systems.* PhD thesis, Princeton University, January 1993.

[5] Blaze, M., and Alonso, R. Issues in Massive-Scale Distributed File Systems. In *USENIX File System Workshop, May 21-22, 1992* (Ann Arbor, MI, 1992), pp. 135–136.

[6] Boggs, D. R., Mogul, J. C., and Kent, C. A. Measured Capacity of an Ethernet: Myths and Reality. Tech. Rep. 88/4, Digital Western Research Laboratory, April 1988.

[7] Carson, S., and Setia, S. Analysis of the periodic update write policy for disk cache. *IEEE Transactions on Software Engineering 18*, 1 (January 1992), 44–54.

[8] Carson, S., and Setia, S. Optimal Write Batch Size in Log-Structured File Systems. In *USENIX File System Workshop, May 21-22, 1992* (Ann Arbor, MI, 1992), pp. 79–92.

[9] Dahlin, M. D., Mather, C. J., Wang, R. Y., Anderson, T. E., and Patterson, D. A. A quantitative analysis of cache policies for scalable network file systems. Tech. Rep. UCB:CSD-94-798, Department of Computer Science, University of California, Berkeley, February 1994. Also appeared in 1994 ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems, Nashville, TN, May, 1994, pp 150-160.

[10] Floyd, R. A., and Ellis, C. S. Directory reference patterns in hierarchical file systems. *IEEE Transactions on Knowledge and Data Engineering 1*, 2 (June 1989), 238–247.

[11] Hartman, J., and Ousterhout, J. Zebra: A Striped Network File System. In *USENIX Workshop on File Systems, May 1992* (May 1992), pp. 43–52.

[12] Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and West, M. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems 6*, 1 (February 1988), 51–81.

[13] Kistler, J., and Satyanarayanan, M. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems 10*, 1 (February 1992), 3–25.

[14] Lyon, B. XDR : External Data Representation Standard, June 1987. Network Working Group Request For Comment (RFC) : 1014, Written in association with DARPA and Sun Microsystems Inc.

[15] Lyon, B. RPC : Remote Procedure Call Protocol Specification, April 1988. Network Working Group Request For Comment (RFC) : 1057, Written in association with DARPA and Sun Microsystems Inc.

[16] Mogul, J. C., Rashid, R. F., and Accetta, M. J. The packet-filter: An efficient mechanism for user-level network code. In *Proceedings of the 11th Symposium on Operating Systems Principles* (Austin TX, November 1987), ACM SIGOPS.

[17] Mummert, L., and Satyanarayanan, M. Long Term Distributed File Reference Tracing: Implementation and Experience. Tech. Rep. CMU-CS-94-213, School of Computer Science, Carnegie Mellon University, November 1994.

[18] Ousterhout, J. K. Why Aren't Operating Systems Getting Faster as Fast as Hardware. *USENIX Summer Conference June 11-15* (June 1990).

[19] Ousterhout, J. K., Cherenson, A. R., Douglis, F., Nelson, M., and Welch, B. The Sprite network operating system. *IEEE Computer 21*, 2 (February 1988), 23–36.

[20] Ousterhout, J. K., DaCosta, H., Harrison, D., Kunze, J., Kupfer, M., and Thompson, J. A trace-driven analysis of the UNIX 4.2 BSD file system. In *10th Symposium*

on Operating System Principles (Orcas Island, WA, December 1985), ACM, pp. 15–24.

[21] Reddy, A. L. N., and Banerjee, P. An Evaluation of Multiple-Disk I/O Systems. *IEEE Transactions on Computers 38*, 12 (December 1989), 1680–1690.

[22] Ritchie, D. M., and Thompson, K. The UNIX time-sharing system. *Communications of the ACM 17*, 7 (July 1974), 365–375.

[23] Ruemmler, C., and Wilkes, J. UNIX disk access patterns. Tech. Rep. HPL-92-152, Hewlett Packard Laboratories, December 1992. Also published in the USENIX Winter '93 Technical Conference Proceedings, San Diego, CA, Jan 25-29, 1993 pp 405-420.

[24] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B. Design and Implementation of the Sun Network Filesystem. In *USENIX Conference Proceedings, Summer 1985* (Portland, OR, June 1985), USENIX, pp. 119–130.

[25] Satyanarayanan, M. The Influence of Scale on Distributed File System Design. *IEEE Transactions on Software Engineering 18*, 1 (January 1992), 1–8.

[26] Shirriff, K., and Ousterhout, J. A Trace-Driven Analysis of Name and Attribute Caching in a Distributed System. In *USENIX Conference Proceedings, Winter 1992* (San Francisco, CA, 1992), USENIX, pp. 315–332.

[27] Siebenmann, C., and Zhou, S. *Snooper Users Guide.* University of Toronto, August 1993.

[28] Smith, A. J. Disk cache - miss ratio analysis and design considerations. *ACM Transactions on Computer Systems 3*, 3 (August 1985), 161–203.

[29] Spasojevic, M., and Satyanarayanan, M. A usage profile and evaluation of a wide-area distributed file system. Tech. Rep. CMU-CS-93-207, School of Computer Science, Carnegie Mellon University, October 1993. Also appeared in Winter USENIX Conference, San Francisco, CA, January, 1994.

[30] Tanenbaum A., et al. Experiences with the Amoeba Distributed Operating System. *Communications of the ACM 33*, 12 (1990).

[31] Thompson, J. File Deletion in the UNIX System: Its Impact of [sic] File System Design and Analysis, April 1985. Computer Science Division,EECS,University of California, Berkeley CS 266 term project.

[32] Thompson, J., and Smith, A. Efficient (stack) algorithms for analysis of write-back and sector memories. *ACM Transactions on Computer Systems 7*, 1 (February 1989), 78–117.

[33] Thompson, J. G. *Efficient Analysis of Caching Systems.* PhD thesis, EECS, University of California, Berkeley, September 1987. Also available as UCB/EECS technical report CSD-87-374.

[34] Wang, R. Y., and Anderson, T. E. xFS: A Wide Area Mass Storage File System. In *Fourth Workshop on Workstation Operating Systems* (October 1993), pp. 71–78.

[35] Zhou, S., DaCosta, H., and Smith, A. J. A File System Tracing Package for Berkeley UNIX. *Proceedings 1984 USENIX Summer Conference Portland Oregon June 12-14,* (June 1985), 407–419.

[36] xFS : Serverless Network File Service, July 18th, 1995. Available via the World Wide Web `http://now.cs.berkeley.edu/Xfs/xfs.html`.