NFS and Diskless Workstations:

a guide to tuning for performance

by

Paul Reilly
Technical Marketing, SSD
Silicon Graphics Computer Systems

January, 1991

SiliconGraphics
Computer Systems

Table of Contents

List of figures

List of Tables

Section 1.0 — Overview.

This paper explores the present-day performance issues surrounding Network File System (NFS) as well as diskless workstations, which depend on NFS for disk I/O. We'll begin with a brief look at the natural limitations of ethernet and then examine several of the NFS performance enhancements in IRIX 3.3. Next, we'll delve into how these enhancements affect diskless workstation performance, and how one can use local disks to improve the overall effectiveness of "diskless" workstations. Finally, we'll derive some configuration guide lines for building networks of diskless workstations and file servers.

The second half of this paper is a practical "how-to" guide for sysgening a diskless workstation, including how to add local disks to a "diskless" workstation and other useful information.

Section 2.0 — Ethernet Performance in the 1990's.

Before launching into the performance of NFS and diskless workstations, perhaps it would be best to take a moment to examine the issues affecting ethernet performance. As we shall see, a thorough understanding of how ethernet works in real-world environments is crucial to understanding the performance issues surrounding NFS and, hence, diskless workstations.

Nominally, ethernet is a 10-Megabit per second communications medium. This translates into a theoretical maximum of 1.25 Megabytes per second. However, that is just the raw speed of the medium. There are several factors which severely limit the actual transfer rate. These fall into two categories:

- First, the computational overhead induced by the higher-level protocols such as TCP, NFS, UDP, and IP.
- Second, how ethernet itself works in real-world environments.

Section 2.1 — Overhead Caused by Higher Level Protocols.

The first major factor limiting the effective transfer rate of ethernet is the computational overhead caused by IP, TCP, UDP, NFS, and other protocols. Until recently, the limiting factor in data transmission over ethernet was not the speed of the ethernet but the speed of the CPU. This was due to the following:

- First, each of the IP, TCP, and UDP layers of protocol have a CRC or Cyclic-Redundancy-Check. While relatively simple to calculate, CRCs are very repetitive and therefore time consuming.
- Second, the data packet might be copied memory-to-memory two or three times. While this is now more a historical problem today, some of the older implementations of these protocols still copy data between user and system memory. Fortunately IRIX does not suffer from this deficiency, but other vendor's implementations of these protocols might.
- Third, several timers are needed to keep track of what outstanding data requests are still unfulfilled; the administration of these adds substantially to the overhead.
- Finally, last but not least, several data transformations may be required to convert data from one manufacturer's computer into a format suitable for the local host.

All in all, your workstation's CPU is quite busy when it's pumping packets over the ethernet. So much so that the original implementations of ethernet were lucky to achieve 100,000 bytes per second of throughput.

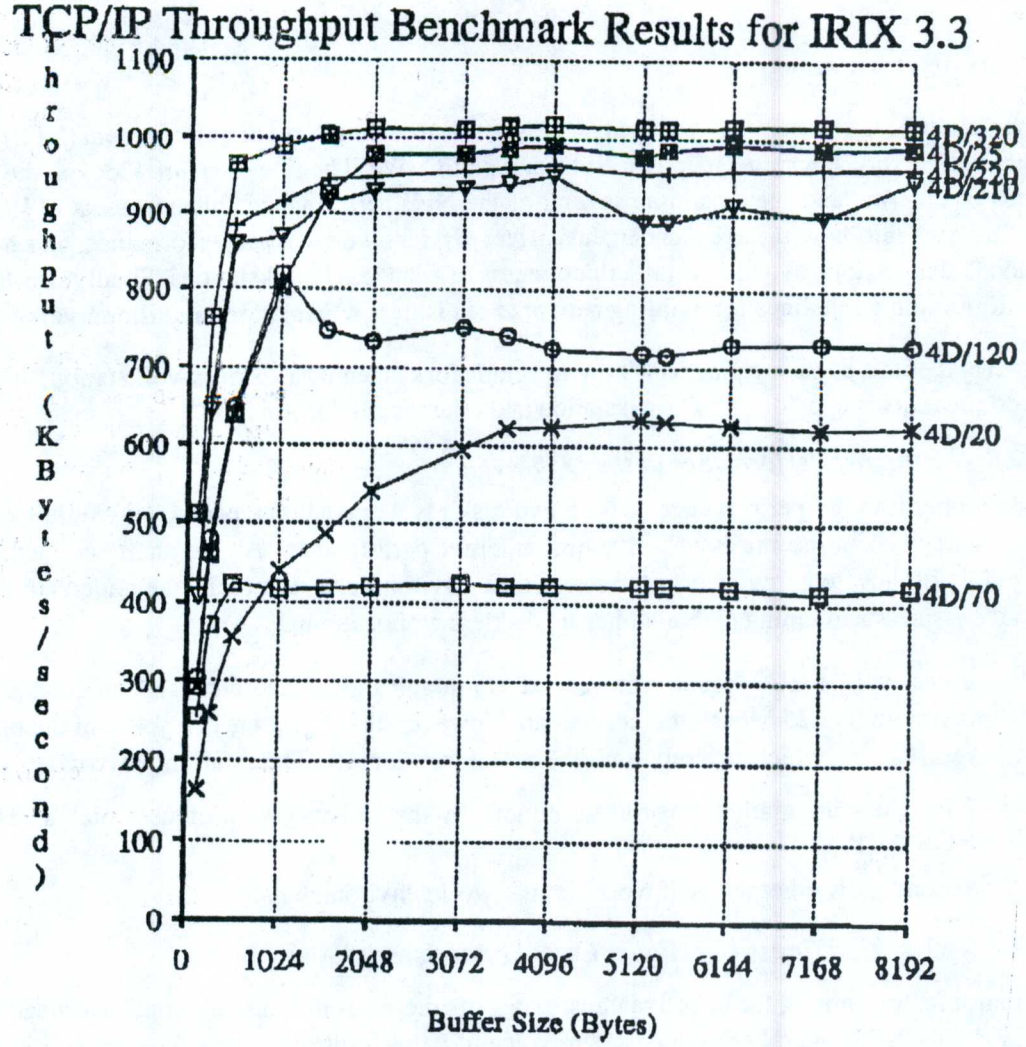


Figure 1:

IRIX 3.3 TCP/IP performance as functions of IRIS processor type and buffer size.

The natural outcome of this state of affairs was the Great Ethernet Benchmark Wars of the 1980's. All of the manufacturers proudly proclaimed that their ethernet was faster than anybody else's, and, to prove it, they waved charts and graphs full of data. And, in fact, tremendous increases in performance were achieved over the years by ever more clever programming; and they are still are, as we shall see a little later.

However, with the advent of high speed RISC computers such as Silicon Graphics'(SGI), the CPU became fast enough so that the ethernet itself was the limiting factor. This can be clearly seen in Figure 1, which shows SGI's latest TCP/IP benchmark results. While NFS runs on UDP and not TCP, these data are nevertheless pertinent because it is generally held that TCP performance is usually inferior to UDP's. The data on figure 1 clearly show distinct performance differences between various SGI systems. It is important to note that the quality of software is important, for the competition's results were generally inferior even when they were using the same RISC microprocessor. However, since we are comparing SGI systems to one another, we are using the same software and therefore we can clearly see what effect the speed of the microprocessor had on ethernet performance. Let's first examine several of our older and slower systems. The following are the TCP/IP throughputs:

- 4D/70 .....(8 Mhz CPU)..... 425,000 bytes per sec.
- 4D/20 .....(12.5 Mhz CPU).....625,000 bytes per sec.
- 4D/120 .....(16.7 Mhz CPU)..... 730,000 bytes per sec.

Quite obviously, the faster the CPU chip, the better the TCP/IP performance. If you were to fit these data to a linear regression model and extrapolate it for 20-, 25- and 33-Mhz CPUs, you would expect the following:

- 4D/25 .....(20 Mhz CPU)..... 1,000,000 bytes per sec.
- 4D/210.....(25 Mhz CPU)..... 1,125,000 bytes per sec.
- 4D/320 .....(33 Mhz CPU)..... 1,350,000 bytes per sec.

On the other hand, the same graph shows virtually no differences between the latter three systems; they are all asymptotically peaking at about one Megabyte per second. The reason for this sudden bunching is that the CPU speed is no longer the limiting factor—now it's the ethernet. Once we have at least a 20-Mhz CPU available, there is sufficient computational power to handle the protocol overhead faster than the ethernet can handle the packets, even under the most ideal of conditions. Clearly, a CPU chip speed beyond, say, 20-Mhz, does not increase TCP/IP performance because we have reached the practical performance limit of ethernet—about 80% of the 1.25 Megabyte the theoretical maximum. In other words, we've gone as fast as we can go, in benchmarks at least.

Section 2.2 — How Ethernet Works in the Real World.

The next question is whether these benchmark numbers really mean anything. The answer is yes, but not for measuring how much data you can pump over an ethernet in real-world conditions. However, since we now know roughly how many Kilobytes of TCP/IP data each type processor is capable of handling, we now have some idea of how many ethernets can be attached to each type server. We'll see this relationship in a few minutes, but we must first take a hard look at the benchmark results and what they mean in real-world conditions.

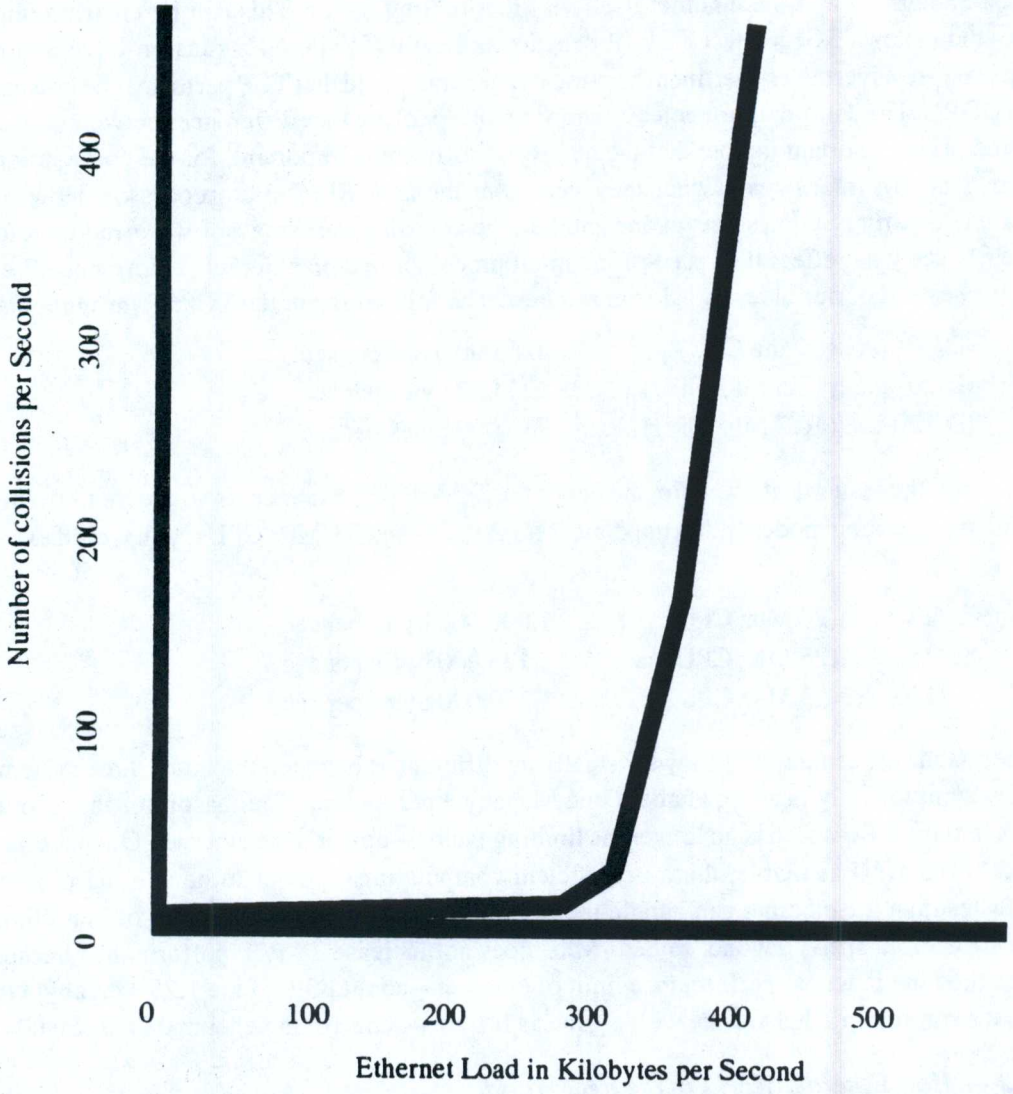


Figure 2

The number of ethernet collisions as a function of ethernet load.

When studying ethernet benchmark results, we must keep in mind that a benchmark is an artificial situation. The TCP/IP numbers presented in Figure 1 are analogous to building a racing car, taking it to the race track, and seeing just how fast we can go. Just imagine what would happen if you tried putting that same racing car on route 101—the main commuter road for Silicon Valley—during rush hour. Even a racing car would be lucky to go 55 MPH in such traffic. That's the real-world reality when it comes to driving racing cars in traffic. And the same is true of ethernet performance—don't expect any real-world ethernet to run at anything close to a million bytes per second.

We're going to focus our attention on real-world ethernet performance next. But to do so, we must first spend a few moments reviewing ethernet, and how it works.

Technically, ethernet is a CSMA/CD, or Carrier Sense Multiple Access with Collision Detection protocol. What that means, in simple English, is that each station on the ethernet listens to the carrier frequency on the ethernet cable, and when the station determines that nobody else is using the cable, it is free to broadcast its message onto the cable. Normally, this works well and everybody is able to get their work done. However, there is no control over who may speak when beyond the simple rule that the cable must be quiescent when a station begins to transmit its message. As you know, electricity doesn't travel instantaneously. Thus, it is possible for two stations even a few meters apart to begin transmitting simultaneously and thereby garble each others messages. For this reason, each station listens to what is on the ethernet cable while it is transmitting and continuously compares what it hears on the cable with what it is trying to transmit. When the two don't match, the station knows that its transmissions were garbled by another station's activity. Such garbled messages are known as "collisions," a very descriptive nomenclature.

Whenever a station transmitting on the ethernet detects a collision, it immediately ceases transmitting and waits a random period of time before attempting a retransmission. Eventually, the message will get through, but if there is a lot of activity on the ethernet, it might require several attempts. The end result is lost time and hence lost effective bandwidth. Several factors affect the occurrence of these collisions, the two most important being:

- First, the number of stations on the cable.
- Second, how many packets per second each wishes to send.

The reason for this is self-evident: The more activity you have on the network, the more likely a collision becomes—just as if you were driving on the freeway during rush hour. Increased traffic begets increased collision rates.

A great deal of research has been conducted on this topic, and the bottom line is that as the amount of traffic increases, the rate of collisions increases disproportionately until the network becomes saturated with collisions as shown in figure 2. The effect is quite non-linear. In fact, there is a distinct "hockey-stick" break in the curve when the number of collisions are plotted as a function of load. The break in this function is usually somewhere between 300,000 and 400,000 bytes per second, or between 25% and 33% of the maximal bandwidth of ethernet. Depending on the exact details of the situation, the number of collisions will suddenly increase exponentially until virtually every packet is colliding with another and none are getting through to the intended recipient. If you happen to have ethernet transceivers equipped with LEDs, the one for collisions (often marked "CP") will be solidly

lit. Thus, in the real-world ethernet situation in which many stations are actively trying to use the ethernet, trying to push more than 300,000 to 400,000 bytes per second through the ethernet is self-defeating. You will only increase the rate of collisions to the point that no useful work is done.(1)

Thus in real world situations, such as those you are likely to see in office environments, the maximum sustained performance you can hope for is roughly 350,000 bytes per second. The limitation is not in the processor (if it's fast enough), but in the cable. There are, nevertheless, several techniques which allow you to maximize the utilization of ethernet, particularly when using NFS.

Section 3.0 — IRIX 3.3 NFS Performance.

Now that we have explored ethernet's inherent performance and discovered that it's limited by the capacity of the cable, let's look at ways of making the most of it. The obvious answer is do some very clever programming. In the case of NFS under IRIX 3.3, that is exactly what our software engineers have done. The major enhancements include delayed writes and private mounting. These are essentially enhanced buffering on the server and the ability to extensively cache writes on the client. There is also a third option called write sync, or wsync, which needs to be explained, mainly because it controls the use of delayed writes.

Section 3.1 — IRIX 3.3's Delayed Writing Option for NFS.

The delayed write option is the default condition and is very similar to the buffering performed on EFS disk files, except it works on NFS files. What it does is buffer write operations coming from the client to the server into the main memory of the server and immediately replies to the client that the write operation is complete. A secondary feature of the delayed write software is that it will use the buffer in the server as a read cache, searching it first for the data requested by a client's read operations. It's only if the requested data aren't in the buffer that an actual disk read on is made by the server. Another feature of IRIX 3.3's NFS server buffering is that it will automatically read an 8 Kilobyte block even if only one byte is requested and retain the remaining data in the server's buffer against the likelihood that more of that 8 Kilobyte block be requested in following read requests from that client.

There is an obvious downside to delayed writes and that is the potential that data written out by the client might be lost is the server were to fail. While this vulnerability normally lasts only a few seconds, even that might be too long for some applications in which such risk is unacceptable. For this reason, delayed write may be turned off by specifying "*-wsync*" in the `/etc/exports` file on the server. An example `/etc/exports` entry might look like the following:

```
/usr/local  -wsync,rw,anon=guest     #Turn off delayed writes.
```

While using the *-wsync* option in the `/etc/exports` file doesn't turn off the cache reading, it does force each write to the file to be strictly synchronous. That is to say, the data is written to disk and

---

(1) Since it is not the purpose of this paper to explore the reasons why ethernet collisions occur, we recommend that those of you interested in an in-depth review read chapter nine of ***Local Networks*** by William Stallings, Macmillian Publishing Company, New York, 1990.

the disk write completed before the server will report back to the client that the write request was completed. This, in turn, suspends the task on the client making the write request until the write is actually completed on the server. As you might suspect, this has a deleterious impact on NFS performance and so should not be used unless the security of the data is far more important than performance.

Section 3.2 — IRIX 3.3's Private Mount Option for NFS.

While delayed writes are a feature implemented on the server, private mounting is implemented on the client. Specifically, it permits the client to locally cache NFS I/O. The great advantage of this is that if the application on the client is doing frequent reads of data it may have just written, the reads can occur out of local memory at memory-to-memory speeds. Since this mechanism also buffers writes to the server, it permits the application to run even faster than delayed writes on the server since the actual transmission of the ethernet can be done asynchronously.

There are two downsides to this mechanism. The first is the same as delayed writes on the server: Critical data could be lost if either system were to fail. The second is that record locking is performed on the client which makes it mandatory that no more than one client tries to write into a file that is mounted as private. If more than one client were to do so, the results will be very unpredictable. It's for this reason that this option is called "private": Only one system can safely write into a file mounted as private; the file must be the "private property" of that one client.

This option was originally developed with the intent of it being used by diskless workstations to mount their root directory. In fact, you can use this very powerful tool with any NFS file as long as you respect the primary rule that only one client may write into such files. Please note that this does not prohibit several tasks on that one client from using those files. In fact, they can. This is because all the write-locking is being performed on the client. On the other hand, ***never mount a diskless workstation's swap file as private!*** You would be trying to buffer your swapping, which will only increase the amount of swapping.

The private option can be enabled by either declaring it when you manually mount a NFS file system, or by declaring it in the `/etc/fstab` of the client. Below is an example:

```
nfsserver:/usr/my_files   /usr/data  nfs  rw,private,bg  0 0
```

Section 3.3 — IRIX 3.3's NFS Performance.

Although it is not the purpose of this paper to report benchmark data, a close look at the results of a recent engineering study can help quantify the effects of delayed writes and private mounting of NFS files upon IRIX 3.3 NFS performance.

Since it was possible for a short running benchmarks to give misleading results, the testing was conducted by using BMXX, an in-house ***sustained I/O*** NFS test. BMXX uses two systems, the client and the server. Unlike the TCP/IP benchmark data given above, the BMXX benchmark causes a great deal of ethernet traffic in both directions over the ethernet and causes a normal number of collisions. In short, it does an excellent job of simulating a very heavily loaded real-world ethernet.

When started, BMXX spawns eight processes in the client; they proceed to open, write, read, and read/write files with varying block sizes. The end result is many millions of bytes being shoved through the ethernet in both directions. Although BMXX is normally measured in wall-clock execution time, we've calculated the aggregate NFS throughput as well and presented it in Table 1, below.

As can be seen in Table 1, the performance varied immensely under the various test conditions. The first test, which was a base line test of IRIX 3.2, took about 48 minutes to complete. The aggregate throughput was only about 135,000 bytes per second, which is a good indication that BMXX is much more a real-world test than other IRIX 3.2 benchmarks which gave appreciably better results. The difference is that BMXX was designed to saturate NFS for many minutes with bi-directional I/O; not to see how fast it can run under ideal conditions.

During these tests, we used *gr_osview* to measure the disk, CPU, and I/O activity and *sysmeter* to keep track of number of packets and number of collisions on the ethernet. The BMXX benchmark tended to demonstrate peaks and valleys of ethernet activity, along with a large increase in the number of collisions during the peaks of ethernet activity. This phenomenon was most pronounced when we ran BMXX on IRIX 3.2 and was much less apparent in the IRIX 3.3 tests. We believe that this cyclic activity was due to the server and client periodically overfilling their rather minimal disk I/O buffers. This, in turn, caused NFS I/O requests to time out, causing a flurry of retransmissions, thus increasing the number of collisions, which then caused a further increase in retransmissions. Eventually, the impasse would break as the server and client each finally caught up with their disk I/O and cleared out their disk buffers. Then the cycle began again. Collaborating evidence for these suppositions was the activity lights on the disk drives, which showed cyclic activity patterns while the CPU Wait bar of *gr_osview* showed long periods of 100% I/O wait time. As noted, this cyclic activity was most pronounced under IRIX 3.2 and became progressively less noticeable as the testing progressed. This was due, in large part, to the ever more extensive buffering each test provided over the previous one (that is, except test 5, which was a special test).

Table 1. Results of the BMXX Benchmark as tested on varying sized servers and with various software features enabled. The server was a 4D/240, while the client was a 16MB 4D/120 in test 1 through 5 and a 16MB 4D/25 in test 6 and 7. (Note: Test 3, 4, 5, 6, and 7 all had two disk drives on the server and the second drive was used for testing. Private mounting was used in only test 7.)

Test	Irix	Server Configuration	Bytes/Sec	Wall Clock Time
1	3.2	16-MB	135,000	48 minutes
2	3.3	16-MB wsync off	259,000	26 minutes
3	3.3	16-MB wsync off (second disk)	269,000	25 minutes
4	3.3	32-MB wsync off	345,000	18:50 minutes
5	3.3	32-MB wsync on (no delwrites)	89,000	73 minutes
6	3.3	64-MB wsync off	409,000	15:54 minutes
7	3.3	64-MB wsync off, private on	635,000	10:14 minutes

The second test of BMXX was essentially identical to the first, except the operating system was IRIX 3.3.0. The improvement was dramatic. During this test, cyclic behavior was far less extreme than in the first test. The reason for this was that the server was using some 8 Megabytes of its memory to buffer write requests as delayed writes (shown as "delwri" in the Real Memory Pages bar of *gr_osview*), effectively filling in some of the valleys with some of the peaks. Still, in test 2, it was obvious that we were running out of buffer space because there were many periods of time that the CPU was waiting 100% on I/O.

Our first remedy was to add a second disk on its own controller. This was the third test. Performance improved, but only slightly. The BMXX benchmark now completed in 25 instead of 26 minutes. The reason for this is clear in hindsight—no swapping nor paging were occurring, and except for the BMXX activity, there was virtually no other activity on the disk.

Finally, we added more main memory in the fourth test, doubling the server's memory to 32 Megabytes. The effect was dramatic. In test 4, the BMXX benchmark completed in under 19 minutes and was pumping over 345,000 bytes per second through the ethernet and NFS. Quite a feat. Remember, this number is not for a brief test, but for a sustained period of almost 19 minutes. One inevitable conclusion of this is that you need rather large buffers for network I/O, particularly on the servers.

Having proven to ourselves that delayed writes were extremely effective, we decided to see what would happen if we turned off the delayed write facility by adding "*-wsync*" to the `/etc/exports` file on the server. The results are shown in test 5, a nearly four-fold increase in execution time. The reason for this is simple, NFS was writing synchronously, proven by the fact that the CPU Wait bar of *gr_osview* went solid blue, indicating 100% wait on I/O for the entire 73 minutes. Please note that the fifth test, the wsync-on test, took nearly twice long as the first test in which we tested IRIX 3.2 performance. The reason IRIX 3.2 was faster than the fifth test is due to the fact that IRIX 3.2 was in actuality using a limited amount of buffering; while in the wsync-on test, all buffering was suppressed. This clearly points out the very high price to be paid if one insists on using wsync.

Having completed the wsync-on test, we returned to exploring the effects of increasing the size of the server's buffers. This time we doubled the server's main memory to 64 Megabytes. And the performance increased again, jumping to over 409,000 bytes per second. During this test, the server used as much as 24 Megabytes for delayed write buffering. In fact, this buffering was so extensive that it distorted the results. The real throughput on the ethernet was actually less than the 409,000 bytes per second reported because BMXX finished executing about a minute before the last data buffered in the delayed write buffer was actually written on the disk. The real NFS throughput was closer to 350,000 bytes per second, which matches the rule of thumb that you should only expect about 30% ethernet utilization in the real-world ethernet environments.

Having established that our NFS is limited by the ethernet and not the CPU speeds, we started to examine ways of eliminating unnecessary traffic over the ethernet. The rationale for this is simple. If you can eliminate part of the load, then everything appears to have been speeded up. One effective technique is to cache I/O on the clients, which is exactly what the new private mount feature of IRIX 3.3 NFS does.

In the seventh test, we mounted the NFS files on the client as private. Each of the eight benchmark sub-tasks had its own file space so this was a legal configuration in which to use private mounting. This time the BMXX benchmark went into hyperdrive, pumping out a phenomenal 635,000 bytes per second of NFS performance. Obviously, this is not what actually happened, for examination of the individual sub-test results show that reading after writes were being conducted at well over 4 million bytes per second. The reason for this was the very clever caching being performed on the client. NFS was now looking in the local caches first to determine whether the data it had been asked to read was there. If it found it, as it often did in the case of the BMXX benchmark, it simply did a memory-to- memory copy and forgot about sending a read request to the server.

In summary, two facts are evident from these tests. First, if you are using a real-world ethernet, the performance peaks out at about 350,000 bytes per second. There after, you merely increase the number of collisions if you try to push more through the cable. The second point is that you can do several clever things to improve apparent performance. One is to buffer your write requests in the server so that disk accessing isn't a limiting factor. This also gives the server the ability to copy reads out of the buffer instead of going to the disk in those cases where the same data is being repetitively accessed. Similarly, mounting files as private allows the client to increase its performance not only by caching its NFS writes so that it can read the same data back should it be need, but also by eliminating extraneous traffic on the ethernet, a two-fold gain in performance.

Section 3.4 — Conclusions and Recommendations Regarding IRIX 3.3's NFS

While it may seem to some that we somehow "cheated" on this series of benchmarks, we were merely high-lighting that the present-day RISC-based CPU have far more power than is necessary to drive an ethernet, even with such overhead-intensive protocols as NFS. In addition, with the dramatic decrease in RAM memory prices, it is now feasible to use extensive buffering and caching. This, in turn, permits user-apparent increases in performance because the software is now working smarter, not harder.

A second issue is that there is now more than enough CPU bandwidth to drive more than one ethernet controller on the server at full speed. Please recall that in section 2.2 we noted that the TCP/IP benchmark data in Figure 1 demonstrated that there was enough CPU bandwidth on 20- through 33-Mhz based systems to handle multiple ethernets on the same server. While we would be reticent to recommend using the 4D/25 as a gateway or dual-ethernet controller NFS server, the reason would be solely due to the limited amount of main memory that can be put on the 4D/25. The newer 4D/35, on the other hand, could easily handle two ethernet interfaces.

Table 2 summarizes our recommendations regarding NFS server configurations. Regarding main memory, our recommendation is that you have about 24 Megabytes of available main memory for each ethernet if you plan to run something as horrendous as BMXX on your clients. Assuming a lesser load, 16 Megabytes of available main memory per ethernet should suffice. Thus if you plan to use a 4D/320 as a NFS server with four separate ethernet interfaces on it, design the system's configuration to have 64 Megabytes of memory available for buffering.

Likewise, we have some suggestions regarding the memory configuration on the clients. The amount of free memory on the client during these tests was about 8 Megabytes, which seemed to be sufficient

for the caching of the private mounted files. In separate tests, we found that the delayed write and private mounted buffering would take up all of the free memory of the system during peak loads, but normally limited itself to only about a half of whatever was available. We also found that our NFS's delayed write was very polite in that it didn't force swapping or other activities what would adversely affect other processes running in the system. While it is good that NFS is so well behaved, it is nevertheless a two-edged sword. If you set up a NFS server, don't let a bunch of extraneous jobs be run at the same time and use up all of the available free memory; it will decrease the NFS performance. In other words, time-sharing and NFS file-serving don't mix if you permit the timesharing tasks to eat up all of the available main memory. You must leave some of the memory for NFS I/O buffering.

Table 2. Suggested number of ethernet controllers per type of IRIS server. These numbers assume that you have sized the disk appropriately, allowing approximately 50-100MB of disk for each share tree, 75-100MB per each client (this includes a 20MB swap file on the server and 50 to 90MB for the client's `/usr/people` files. Any space required for the applications such as a DBMS would have to be added to the size of the disk. Generally, you will need ***at least*** a 750MB disk on a server.

We also assume that the server is ***not*** running large background tasks and that the specified main memory is being used only for running IRIX 3.3 or for delayed write buffering. We are also assuming that you are putting a fair number of clients on each ethernet (i.e. according to table 3).

Please note that this table assumes 16MB of buffering per ethernet controller, and that the processor speed (i.e. 20-, 25-, or 33-MHz) is the limiting factor for the number of ethernet controllers each type system can handle. If you wish to add other activities to the larger systems, you can do so, but you must add additional processors so that the 4D/210 is a 4D/220 or even a 4D/240, and you must add sufficient main memory so that the delayed write buffering can still take place.

Memory	Model	Comments	Max. ethernets
16MB	4D/25	Not a recommended server configuration	1
32MB	4D/35		1
64MB	4D/35	You can only add 1 additional ethernet	2
32MB	4D/210		1
48MB	4D/210		2
64MB	4D/210	Suggest splitting I/O over 2 disks	3
32MB	4D/310		1
48MB	4D/310		2
64MB	4D/310	Suggest having 2 disk drives	3
80MB	4D/310	Should have 2 disk drives and controllers	4

Section 4.0 — IRIX 3.3 Diskless Workstations.

Nominally, a diskless workstation has no disk and so must load its operating system and other software over the ethernet, along with other files such as data and even the swap space. Over time, this definition has evolved somewhat until what are called "super-diskless" and "semi-diskless" configurations appeared; paradoxically, these have a local disk even though the basic concept of loading the software through the network remains intact.

But why would anyone want a computer system with no disk? There are several excellent reasons for considering diskless workstations. The three most common are as follows:

- Price—the customer is looking to save money.
- Security—security issues require that there be no writeable media on the system.
- System Administration—the customer desires to control software administration by centralizing it on the server systems so that updating and backing up can become easily accomplished by the administrator.

Diskless workstations are an attractive solution for these requirements and have become surprisingly popular in the last few years. This is not to say that all of the experiences with diskless workstations were necessarily happy. There have been some unfortunate instances where diskless workstations have failed to live up to expectations. The reasons for these failures are usually based in a lack of understanding of how they work and what their limitations are. In this section, we're going to explore how diskless workstations work, as well as what their strengths and weaknesses are.

Section 4.1 — The Relationship of Client and Server.

As noted above, "diskless" may be something of a misnomer, for a "diskless" workstation may very well have a local disk. "Remote booting" may be a better appellation, for diskless workstations do not load their system software from a local disk, but over the ethernet. However, for the sake of simplicity, we shall retain the popular nomenclature and continue to refer to them as "diskless." Diskless workstations, therefore, are systems which have their system software loaded over a network—whether or not they have a disk.

Although a diskless workstation may have no disk, it is nevertheless using disk files. Normally, the files are located on a server and accessed by NFS. This includes all of the normal unix files, especially the root directory, as well as the swap space. If asked how this might be implemented, your first thought may be to suggest exporting the server's root directory by listing it in the server's `/etc/exports` file and then letting the clients mount it with NFS and use the server's root directory as though it were their own root directories.

Basically, this is the way it works—in principle, at least. However, it's not as simple as all that. There is a great deal of opportunity for incompatibilities and conflicts to arise with this simplistic approach. For example, the unix kernels required by the server and client might be different. This is normally the case, since Personal IRISes (PIs) are often used as clients while larger machines such as the 4D/240 and 4D/320 are used as servers. They require distinctly different kernels.

A second class of issues are the conflicts that will occur over the use of the control files in `/etc`. A good example is `/etc/passwd`. How can several different users of the clients have different super-user

passwords for their individual system? Each user would undoubtedly want a private `/etc/passwd` so they wouldn't have to share super-user privileges on their workstation with other users.

Section 4.1.1 — The Share Tree.

These issues were resolved by a two-tier structure of files systems on the servers used to service the clients. The first layer, call the "share tree," is nothing more than a copy of the software distribution for a particular class of computer systems. One such class would be the PIs, also known as IP6 processors, while another would be for the 4D/120, known as IP5 processors; and yet another for the 4D/70s, which are known as IP4 processors. The point is that an IRIX unix kernel built for one type of processor will not necessarily run the others. In fact, in the above example, the respective kernels of the systems listed are incompatible with one another.

Therefore, if you are planning to have a heterogeneous collection of diskless clients, you will need a separate share tree for each type of processor. Figure 3 shows an example of a large server's file structure which includes several share trees, one each for the PI, the 4D/120 family (IP5), and the older 4D/70 family (IP4). As is typical for such servers, the share trees are grouped together on a second disk mounted as `/d`. Thus, `/d/diskless/PI` is the "root" directory for the IRIX 3.3 diskless software for the PIs, and `/d/diskless/PI/3_3_PI` is the actual share tree for this class of systems. Additionally, diskless software "roots" for the the IP4s and IP5s are shown as `/d/diskless/IP4` and `/d/diskless/IP5`. Since most of the diskless clients are PIs, we are going to limit our discussion to these, although you can easily implement share trees for any class of IRIS by using the same techniques.

As noted, the share tree is an copy of the IRIX distribution for that class of SGI system that you are using as clients. This means a complete copy of at least all the files normally stored in the root partition, especially the unix kernels themselves. If not cleverly done, one can use a great deal of space because the entire share tree for each type of system could be from 50- to 100-Megabytes each. You do not want to have applications, for example, in the share tree. It is far wiser to locate them in seperate directories and mount them as standard NFS files systems. However, for now, it is sufficient that you understand that the share tree for a single type or class of diskless workstation is basically a complete copy of a the root directory and a good deal of the `/usr` directory as well. This, therefore, uses a good deal of disk space. For example, in the sample installation procedure presented at the end of this paper, we used 56 Megabytes for the share tree.

Section 4.1.2 — The Client Tree.

Now that we have resolved the problem of having to support different unix kernels for the server and the various types of clients, we now have to deal with the multiple diskless clients of the same type who wish to use the server. As we noted above, there is bound to be a lot of squabbling if we insisted that they all use the same copy of the `/etc/passwd` file and so forth. The concept of the client tree was developed to handle this circumstance.

Basically, all of the files in the share tree are left untouched and yet another set of / (i.e. root) and `/usr` directories are built, one set for each client. However, this isn't as horrible as it sounds for all of the executable files except for `/unix` are merely links into the appropriate share tree, with only the data files such as `passwd, hosts, fstab` and the like, actually being duplicated. While a share tree may be very large, only something on the order of two Megabytes is needed for each client tree.

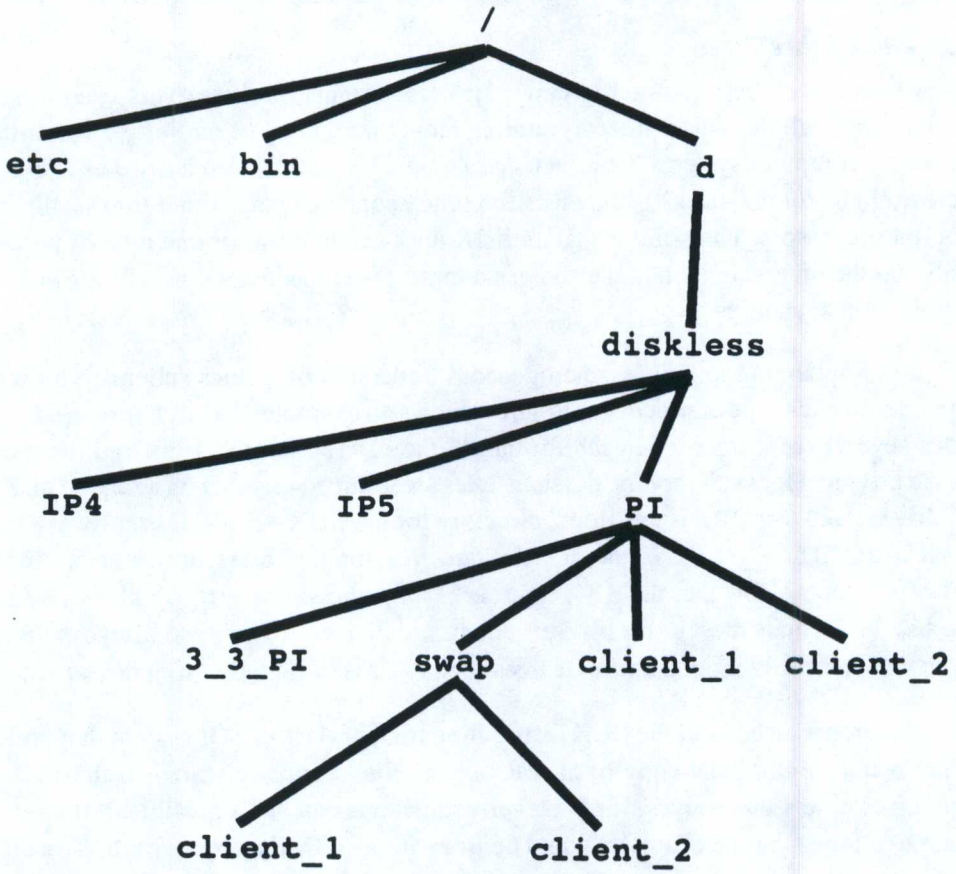


Figure 3

An example of a large NFS file server's disk layout showing the diskless workstation's share and client trees. The disk contains share and client trees for Personal IRISes (`PI`), 4D/120's (`IP5`) and 4D/70's (`IP4`). However, only the sub-directories for PI share and client trees are shown. The entry marked `swap` is a directory and the `client_1` and `client_2` entries in this directory are directories which actual swap files (`_swap`) for these two clients. The `client_1` and `client_2` entries under `PI` are the client tree directories for the two clients, while `3_3_PI` is the share tree directory .

Because each client tree has its own set of `passwd` and parameter files, it is the client tree which is actually used by a diskless workstation once it is booted. This way, the user can change the contents of his `passwd` file, as well as the other files in the system directories, completely free of fear that it will effect anybody else; and more important, that nobody will interfere with his files. He can even delete whatever system utilities he wishes, because all he is doing is removing a link from his client tree to the actual file in the share tree.

In the case of the example mentioned above, the client trees in `/d/diskless/PI` would begin in a subdirectory with that client's name. We have shown two clients in figure 3— client_1 and client_2. In the case of client_1, the server's `/d/diskless/PI/client_1` directory would be, as far as client_1 is concerned, its root directory.

A note about `/usr/people` and `/usr/tmp`: Do not put them in client tree. Mount any user directories as standard NFS files by listing them in the client's `/etc/fstab`, because it will make later software maintenance much, much easier. As a general rule, assume that you will have to delete a client tree at some time in order to perform an software upgrade. While this is actually only rarely the case, when it happens it is a royal pain if you have not followed this advice. In addition, if you have kept the system and user files separated in this manner, the user's files on one client can easily "moved" to another client by simply mounting them via NFS on the new client.

Section 4.1.3 — Swap Space.

Now that we've explored the share tree and client tree, we also need to briefly explore how swap space is implemented in diskless workstations. Normally, it is as a swap file mounted by NFS to the client. The swap spaces can be found in `/d/diskless/PI/swap/client_name`. These are directories, one to each client, each containing a single file in it called `_swap`. Each swap file is automatically generated when you build the client tree, and as we will see later, it can be deleted if you want to have a local swap disk.

Section 4.2 — How Diskless Workstations Boot.

Booting diskless systems is somewhat more complicated than simply pressing the reset button and watching the disk's activity light blink for a few seconds as the disk magically bestows life to your system. In the case of diskless workstations, you still press the reset button, but instead of watching the disk light blink, you now watch the ethernet transceiver's LEDs blink merrily for some seconds as it performs the mysterious rites of loading your system's software over the ethernet. First, the big picture.

There are four distinct phases to booting a diskless workstation:

- First, the client (i.e. your workstation) must locate the server that is to boot your workstation. You initiate this usually by powering-up your system or perhaps pressing the reset button. The code needed in your workstation to do this is a read-only memory known as a EPROM. In SGI systems, this is done with BOOTP which is in the EPROM.
- Second, the server passes to your workstation enough software so that your system can begin executing out of main memory. This includes the kernel, file system (at least NFS) and related software. TFTP is used to accomplish this. TFTP is also incorporated in the EPROM.

- Third, the kernel must now mount the file system on NFS and then proceed to load in the rest of the software you normally see while using your system. A remote procedure call, `bootparams`, is used to locate the those files on the server.
- Finally, once the remote file system is mounted with NFS, the system continues the normal booting procedure from your disk, the primary difference being that the root directory is mounted on NFS and therefore read over the network. A further minor difference is that the swap space is also a NFS file and not a partition on your disk drive; although you may, if you wish, retain a local swap disk.

At this point, your system is up and running just as though it were running from a disk. However, all disk I/O is now routed through NFS. (Unless, of course, you have a disk on your diskless workstation)

Section 4.3 — Details of Booting Diskless Workstations.

Now that we have seen the big picture, let's look at the details of booting a diskless workstation. The first problem that needs to be addressed when booting a diskless workstations is how does the workstation know who it is. Since the workstation has no disk, it can't read `/etc/sys_id` nor can it look itself up in the `/etc/hosts` file to learn its Internet address. Generally, the only information available to a diskless system before it is booted is its ethernet address. (We are cheating here a little bit, because SGI systems have something called a Non-Volatile Read-Only Memory (NVRAM) which can contain several dozen bytes of information, including your system's Internet address, but as we shall see, it's not really required. So, for discussion sake, let's pretend that your system doesn't know its Internet address.)

There are two protocols available to resolve this problem. The first is Reverse Address Resolution Protocol, or RARP, and the second is Bootstrap Protocol or BOOTP. Both protocols allow a server to accept a packet from the unbooted client containing only the client's ethernet address and return the client's Internet address and, depending on the circumstances, other information. Although SGI uses the BOOTP protocol instead of RARP, many competitors, including SUN, use the RARP. Therefore, we'll first look briefly at RARP and then spend the majority of our time on BOOTP.

Section 4.3.1 — Reverse Address Resolution Protocol (RARP)

RARP is a data-link level protocol which simply looks up the client's Internet address in an table and then broadcasts it on the network. The packet format is identical to that of Address Resolution Protocol or ARP, but the utilization is reversed.

While simple, RARP has some limitations. The first being that it is essentially a broadcast. The client issues an ethernet level packet containing its ethernet address and that of the "target" system which is itself. It simply broadcasts the packet on the ethernet and waits for a reply. It may get several. Any and all RARP servers connected to same ethernet cable as the client may respond by issuing an ethernet packet containing what it thinks is the client's Internet address. Hopeful, they all agree or at least the client is lucky enough to pick a valid response. There is no selectivity, no way to limit the response to selected servers, and since RARP never reaches the IP level, there is no way for an RARP packet to cross an IP gateway (or, as some call them more correctly, IP router.) Nevertheless, although some consider RARP to be limited, and while it may be rough and ready; it does work and usually quite well.

Section 4.3.2 — Bootstrap Protocol (BOOTP)

SGI chose a newer and more robust protocol, BOOTP, when implementing diskless booting. This protocol is a UDP level protocol which permits far more information to be exchanged in just two packets. The client issues a packet which must contain, at the minimum, its ethernet address, and may optionally contain its Internet address as well as the name of a target server. A fourth data item the BOOTP packet may contain is the bootfile name. As in the case of the workstation's Internet address and target system name, the bootfile name is optional. This permits the BOOTP protocol to be used in many different manners. In the simplest case, the client can simply broadcast its ethernet address and hope that some BOOTP server will respond just as in the case of RARP. Or it can focus its request to a server by name even though doesn't know the server's Internet address. The advantage of this option is that only one server will respond—even if it isn't on the same network as the client. This is because BOOTP packets are UDP packets and therefore can be sent through an IP gateway (or IP router) to distant servers.

Let's look at how you can use BOOTP packets in detail, starting with the simplest version; that is to say that the client merely knows its ethernet address. When the BOOTP server sees the request packet, it first looks in its `/usr/etc/bootptab` and checks whether the client's ethernet address is listed. If it is, the server will respond with a packet giving the client its Internet address, the server's name and Internet address, the root path of the bootfile for the client, and finally, the default bootfile name.

Should the client's ethernet address not be in `/usr/etc/bootptab`, the server then checks its `/etc/ethers` and `/etc/hosts` to see if it can at least determine the client's Internet address. If the server can determine the client's Internet address from these two files, it will return it to the client in the same packet as described above, but with the information on the bootfile nulled out.

This means a great deal of information can be returned to the client with the first response. Typically, it is enough to get the booting process under way immediately.

In addition, any IP routers that understand BOOTP and see the client's request packet might respond as well, but only if they know the named server. This permits the client to make contact with its named server even it they aren't on the same network. It should be noted that not all IP routers undestand how to forward BOOTP packets. However, since IRIX 3.3, all SGI routers do this correctly.

BOOTP also has a "user-friendly" mode in which the client already knows its Internet address, server's name and usually even the exact bootfile name. The client still broadcasts the packet because it doesn't know if the server is active or not. Therefore, in this mode, the client is essentially just polling to see if the server is there. Normally, only the named server will respond. And, in this mode the server will check the information supplied by the client, fill in missing details such as the server's Internet address. Furthermore, if the bootfile was named by the client, the server will also check for the existence of that file, and if it's non-existent, null the bootfile name field out so that the client knows that there was an error.

While all this verification is nice, the reason why this mode of operation is "user-friendly" is because the server's administrator doesn't have to know the client's ethernet address. The big headache with RARP as well as the simpler modes of BOOTP is that somebody has to go around and collect the ethernet hardware addresses for all the clients and then type them into ether `/etc/ethers` or

`/usr/etc/bootptab`. In the "user-friendly" mode in which the client already knows its Internet address, the client broadcasts not only its ethernet address but also its Internet address for everbody on the ethernet to pick up. This is much, much easier for the poor system administrator who now has to merely assign an Internet address to a new diskless workstation and then show the user of that workstation how to enter it into the NVRAM (i.e. "PROM," as many users call it) of this system, assuming, of course, that the diskless workstation is an SGI system.

Section 4.4 — How SGI uses Bootstrap Protocol (BOOTP)

As noted in the previous paragraphs, SGI's approach to remote booting is to use the "user-friendly" mode of BOOTP in which the client already knows its Internet address as well as the server's name and even the correct name of the bootfile. This immediately raises the question how we can give each client the ability to know this information. Actually, this is easily done with modern electronics, especially with availability of NVRAM. Since booting a diskless workstation requires a considerable amount of code to be present in the EPROM, it wasn't very difficult to go a few steps further and give the user, or perhaps the network administrator, the ability to use the monitor mode (number 5 on the boot menu) of the EPROM code to modify the values of several parameters in the NVRAM. The two of most interest are "*netaddr*" and "*bootfile*." A third parameter, "*diskless*," must also be set to "*1*" in order for the IRIX kernel to do the correct things while booting.

The first parameter, *netaddr* is the Internet address. It comes from the factory set to `192.0.2.1`. The *setenv* monitor command can be used to set this to any valid Internet address. This is then stored in the NVRAM so that it is retained even if the system is turned off and unplugged. An example of setting netaddr would be as follows:

```
>>setenv netaddr 192.60.26.18
```

The *bootfile* NVRAM parameter normally points to a disk file known as *sash* or standalone shell, which is in fact a very limited version of the unix kernel. This file is read off the disk in "raw" format without the benefit of the unix file system. Upon starting execution, *sash* mounts the root partition, reads in the unix kernel and passes control to it. This second kernel is the multi-user version of unix we normally work with. It opens several files in `/etc` and uses them to complete the booting process.

When remote booting, the *bootfile* parameter must be changed to give the server's name and the correct path of the bootfile. This command might look something like this:

```
>>setenv bootfile bootp()server:/usr/etc/boot/client_1/unix
```

In this case, the booting procedure is quite different than from booting from the disk. As noted above, BOOTP is used merely to verify the existence of the server and the bootfile, and also determine if there are any intervening IP gateways .

Once all this has been accomplished, it is time to actually read the kernel in. Unlike the case where unix is booted from the disk, *sash* is not used. The client already knows where the actual unix file it needs is and so asks the server to open it and start sending it down the ethernet.

The actual transmission of the bootfile is via Trivial File Transfer Protocol (TFTP), which is a very small and simple protocol which will easily fit into the client's EPROM. And this has caused some problems because, until recently, TFTP has been highly insecure. TFTP, you see, will happily transmit to whomever asks any file in the system regardless who that requester is, as long as "world" read permissions are in

the file's mode bits. Naturally, this behavior was found wanting, particularly by security conscious system administrators.

However, this problem has been eliminated starting with IRIX 3.3. The server's *tftpd* daemon now has the "*-s*" or secure option enabled. In this default mode, *tftpd* will only read those files which are in the `/usr/etc/boot` directory. In addition, *tftpd* may also access any additional directories that may be listed in `/usr/etc/inetd.conf`. In IRIX, we have listed `/usr/local/boot`. Thus, unless the bootfile is under one of these two directories, *tftpd* can not read it. Therefore, since the introduction of IRIX 3.3, the bootfile for a specific client is always `/usr/etc/boot/client_name/unix`. While this file is simply a link into the unix in client's client tree, you must use it in order for the booting to work.

The next step in the booting procedure is for the client to request that the bootfile to be downloaded from the server to the client using TFTP. This is fairly straight forward, and with IRIX 3.3.0, it requires some about two thousand TFTP packets . Once this is complete, you'll see your diskless workstation's screen flash and the SGI copyright notice appear. At this point, you have a running unix kernel, but just barely. It still has no access to `/etc`, or for that matter, the root directory itself. Your neonatal unix system now needs to find out where those files are.

In the good old days when every system had a hard disk, all the kernel had to do was mount disk partition 0 of the boot disk to find the root directory, and with it, `/etc`. Now, in the case of the diskless workstation, things are far more complex. What the client does at this time is issue a remote procedure call to the server known as a "`bootparams.rpc`," which basically asks the server to look the client up in its `/etc/bootparams` file and pass back the contents listed for the client. For client_1, the entry would look something like the following:

```
client_1 root=server:/d/diskless/PI/client_1 \
share=server:/d/diskless/PI/3_3_PI \
swap=server:/d/diskless/PI/swap/client_1
```

This information is then used to remotely mount the client's root and swap space using NFS. From then on the rest of the booting process is fairly much the same as the normal booting procedure from disk, except the reads and writes are through the ethernet.

Section 4.5 — Variations on using Bootstrap Protocol (BOOTP)

While the following is not part of the SGI officially released diskless software, we should look at several of the possible variations on this procedure for educational purposes. Please note that we are not suggesting that you actually use these in a production system. In fact, we seriously doubt you would ever wish to use them because they are all much less "user-friendly" than the standard SGI procedure. However, in the name of understanding how it all really works, let's poke around.

First, let's examine why the booting procedure as currently implemented by SGI in IRIX 3.3 requires the client to know the correct path and name of the bootfile. The answer is that no entries are made into `/usr/etc/bootptab` file of the server during the construction of the share tree or client trees. However, it does mean that you can't edit this file and make things work a little differently. Let's first look at the `/usr/etc/bootptab` file.

The /usr/etc/bootptab file has two main sections:

- The first defines the root directory of a boot sub-tree. Typically, this is defaulted to /usr/local/boot .
- The second section of the /usr/etc/bootptab file contains the each client's individual hostname, ethernet address, hardware type, Internet address, and finally, the bootfile as under the /usr/local/boot directory.

As noted earlier, in the "user-friendly" mode, there is no need for this information, so it wasn't filled in by *clinst*. As also noted earlier, the advantage of this approach is that the system administrator doesn't have to go around from client system to client system and obtain the individual ethernet address, write them down on a piece of paper and then transcribe them into the /usr/etc/bootptab file. There is lots of room for error in this procedure, particularly when you consider the alternative of simply entering the Internet address and bootfile in the client's NVRAM.

But you can still do it the old fashion way, if you wish. Once you've build the client trees, you can go into the /usr/local/boot directory and build a link to the each client tree's root directory. Then edit /usr/etc/bootptab to add the host name, ethernet address, Internet address and bootfile name in the bottom of the file. For example, if you built a link from /d/diskless/PI/client_1 to /usr/local/boot/client_1, your entry in /usr/etc/bootptab would look something like the following:

```
# host          htype  haddr              iaddr          bootfile
#

IRIS            1 01:02:03:8a:8b:8c       192.0.2.1      unix
client_1        1 08:00:69:83:ab:0c       192.10.26.10   client_1/unix
```

With this information, the only things you now have to set up in the NVRAM of the client is the following:

```
setenv     diskless 1
unsetenv   netaddr
setenv     bootfile bootp()
```

The above causes the client to broadcast a message to all BOOTP servers; if any have the client's ethernet address listed either in the /usr/etc/bootptab or /etc/ethers and /etc/hosts files, an response will be returned. (By the way, you really must have the client listed in the server's /etc/hosts or in yellow pages as well. Even if you managed to get the BOOTP packets to exchange correctly, you'll never get NFS to work later in the booting cycle without the server having the /etc/hosts file set up.)

Alternatively, you can also direct a the BOOTP packet to a specific server by using the following:

```
setenv     diskless 1
unsetenv   netaddr
setenv     bootfile bootp()server_name:
```

Assuming that you have /usr/etc/bootptab set up correctly, the named server will return a BOOTP packet with all the information to the client and, interesting enough, the client will then automatically set its NVRAM *netaddr* to the Internet address obtained in the return packet. Now, how's that for service!

Section 5.0 — Measuring Diskless Workstation Performance.

The most common question raised regarding diskless workstations is "How many workstations can I put on a server?" A somewhat facetious-sounding, but nevertheless completely accurate answer is as many as you want, as long as you don't care about performance. Obviously, you do care about performance; but then you ask again: " What constitutes good performance?" The basic fact remains that there are no simple rules for calculating how many diskless workstations you can put on an ethernet. You have far too many variables to deal with. What you must do is determine for your installation what constitutes an acceptable performance level and then tweak and tune your installation until everybody is happy, or at least nearly so. This is an on-going activity as well, because no network is static. New workstations are forever being added to the network and new applications are being added to the workstations.

However, there is still only one truly limiting factor when working with a diskless workstation environment. And that is, on average, you can only get about 350,000 bytes per second through a real-world ethernet. As noted earlier, this is the real bottleneck. The number of diskless workstations you can successfully run on a single ethernet depends on the following:

- What does successful means in your installation? This is really the same as asking how long will the user be willing to wait for something to happen. Generally, the less experience the user, the longer he will be willing to wait. The more experienced user will be much less patient. There is no simple way of determining this. However, as a rule-of-thumb, if the users get up and leave their workstations grumbling, you can be assured that you are not being very successful.
- What sort of NFS I/O load does each workstation impose upon the server? If the application is a simple "fill-in" the forms program, there may be a very small load. If the users are all doing software development, there will be a very heavy load.
- How much main memory is there on the workstations? This, surprisingly, is very important if there is a limited amount of main memory on the workstations.
- How many workstations are being used at the same time? That is to say, if you have 50 workstations connected to the ethernet, but only two are ever in use at the same time, you really have only two workstations to worry about, until, of course, the day comes when all 50 users decide to login at the same time.

Basically, what all this is driving at is that you have to measure the load, look at the response time, and decide for yourself what is "good." Therefore, our first order of business is to review what performance tools are available for doing this. These include the following:

- The ethernet transceiver. If it has LEDs that indicate what's happening, you can use it to judge the load on the ethernet.
- *NetVisualyzer*. This is a package of network monitoring tools offered by SGI. This includes *netgraph* which allows you to see the overall network load.
- *Gr_osview*. This graphical tool is included with the basic IRIX software and is extremely valuable in gauging the disk I/O and main memory requirements of an individual diskless workstation while it's running its applications.
- *Sysmeter*. Like *gr_osview*, *sysmeter* is included in the standard IRIX software. It permits you to see the total number of packets as well as collisions occurring at an individual workstation.

Section 5.1 — Using the Transceiver's LEDs.

Most ethernet transceivers have LEDs to indicate that it has power and that it is working. Typically, although it is not always the case, there are individual LEDs for power-on, Receive, Transmit, and Collisions. It is the last three that interest us. You can use them to estimate the load on you network and the approximate per cent load on the ethernet.

- Very lightly loaded (about 5%): Typically, the Receive LED blinks once or twice a second. The Collision LED blinks very rarely.
- Lightly loaded (about 10%): If the Receive LED is blinking at a fairly steady rate, you have a lightly loaded network. Usually the Collision LED blinks once or twice a minute.
- Moderately loaded (about 15%): The Receive LED is blinking steadily, but it is not on solidly. The Collision LED is blinking about five times a minute.
- Loaded (about 20%): The Receive LED occasionally blinks, but appears to be on solidly most of the time. The Collision LED is blinking ten to twenty times a minute. Occasionally, the Collision LED blinks rapidly.
- Heavily loaded (About 25%): The Receive LED appears to be on solidly. The Collision LED blinks at twenty to thirty times a minute and blinks rapidly for several seconds every minute or so.
- Overloaded (About 30%): The collision LED blinks merrily for long periods of time.

The important thing to remember is that you are interested in the *sustained* activity. If you see a flurry of activity, ignore it, even it it goes on for a minute or so. You might be just watching somebody boot up his diskless workstation. Another important bit of information is that the Transmit LED tells you a lot about the ethernet activity of individual workstations. If the transmit activity is high (i.e. blinking merrily) you have a potential problem. That station is eating up a good part of the ethernet bandwidth.

Section 5.2 — Using NetVisualyzer.

The *NetVisualyzer* tools are a far more accurate judge of overall ethernet activity than simply eyeballing the LEDs on the transceivers. This package includes *netgraph* which displays the on-going ethernet activity in strip-chart format. Every serious ethernet administrator should have *netgraph* running somewhere on the network, preferably on the server. You have many choices of what data you can look at with this tool. We recommend that you use look at the TOTAL, UDP, NFS, TCP, and ICMP packets in terms of per cent ethernet utilization. Generally, if you have a total utilization in excess of 25% for long periods of time, you must do something about it. The other measures will give you some idea where to look. For example, if there are more than one or two percent ICMP packets, there is a workstation doing excessive *ping*ing. This may be due to an uninformed user abusing this tool, or there may be something wrong with the workstation. You can use *analyzer* to capture ICMP packets and find out who it is.

There are several reasons why the TCP packets might be high. Usually, though its due to users using *rcp, ftp, rsh* and related software. If this continues for long periods of time, you may wish to investigate.

The UDP bar will tell you if several people are playing *dogfight.* While a lot of fun, *dogfight* will pump a heavy load of UDP packets into the ethernet and eat up an appreciable portion of the available 350,000 bytes per second. *Netlook* will quickly spot these miscreants because they are all connected to UDP port

5130. This shows up as a separate ring on *netlook* with very bright, almost white, lines leading back to each of the workstations participating in the game. Call these folks up and ask them to wait until after hours.

Netlook is also very useful for finding out whether somebody else is causing your headaches. Unless you have remote snoopers running, you will not see any end-point activity which begins and ends outside of your network. If you do, it is because it is going through your network. That is to say, others are routing their ethernet activity through your network, and, in the process, eating up a portion of that all-important 350,000 bytes per second.

Section 5.3 — Using gr_osview.

This package is a system performance monitoring tool. Unlike the transceiver LEDs and *NetVisualyzer* which monitor overall ethernet activity, *gr_osview* gives you an in-depth understanding about when and why a particular workstation is generating so much ethernet traffic. Most important are the Real Memory Utilization and CPU Wait bars. The memory utilization bar tells you if there is enough memory. If there is little or no free memory (green) you should think seriously of adding more main memory. The CPU Wait bar tells you what's happening. Most important, whether the system is waiting on normal I/O (i.e. that caused by the application program's reads and writes which are in blue) or swapping (in yellow). As a good rule of thumb, ***Do not swap over the ethernet!*** If you see any appreciable amount of swapping (yellow) for more than a few seconds at a time, that workstation is swapping too much, and you must do something about it.

Section 5.4 — Using sysmeter.

While most of the functions of *sysmeter* are covered by *gr_osview*, it does have several advantages in that it allows for a simpler display of just what you're interested in, plus it will show the collisions generated on the workstation its being run on. This second feature is of particular interested if you aren't blessed with ethernet transceivers with LEDs.

Section 6.0 — The Causes of Poor Ethernet Performance.

Let's pretend that you're the administrator of a diskless workstation network, and there is grumbling about performance. What can you do about it?

Obviously, before you can fix the problem, you need to understand it. As noted many times above, the root cause of poor ethernet performance is that the bandwidth is only 350,000 bytes per second in real-word situations. The first thing to do is to measure the utilization. So you run *netgraph* and sure enough, your network is running at over 30% utilization, not that you needed *netgraph* to tell you that because the collision LED on your system's ethernet transceiver will be glowing red, like a flaming ember. Following the advice given in Section 5.3, you personally went around and persued everyone to stop playing dogfight during peak hours. Still, your network is acting sluggish and one quick glance at the LEDs on the transciever in your office tells it's still overloaded. So you again fire up *netgraph* and discover that almost the entire load on you network are NFS packets. Now what?

The answer is to reduce the load. But to do that you first need to understand where the NFS load is coming from. If you were to divide this load, you would find that it falls into four categories:

- Program loading: This may be a considerable load when a diskless workstation is booting. However, except for booting, these loads, though heavy, are brief. Even booting a diskless workstation only takes a few minutes and, once done, is no longer a problem. Normally this is only a problem first thing in the morning when everybody is starting up their systems. The fix for this is simple—don't turn the systems off at night.
- Normal "disk" I/O: What is meant by this is the I/O caused by the application program doing disk reads and writes on data files. You should be able to spot this easily with the various tools, particularly *gr_osview*. We will look at this problem and what you can do about it in a moment.
- "Disk" I/O to `/tmp`: This is a special case of the above, and while very similar in nature, these files contain temporary information used by programs such as compilers. As we shall see, the cure is a local disk.
- Swapping: This is the single largest load on a normal ethernet when diskless workstations are connected to it. Not only can it occur frequently, but when it does, the transfer is often very large, in the order of Megabytes. The cure for this is also a local disk.

Section 6.1 — Fixing the Problems—Part I.

Now that we understand the problem, how do we figure out who is doing what to your network? First, make a list of all of the workstations on your network, including whether or not it has a disk, and how much main memory each has. Also include, if you can, what the station is used for, whether it be for office automation (spread sheet, e-mail, etc.), data entry, software development, or whatever. If nothing else, a friendly chat with the users will usually give you an idea of what they are doing.

Next, fire up *netlook* and study the traffic patterns. You should pay particular attention to which workstations are the most active as far network load, and what, if any workstations tend to have collateral transfers between themselves. Often, you'll find that the workstations used by software developers working on the same project spend a lot of time talking to each other. You'll want to keep track of this because if you decide to split the network, you'll definitely want to keep workstations that frequently communicate with one another on the same sub-network. Also, as noted above, keep an eye peeled for activity which begins and ends outside of your network. If you see any, it's probably due to others routing their packets through your network and increasing your network's load. Often you'll have no idea what this routing-through load is until you investigate with the *NetVisualyzer* tools.

Finally, do an inventory of what sort of activity is going on in each system on your network. This may be time consuming, but at least you can do this in your own office. And you'll only be interested in the workstations with in the heaviest network load, anyhow. You can use the *analyzer* tool of *NetVisualyzer* to collect, say, 1000 packets, save them in a file and then select only those packets for the workstation you're interested in by using the filter capability to re-read the file. (In case you don't know how to do this, move the cursor to the top line of the capture window, the one marked "filter" and type `ip.host hostname`, where "`hostname`" is the name of the host you're interested in checking. You'll s ee only those packets sent to and from that host.)

Pretty soon patterns begin to develop. You'll learn which systems tend to interact with what other systems and which tend to put the heaviest load on the network. You'll also now have several options available to remedy the problem, which include:

- Split the network: Remember, the bandwidth of a (i.e. ***one***) ethernet is 350,000 bytes per second. If your have a sufficiently powerful server, you can add additional ethernet cards and have two, three, or even four ethernets attached to it. Typically, you would want to put workstations that often communicate with each other on the same ethernet so as to minimize the load on the gateway. Remember, ethernet packets that cross gateways are a load on all the networks they traverse, so strive to keep such traffic to a minimum.
- Add more main memory to some or all of the workstations: Any 8 Megabytes workstations are definite candidates for this because, because, almost by definition, they are swapping too much. This is due to the unix kernel requiring four to six megabytes just for itself. You can verify this by running *gr_osview* either locally or remotely (read the man page!) and seeing if there is excessive swapping. Excessive swapping is defined as you seeing yellow on the CPU Wait bar for more than a second or so—and don't be surprised if it goes on for a minutes. Often, doubling a workstation's RAM from 8 Megabytes to 16 Megabytes decreases the amount of swapping ten-fold. You'll have to experiment to find out just how good the results will be in your situation, but definitely take a look at it.
- Add a local disk: While this seems a contradiction in terms, there is usually no reason why you can't have one (beyond price, that is). Most of the rationale for having a diskless workstation is not for low price, but for software control. Even high-security environments will permit a local disk as long as: 1.) no software is ever loaded on it, and 2.) the disk is locked up when not in use. The PI is a perfect candidate for this because the 5.25" SCSI disk can be easily removed and locked in a safe at night. The idea is to put the swapping on the local disk, off loading it from your network. This configuration is called "super-diskless" by those who have tried it, mainly because of the performance increase, often ten-fold. In addition, if you have software developers, you can also put their `/tmp` on the local disk as well. This, by itself, may off-load your network by half if you have an number of users doing compiles.

Section 6.2 — Fixing the Problems—Part II.

Normally, adding extra RAM to the 8 Megabyte workstations and giving heavy-duty users a local disk for local swapping and `/tmp` does wonders. And instead of having to sneak down to the coffee machine, you can walk down the hallway, standing proud and tall, completely free of fear of being accosted by angry users. Splitting the network and adding RAM and local swap disks will fix most of the problems you're likely to run into, particularly those caused by swapping and `/tmp` usage.

However, there remains one class of ethernet load which may prove resistant to these fixes and that is the case in which the application uses a great deal of data. There is a sure-fire solution for this, and that is to put the data on a local data disk which is used much like the local swapping disk. This configuration is often called "semi-diskless" because the only data is on the disk. (In fact, the "super-diskless" configuration noted above is a "semi-diskless" configuration.) However, let's assume that you can't use that approach, and for whatever reason, the data must remain on the server. It's this challenge we shall now address. The answer, we shall see, is to add more RAM. The question is where.

As you should remember from Section 3.0, IRIX 3.3 has two major enhancements, delayed write buffering (typically on the server) and private mounting of NFS files by the client. Both require available (i.e. free) memory. Thus, if you are allowing large background tasks to run on the server and permit them to use up all available memory, you are not going to see much delayed write buffering. You must have memory left free and available for buffering on the server and caching on the client. You can easily check this by running *gr_osview* and watching the Real Memory Utilization bar. If you don't see red (*delwri*) or green (*free*), you will have to do something; either you'll have to kick off some of those background tasks or add more main memory.

Let's look at this in a little more detail. If there is a delayed write buffer active (it's the red segment of the Real Memory Utilization bar), estimate how large it is. The bar is divided into ten divisions, so for a 24 Megabyte system, each division is 2.4 Megabytes. Generally, in the case of a server, there should be between 12 and 16 Megabytes of combined *free* (green) and *delwri* (red) memory available. If there isn't, do what it takes to make it happen—i.e., add main memory or remove some background tasks. This buffering is used for a number of good purposes, the most important being the read-ahead buffering and cache-reading by the server. Without adequate buffering, you will find that your server is spending a large part of its time waiting on I/O (also determinable by *gr_osview*.)

Next, we'll turn our attention to the client workstation and check the possibility of mounting the data files as private. If the files being used by the client's applications are being written into by several different clients simultaneously, there is little you can do. You can not mount such files as private because the file locking must remain on the server. However, if the data files are being used by only one client, then you should certainly try this approach. The problem is if there is no available free memory on the client workstation, nothing much will happen. This, of course, can be check by using *gr_osview* exactly the same way you did it on the server. The rule of thumb in the case of the workstation is that you need about 8 Megabytes of combined *free* and *delwri* space on the client. The *delwri* space is where the private mount caching is occurring. If, once you enable private mounting, you notice that all *delwri* space has gobbled up all of the *free* memory, seriously consider increasing the RAM on the client some more.

Section 7.0 — Recommended Numbers of Diskless Workstation per Ethernet.

If you have just flipped through this paper to this section without having bothered to read what was said in previous sections, please go back and read them now. There is no greater abuse of "recommendations" than to apply them without a clear understanding of the issues involved. As noted previously, diskless workstation environments are highly individualistic, especially when it comes to the users. What one set of users consider "state-of-the-art" and an major improvement over their previous computer equipment might cause another group of users to openly rebel.

The following should be used as a rule-of-thumb for "average" diskless workstations on an "average" network with "average" users. In other words, these suggestions are just a starting point to give you some idea of what to expect. You must, however, continuously measure and analyze the performance of your diskless workstation network and make appropriate changes and upgrades as necessary.

As noted repeatedly, what you are actually doing in a diskless workstation environment is taking roughly 350,000 bytes of available traffic per second and dividing it up among the various users. And remember,

just one workstation that is doing excessive swapping and/or `/tmp` activity can usurp that entire bandwidth.

In addition, we are assuming that you are using one of the servers configured in table 2 (page 11) as the server for the clients. In particular, we are assuming that you are not using either a 4D/25 or a 4D/120 system for a server as they are not sufficient for serving the NFS load that could be generated by the number of clients listed in table 3 should they are be fairly active. These systems do make fine servers, but only in moderate loading conditions.

Last but not least, a note regarding the nomenclature used in table 3. We have presented the number of clients as a function of being on one ethernet. If you have a large 4D/310 with at least two disk drives and 80 Megabytes of main memory, you should be able to drive four ethernets, each with the number of systems listed in table 3.

Table 3. Suggested number of diskless clients per ethernet. These numbers assume that you are following the recommendations given in table 2 regarding the configuration of the server and that you are using IRIX 3.3.

Memory	Client Model	Configuration	Clients per Ethernet
8MB	4D/25	With NO disk (i.e. Bare Bones)	2-4 Clients
16MB	4D/25	With NO disk	5-6 Clients
16MB	4D/25	With local swapping (and `/tmp`) disk	12-16 Clients

Some explanations:

The 8MB system, by its nature, will swap excessively.

The 16MB system with no disk will, if you are careful, not swap excessively but will nevertheless do so from time to time.

The addition of a local disk for swapping and `/tmp` use eliminates much of the uncertainty caused by swapping and compilation loads. You must, however, still look at the I/O load imposed by the applications.

Section 8.0 — Sysgening IRIX 3.3 Diskless Workstations.

Building the share and client tree for IRIX 3.3 is not difficult. However, it is something that is a little out of the ordinary, and so we are presenting a step-by-step example of how to do this. We are going to first go through the standard procedure and build a truly diskless workstation, and then we are going to through the changes you must make to add disks to the clients. These are, of course, the "super-diskless" and "semi-diskless" configurations.

Section 8.1 — Standard Sysgening of IRIX 3.3 Diskless Workstations.

The first step in generating a diskless server is to build a share tree. You ***must build one for each class of client.*** That is to say, you need a different one for IP6's, IP5's, etc. The reason for this is that the unix kernel is different. What follows is an example of how to build one for the Personal Iris, or IP6. You can build additional share trees in exactly the same manner. The only differences are in the parameters in the `clinst.dat` files you make for each class of client.

If you have any doubt (and you should) about what type of systems can be clients of the same share tree, look at the table in `clinst.dat` in the `/usr/etc/boot` directory. A copy of this is also in Section 8.1.1, which follows. Please note that there are entries for IP4, IP6, IP5, IP7, and IP9 type systems. Many of you know that the IP5, IP7, and IP9 systems are "virtually" identical except for clock cycle speed, data cache design, etc. ***Do not treat them as the same!*** They aren't identical. You must have a separate share tree for each. Secondly, the graphics subsystems are also a concern. There is a column marked "GFXBOARD." ***Only those systems with the same IP type and GFXBOARD type may be on the same share tree.*** Thus the 4D/20, 4D/20G, and 4D/25 may share the same share tree, but, in the case of the IP4, you'll need two separate share trees if you have a combinations of CLOVER1 and CLOVER2 type graphics subsystems. Likewise, the IP5, IP7, and IP9 systems must each have their own share trees as well. There are very good reasons why engineering went through all the trouble of classifying the various systems according to IP and GFXBOARD types. And although the systems may be 99.99% identical to one another, it's that last 0.01% that's going to get you.

Therefore, your very first order of business is to draw up a list of all of your diskless clients and organize them according to share tree type. Once that is done, collect the distribution tapes. We strongly advise that you use only officially released SGI distribution tapes. We all know that there are backup copies of these tapes floating around. These were made with *distcp*, and while they probably will work just fine, then again they might not. If you are using backup copies and you experience problems, switch to the official copies.

You will need as a minimum *eoe1, eoe2, and nfs*. Most of you will want *dev* as well, but it is not required. Also, please note that you should not use IRIX 3.3.1 for building diskless workstations. Use IRIX 3.3.0, 3.3.2 or later versions.

Now to work. You must, of course, have super-user privileges on the server, so either login as root, or run *su* and become super-user. Then *cd* into the `/usr/etc/boot` directory. This is what it looks like in it's virgin state.

```
#cd /usr/etc/boot
#ls
clinst      clinst.dat  makedev     rclinst
```

Section 8.1.1 — Editing the clinst.dat file.

Next, copy the `clinst.dat` file to a suitably named work file. ***Never*** use the `clinst.dat` file itself as a work file because you might need to build another share tree for a different type client. If you are building several share trees, name each copy of the `clinst.dat` file according to software versions and hardware type. In this example, we chose `3_3_PI` to signify IRIX 3.3 for the PI.

```
#cp clinst.dat 3_3_PI.dat
#chmod +w 3_3_PI.dat
#vi 3_3_PI.dat
```

Below is the `3_3_PI.dat` file after it has been edited. Comments added for the purposes of this paper are in italic. Changes are in **bold**.

```
#       @(#)clinst.dat 1.1 8/11/88
#
#       SERVER ENVIRONMENT SETUP
#
#       HOSTNAME  : Do not change
#       BOOTP_DIR : Do not change
#       YP        : yes - client setup is through YP.
#                   no - client setup is done locally.
#       YPDOMAIN  : If set to any value, then ypdomain is set
#                   as specified. If set to null, then client's
#                   ypdomain is set as diskless server's ypdomain
#                   unless client's ypdomain is set previously.
#
HOSTNAME="`hostname`"
BOOTP_DIR="/usr/local/boot"
YP="no"
```

If you aren't using YP on your server, use a fake domainname as described in Section 8.1.6 of this paper.

```
YPDOMAIN=""
#
# SHARE TREE SETUP
#
#       SHAREHOST : Do not change.
#       DISKLESS  : directory where all the share trees reside.
#       SHARE     : Do not change
#       USESRVROOT: If this flag set to "yes", then clinst will make
#                   server's root(/) share-tree.
#
SHAREHOST="$HOSTNAME"
DISKLESS="/d/diskless/PI"
```

Don't make this a really long name because there is only 50 characters in the NVRAM and clinst *will limit this (with the server's name and client's directory) to 37 characters. Please notice that the next line picks up the CLASS from the* clinst *command we use later.*

```
SHARE="$DISKLESS/$CLASS"
USESRVROOT="no"
```

NOTICE —Do not use USESRVROOT!

This feature does not work in several situations and has been removed in IRIX 3.3.2. Please do not try to use it in 3.3.0. If you have IRIX 3.3.2, this variable will not be present in the clinst.dat file.

```
#
#       CLIENT TREE SETUP
#
#       CLROOT    : directory where client tree resides.
#       SWAP      : directory where swap file resides.
#       SWAPSIZE  : size of swap file.
#                   Same format as "mkfile" takes.
#       LOCALDISK: Include efs option so that file system on
#                   local disk also works.
```

```
#     NOTE:
#             If one wants to prevent the client of using
#             swap space on server, then set SWAPSIZE="0".
#             However, If one still wants to preserve already
#             existing swap space on server for some reason,
#             then set SWAP="".
#
CLROOT="$DISKLESS/$HOST"
SWAP="$DISKLESS/swap/$HOST"
SWAPSIZE="20m"
```

You may want to have a local swap disk, in which case you MUST set this to SWAPSIZE="0". However, for now, we are assuming a swap space on the server. Normally, 20 Megabytes is more than enough. If you need more, consider a local swap disk—you're doing too much swapping. Unlike most of the other parameters in this file, SWAPSIZE is only used when building or removing a client tree. You can also change it between building individual client trees, allowing you to have different size swap files for different clients.

```
LOCALDISK="yes"
```

The above is normally "no," and for secure installations where a local disk is absolutely NEVER wanted, leave it "no" for it leaves the efs.o file out of your kernel. However, if you ever want to add local swap or data disks, change this to "yes" before you build the share tree, or you will have to start from scratch. Our strongest advice is to change this to "yes" unless you are ABSOLUTELY POSITIVE that you'll never use this feature. As far as we can see, the only such "absolutely positive" reason you'll never use a local disk is due to security issues.

```
#
# CLIENT CPUBOARD/MACH/GFXBOARD SETUP
#
```

Machine Type	CPUBOARD	MACH	GFXBOARD
4D20	IP6	IP6	"ECLIPSE -mSUBGR="
4D20G	IP6	IP6	"ECLIPSE -mSUBGR="
4D25	IP6	IP6	"ECLIPSE -mSUBGR="
4D25G	IP6	IP6	"ECLIPSE -mSUBGR="
4D50G	IP4	IP4	"CLOVER1"
4D50GT	IP4	IP4	"CLOVER2 -mSUBGR=IP4GT"
4D70G	IP4	IP4	"CLOVER1"
4D70GT	IP4	IP4	"CLOVER2 -mSUBGR=IP4GT"
4D120GTX	IP5	IP5	"CLOVER2 -mSUBGR=IP5GT"
4D210GTX	IP9	IP9	"CLOVER2 -mSUBGR=IP9GT"
4D220GTX	IP7	IP7	"CLOVER2 -mSUBGR=IP7GT"
4D240GTX	IP7	IP7	"CLOVER2 -mSUBGR=IP7GT"
4D280GTX	IP7	IP7	"CLOVER2 -mSUBGR=IP7GT"

```
CPUBOARD="IP6"
MACH="IP6"
GFXBOARD="ECLIPSE -mSUBGR="
```

The above are the defaults and assume PI clients. Change all three as needed if you have another configuration. If you have different types of clients with incompatible IP models, for example 4D/25's and 4D/120's, you have to have multiple share trees, one for each type of CPU and graphics board. Change CPUBOARD, MACH, and GFXBOARD according to the values giving for each type of system in the table above. Also, never modify these three parameters once defined for a share tree.

```
#
# INTERNAL USE ONLY(DO NOT CHANGE)
#
DLMAJOR=18
#
```

Section 8.1.2 — Building the Share Tree.

At this point we are ready to build the share tree. There are several ways you can do this. The easiest is from tapes on your server's streaming tape drive. Some of you might have a remote tape drive, and others of you might be lucky enough to have a software distribution on ethernet. We'll show you how to use all three. To start, make certain that you are in `/usr/etc/boot` and type the following. Please note that the ".dat" is assumed on `3_3_PI.dat`. Do not type "`.dat`" in.

`# ./clinst -c 3_3_PI share`

What you see is as follows:

```
About to install shared tree at /d/diskless/PI/3_3_PI......
Enter confirmation (y/Y):y

Ready to install software.

  Choose an item, then press <enter>:

     1. Automatically install software
     2. Use manual installation features

     3. Help
     4. Quit

Install 2
```

Please note that we chose manual installation for the share tree because we want to be certain that the share tree contains only those files we want. While this is a bit of trouble and may take a little more time, it is worth the bother. At this point, we'll interrupt the standard procedure which assumes a local tape drive, and show you how to use a remote tape drive and then a ethernet distribution. For those of you who may wonder how you can build an network distribution, try reading the man page for *distcp*.

Returning to the example:

```
Install 2

Manual Installation

  New software will be read from tape.
  Installation root is /d/diskless/PI/3_3_PI; mode is normal.

  /dev/dsk/dks0d2s6      mounted as  /

  1. from [source]       Get new software from [source]
  2. list [names]        List available subsystems
  3. standard            Install the default set of subsystems
  4. all                 Install all available subsystems
  5. select              Examine/choose subsystems to be installed
  6. recalculate         Recalculate space required for installation
  7. clean               Clear all files from / and /usr file systems
  8. admin               Miscellaneous administration functions
  9. return              Return to previous menu
 10. help [item]         Get help in general or on specific subjects
 11. quit                Terminate software installation
```

Next we select number 1, "from" and see the following:

```
Manual 1

Read software from where? (tape) TYPE_ANSWER_HERE
```

In place of "**`TYPE_ANSWER_HERE`**," you can type either

guest@big_sgi:/dev/tape *If you are using a remote tape, or*

guest@big_sgi:/d/4D/3.3.0 *if you have a network software distribution*

Returning to the mainstream of installing diskless software, we're going to assume that the majority of users are going to use tape. So that's what we are going to do. Sooner or later, you'll get a new prompt of "Manual," either after you selected a remote tape, or as in the majority of cases, the first time you responded to install mode prompt. No matter how you load the software into the share tree, you must load the following as a minimum: *eoe1, eoe2*, and *nfs*. In this example, we are also going to load *dev* as well, since most users do use it. However, remember to check the amount of disk available in the partition you are loading the share tree. It will use a good deal of space—usually in the order of 50 to 80 Megabytes.

Load the *eoe1* tape into the drive, and let's continue. Since we want to check the installation procedure as it happens, chose "5" for "select." What you'll see is as follows:

```
Manual 5
Subsystem Selection

  New software will be read from tape.
  Installation root is /d/diskless/PI/3_3_PI; mode is normal.

  1. list [names]          List current subsystem selections
  2. files [names]         List files in subsystems
  3. install [names]       Request subsystem installation
  4. remove [names]        Request subsystem removal
  5. keep [names]          Request no action; keep existing versions
  6. default [names]       Request default installations
  7. step [names]          Use interactive step mode for requests
  8. recalculate           Recalculate space required for installation
  9. go                    Perform requested installations and/or removals
 10. return                Return to previous menu
 11. help [keyword]        Get help in general or on specific subjects
 12. quit                  Terminate software installation
```

At this juncture, we want to review the installation, so select "7" for "step." This way *inst* works interactively. The *inst* program will show you a line item and its present status. The letter "i" means it will be installed. The letter "k" means that it's already been installed in a previous run of *inst* (*clinst* calls *inst*). Typically, because we are installing the share tree for the first time, all you'll see in the first column "i" or " " (i.e. a space). The latter means "do not install."

Type "7." The following is what you'll see.

```
Select 7

Reading product descriptor from tape:

    eoe1        4D1-3.3 Execution Only Environment (part 1)

Computing disk space changes:

i    eoe1.man.relnotes          437+ Workstation Release Notes
i    eoe1.man.unix             4580+ Basic UNIX
note, we've changed first column to "i" to include man pages.
i    eoe1.sw.unix             32070+ Basic UNIX
[Bottom - ? for options]
```

Since we usually forget what the options are, type a "?" to see what they are.

```
 Select ?
Subsystem Selection
  New software will be read from tape.
  Installation root is /d/diskless/PI/3_3_PI; mode is prototype.
  1. list [names]         List current subsystem selections
  2. files [names]        List files in subsystems
  3. install [names]      Request subsystem installation
  4. remove [names]       Request subsystem removal
  5. keep [names]         Request no action; keep existing versions
  6. default [names]      Request default installations
  7. step [names]         Use interactive step mode for requests
  8. recalculate          Recalculate space required for installation
  9. go                   Perform requested installations and/or removals
 10. return               Return to previous menu
 11. help [keyword]       Get help in general or on specific subjects
 12. quit                 Terminate software installation
```

Since we have selected what we want from the eoe1 tape, type "9" to have it do the install.

```
Select 9
```

Now we wait about 20 minutes while the tape loads.

```
Installing new versions of selected eoe1.sw subsystems
Installing new versions of selected eoe1.man subsystems
Installation succeeded.
Select
```

Now change tapes and do eoe2.

```
Select 7
Reading product descriptor from tape:
 no    eoe2       4D1-3.3 Execution Only Environment (part 2)
Computing disk space changes:
```

In this case, we want to be certain we don't select stuff we don't want. The following is a reasonable selection. Please note that you would want to include X11 in future releases of IRIX. What we are showing here is IRIX 3.3, which uses NeWS.

```
    eoe2.man.X11                1127+ X11 Execution Environment
    eoe2.man.bsdlpr                6- BSD Line Printer Spooling Utilities
    eoe2.man.demos               329+ Graphics Demonstration Programs
    eoe2.man.spaceball            30+ Spaceball Documentation
i   eoe2.sw.NeWS              13060+ 4Sight Windowing System
    eoe2.sw.X11                 4385+ X11 Execution Environment
    eoe2.sw.Xapps              23203+ X11 Applications
    eoe2.sw.Xdemos              2888+ X11 Demos and Images
    eoe2.sw.Xfonts              4735+ Additional X11 Fonts
    eoe2.sw.acct                 937+ System Accounting
    eoe2.sw.bsdlpr              1101+ BSD Line Printer Spooling Utilities
    eoe2.sw.cdsio                 46+ Multiport Serial Board Support
i   eoe2.sw.crypt                 58+ Security Administration Utilities
    eoe2.sw.demos               4796+ Graphics Demonstration Programs
i   eoe2.sw.dfm                  999+ Directory and File Management Utilities
l   eoe2.sw.editors              773+ Editing Utilities
i   eoe2.sw.envm                6871+ Visual File System Interface
i   eoe2.sw.gltools              849+ Graphics Library Tools
    eoe2.sw.hyper                198+ HyperNet Support
    eoe2.sw.ikc                   30+ Ikon Printer Interface
i   eoe2.sw.ipc                  242+ Inter-Process Communication Utilities
i   eoe2.sw.ipgate              646+ IP Network Gateway Support
i   eoe2.sw.lp                 1516+ Line Printer Spooling Utilities
i   eoe2.sw.mast                   0  Graphics Master
    eoe2.sw.moregltools         2613+ More Graphics Library Tools
    eoe2.sw.optfonts            4546+ Optional Fonts
i   eoe2.sw.perf              1606+ Performance Measurement Utilities
    eoe2.sw.spaceball            778+ Spaceball Software
    eoe2.sw.spell                865+ Spell Utilities
```

```
i  eoe2.sw.sysadm              523+ System Administration Utilities
i  eoe2.sw.tcp                8475+ TCP/IP Networking Support
   eoe2.sw.terminf            2602+ Terminal Information Utilities
   eoe2.sw.ts                   29+ VME 1/4" Streaming Tape Support
i  eoe2.sw.uds                  36+ UNIX Domain Socket Support
i  eoe2.sw.usrenv             1051+ User Environment Utilities
   eoe2.sw.uucp               2251+ UUCP Utilities
i  eoe2.sw.vadmin             8852+ Visual System Administration Interface
   eoe2.sw.xm                   61+ Xylogics 1/2" Tape Support
   eoe2.sw.xylsio               56+ 16-port Serial Board Support
```

When finish, run inst.

```
Select 9

Installing new versions of selected eoe2.sw subsystems
Installation succeeded.
```

It will take about twenty minutes to copy in the tape. Once you see the "Installation succeeded," change tapes to the nfs *tape and type "7" in response to "Select."*

```
Select 7

Reading product descriptor from tape:

    nfs         4D1-3.3 Network File System

Computing disk space changes:

i   nfs.man.nfs                341+ NFS Support
i   nfs.man.relnotes            15+ NFS Release Notes
i   nfs.sw.dskless             231+ Diskless Support
i   nfs.sw.nfs                2862+ NFS Support
i   nfs.sw.yp                 4024+ Yellow Pages

Select 9

Installing new versions of selected nfs.sw subsystems
Installing new versions of selected nfs.man subsystems
Installation succeeded.

Select
```

Now change to the dev *tape and install dev selectively.*

```
Select 7

Reading product descriptor from tape:

    dev         4D1-3.3 Development System

Computing disk space changes:

    dev.man.cc                4707+ C Compiler and Support
    dev.sw.G0libraries       10080+ Libraries Compiled -G 0
    dev.sw.NeWSimg           12719+ 4Sight Images
i   dev.sw.bsdhdrs             70+ Pre-3.3 BSD Header File Compatibility
i   dev.sw.cc                21228+ C Compiler and Support
    dev.sw.cedgetut             15+ C Edge Tutorial
i   dev.sw.crypt                29+ Security Administration Library
    dev.sw.debug               865+ Debugging Kernels
    dev.sw.giftssrc           7125+ Gifts Source
    dev.sw.moredemos          5778+ More Graphics Demonstration Programs
    dev.sw.rcs                1677+ Revision Control System Utilities
    dev.sw.sccs               1689+ Source Code Control System Utilities

Select 9

Installing new versions of selected dev.sw subsystems
Installation succeeded.
```

*Now that we have loaded all the tapes we want in our share tree. Now type "12" to complete installation. You **must** do this or it will not work. Do **not** control C out or anything like it; if you, you'll have to do the install all over again.*

```
Select 12
```

The system is busy for four or five minutes and then you'll see.

```
#
```

You have now built the share tree. Let's see how big it is.

```
# du -s /d/diskless/PI/3_3_PI
116324  /d/diskless/PI/3_3_PI
```

That's a little over 56 Megabytes (hint: divide 116324 by 2048).

A word of warning: If you saw any error messages, don't bother to continue. There is something wrong, and the diskless environment either will not work, or it will be flaky. You must have done one of the following:

- You made a mistake while editing the share tree `.dat` file.
- You selected out one of the vital elements of the system while you stepped through *inst.*
- You ran of disk space or some such thing.
- You used a bad distribution media.

The most likely cause of a share-tree build problem is bad media. Maybe you just have a bad tape, or if the tape is a copy, somebody copied it incorrectly. And if you are using a ethernet-based distribution, don't assume that something isn't wrong in that distribution. There is one case we know of where the ethernet-based distribution had been tampered with. If you suspect that you have a distribution problem, get a new, officially released set of distribution tapes and try again. The share tree must build without error.

Section 8.1.3 — Building the Client Tree.

Now that we have the share tree built, let's build a client tree. You have to build one for each client on the share tree. While the bad news is that you have to build one for each client on your network, the good news is that its much easier than the building the share tree. And since it uses the product descriptors in the share tree, you can use the automatic option. As with building the share tree, you use *clinst.* It takes about ten minutes to build each client tree. You ***must build them one at a time*** since *clinst* writes into the server's `/etc/exports` and `/etc/bootparams`. You might garble these files if you run more than one copy of *clinst* at the same time. The client's name must appear in the server's `/etc/hosts` file.

```
#cd /usr/etc/boot
./clinst -c 3_3_PI -h client_1 client
Client tree = /d/diskless/PI/client_1, shared tree = /d/diskless/PI/3_3_PI
Enter confirmation (y/Y) :y
Create 20m swap file ........
Ready to install software.
  Choose an item, then press
              :
     1. Automatically install software
     2. Use manual installation features
     3. Help
     4. Quit
Install 1
```

```
Installing new software.

Reading product descriptors from /d/diskless/PI/3_3_PI/usr/lib/inst:

    eoe1        4D1-3.3 Execution Only Environment (part 1)
    eoe2        4D1-3.3 Execution Only Environment (part 2)
    nfs         4D1-3.3 Network File System
    dev         4D1-3.3 Development System

Computing disk space changes:


Installing new versions of selected eoe1.sw subsystems
Installing new versions of selected eoe1.man subsystems
Installing new versions of selected eoe2.sw subsystems
Installing new versions of selected dev.sw subsystems
Installing new versions of selected nfs.sw subsystems
Installing new versions of selected nfs.man subsystems
Done.
Is there more software to install? n
sh: ./tagscript: not found
```

The above error message is not a problem—just ignore it.

Now that we have a client tree built, let's poke around the server and see what has happened. First let's look at `/usr/etc/boot` which will have the following files.

```
# ls
3_3_PI.dat          client_1            clinst.dat          rclinst
bootparam.client_1  clinst              makedev
```

There are two new files: `client_1` and `bootparam.client_1`. The `client_1` directory is actually a link to `/d/diskless/PI/client_1`, as the following shows:

```
# ls -l client_1
total 1
l—    1 root    sys       29 Oct 22 14:45 unix - /d/diskless/PI/client_1/unix
```

The other new file, `bootparam.client_1`, is a temporary file used to update `/etc/bootparams`. Let's look at it. The server's name is "sandy".

```
# more bootparam.client_1
root=sandy:/d/diskless/PI/client_1
share=sandy:/d/diskless/PI/3_3_PI
swap=sandy:/d/diskless/PI/swap/client_1
```

Let's compare the above with the actual `/etc/bootparams` file. As you can see, it was transcribed in a fairly straight forward manner. (By the way, the `/usr/etc/boot/bootparam.client_1` file is of no further use and may be deleted if you wish.)

```
# more /etc/bootparams
3_3_PI root=sandy:/d/diskless/PI/3_3_PI share=sandy: swap=sandy:
client_1 root=sandy:/d/diskless/PI/client_1 \
share=sandy:/d/diskless/PI/3_3_PI  \
swap=sandy:/d/diskless/PI/swap/client_1
```

(Please note that the actual entry in `/etc/bootparams` for client_1 is one long, continuous line. We inserted the "\" in the above example for the sake of clarity. ***Under no circumstances should you ever edit this file***. Leave `/etc/bootparams` alone; you can cause the booting process to fail because you made what you think is a harmless change.)

Now let's check the server's `/etc/exports` file.

```
# more /etc/exports
#
#NFS exported filesystem database (see EXPORTS(4) for more information).
#
# Entries in this file consist of lines containing the following fields:
#
# filesystem    [ options ]     [ netgroup ] [ hostname ] ...
#
# Filesystem must be left-justified and may name any directory within a
# local filesystem.  Lines beginning with white space continue the previous
# line's entry.  Netgroup(4) and hostname refer to machines or collections
# of machines to which filesystem is exported.
#

/d/diskless/PI/3_3_PI -ro,anon=root    #class=3_3_PI
/d/diskless/PI/client_1 -rw,anon=root,access=client_1  #host=client_1
/d/diskless/PI/swap/client_1 -rw,anon=root,access=client_1     #host=client_1
```

As you can see, all of the appropriate entries have been made. If you were to look into the `/d/diskless/PI/client_1/etc/fstab` you would find matching entries in the client's `fstab`.

Finally, let's see how much space we have used up on the server's disk to build this client tree. As you can see, the actual `client_1` directory is very small, and in fact consists mainly of `/unix` as well as `/etc` files such as `passwd, hosts, sys_id, fstab,` and so forth; all of which have been automatically filled in by *clinst.*

```
# pwd
/d/diskless/PI
# du -s *
116323  3_3_PI      (this is about 58 Megabytes)
4284    client_1    (this ia about  2 Megabytes)
40962   swap        (this is about 20 Megabytes)
```

Section 8.1.4 — Updating to IRIX 3.3.2

The procedure for installing an update such as IRIX 3.3.2 is fairly obvious once you understand how *clinst* works. First, you run *clinst* to add the update on each of the share trees you have previously built using IRIX 3.3, and then you repeat the procedure on each of the client trees attached to that share tree.

However, their is a catch in this, and that is when you are ***ADDING** a new client after you updated the share tree*. While, at first, it may seem illogical, you must run *clinst* ***TWICE*** in this situation. The reason why this is required is because *inst* is used by *clinst* to build the tree and *inst* was designed to work in logical steps. The *inst* program expects to see a base-level IRIX system before it progresses to installing the update. If it doesn't find a 3.3 base system, it insists on building one first so that it has the files need to check against the data it receives from the update tape. While this may be an inconvience, it is safer—a lot safer—doing things this way.

Since this might cause confusion the first time you perform this update, we have included the actual screen output from performing 3.3.2 updates to a pre-existent diskless share tree as well as from adding a new client. The following is the output from *clinst* to update a pre-existent share tree with IRIX 3.3.2. If you are building a new diskless environment with IRIX 3.3.2, add all of the updates to the share tree first, and then built the clients using the "adding a new client" procedure.

The following is the procedure or updating a share tree.

```
#./clinst -c 3_3_PI share

WARNING: Class 3_3_PI is still serving clients.
continue to update 3_3_PI(Y/N)?y

Checking clients status:

    client client_1 ... DOWN

WARNING: clients will be out of sync. Clients which are UP may crash.

About to install shared tree at /d/diskless/PI/3_3_PI......
Enter confirmation (y/Y):y

Ready to install software.

  Choose an item, then press
            :

     1. Automatically install software
     2. Use manual installation features

     3. Help
     4. Quit

Install 2

Manual Installation

  New software will be read from tape.
  Installation root is /d/diskless/PI/3_3_PI; mode is normal.

  /dev/dsk/dks0d1s6     mounted as  /

     1. from [source]          Get new software from [source]
     2. list [names]           List available subsystems
     3. standard               Install the default set of subsystems
     4. all                    Install all available subsystems
     5. select                 Examine/choose subsystems to be installed

     6. recalculate            Recalculate space required for installation
     7. clean                  Clear all files from / and /usr file systems
     8. admin                  Miscellaneous administration functions

     9. return                 Return to previous menu
    10. help [item]            Get help in general or on specific subjects
    11. quit                   Terminate software installation

Manual 1

Read software from where? (tape) guest@bigsgi:/4D/test/4D1-3.3.2
Reading product descriptors from guest@bigsgi:/4D/test/4D1-3.3.2:

    diag        Diagnostics
    maint1      Maint1 Product 4D1-3.3.2
    maint2      Maint2 Product 4D1-3.3.2

Manual 3

Computing disk space changes:
```

```
Installing new versions of selected maint1.eoe1_sw subsystems
Removing old versions of selected maint1.eoe1_man subsystems

Installing new versions of selected maint1.eoe1_man subsystems
Removing old versions of selected maint2.eoe2_sw subsystems

Installing new versions of selected maint2.eoe2_sw subsystems
Removing old versions of selected maint2.nfs_sw subsystems

Installing new versions of selected maint2.nfs_sw subsystems
Installing new versions of selected diag.sw subsystems
Installation succeeded.

Manual quit
```

Now that we have updated the share tree, we will update a ***PRE-EXISTENT*** client tree. This is fairlystraight forward. Just run *clinst*. You should use ***AUTOMATIC*** for this run.

```
# ./clinst -c 3_3_PI -h client_1 client

Client tree = /d/diskless/PI/client_1, shared tree = /d/diskless/PI/3_3_PI
Enter confirmation (y/Y) :y
Create 1 swap file ........

Ready to install software.

  Choose an item, then press
            :

     1. Automatically install software
     2. Use manual installation features

     3. Help
     4. Quit

Install 1
Installing new software.

Reading product descriptors from /d/diskless/PI/3_3_PI/usr/lib/inst:

    eoe1        4D1-3.3 Execution Only Environment (part 1)
    eoe2        4D1-3.3 Execution Only Environment (part 2)
    nfs         4D1-3.3 Network File System
    diag        Diagnostics
    maint1      Maint1 Product 4D1-3.3.2
    maint2      Maint2 Product 4D1-3.3.2

Computing disk space changes:


Installing new versions of selected maint1.eoe1_sw subsystems
Removing old versions of selected maint1.eoe1_man subsystems

Installing new versions of selected maint1.eoe1_man subsystems
Removing old versions of selected maint2.eoe2_sw subsystems

Installing new versions of selected maint2.eoe2_sw subsystems
Removing old versions of selected maint2.nfs_sw subsystems

Installing new versions of selected maint2.nfs_sw subsystems
Installing new versions of selected diag.sw subsystems
Done.
Is there more software to install? n
#
```

Now that we've seen what happens when we update a pre-existent client tree, let's see what happens when you ***CREATE** a new client tree*. In this case, we see error messages which we can ignore, and we also have to run *clinst* **TWICE**!

```
# ./clinst -c 3_3_PI -h client_2 client

Client tree = /d/diskless/PI/client_2, shared tree = /d/diskless/PI/3_3_PI
Enter confirmation (y/Y) :y
Create 1 swap file ........

Ready to install software.

  Choose an item, then press
                :
     1. Automatically install software
     2. Use manual installation features

     3. Help
     4. Quit

Install 1
Installing new software.

Reading product descriptors from /d/diskless/PI/3_3_PI/usr/lib/inst:

    eoe1        4D1-3.3 Execution Only Environment (part 1)
    eoe2        4D1-3.3 Execution Only Environment (part 2)
    nfs         4D1-3.3 Network File System
    diag        Diagnostics
    maint1      Maint1 Product 4D1-3.3.2
    maint2      Maint2 Product 4D1-3.3.2

Computing disk space changes:

Installing new versions of selected eoe1.sw subsystems
/d/diskless/PI/3_3_PI/usr/sysgen/master.d/m333X25: No such file or directory
Can't install usr/sysgen/master.d/m333X25
```

Ignore this error - type 2 for continue

```
Interrupt/Error

     1. stop                    Terminate current procedure
     2. continue                Continue current procedure

     3. help [item]             Get help in general or on specific subjects
     4. abort                   Immediately terminate software installation

Interrupt 2

/d/diskless/PI/3_3_PI/usr/sysgen/master.d/sat: No such file or directory
Can't install usr/sysgen/master.d/sat
```

Ignore this error - type 2 for continue

```
Interrupt/Error

     1. stop                    Terminate current procedure
     2. continue                Continue current procedure

     3. help [item]             Get help in general or on specific subjects
     4. abort                   Immediately terminate software installation

Interrupt 2
```

```
Installing new versions of selected eoe1.man subsystems
Installing new versions of selected eoe2.sw subsystems
Installing new versions of selected nfs.sw subsystems
Installing new versions of selected nfs.man subsystems
Installing new versions of selected diag.sw subsystems
Done.
Is there more software to install? y
```

We really should have answered "n,".but we typed "y" just so you can see what happens if you do.

```
Ready to install software.

  Choose an item, then press
            :

     1. Automatically install software
     2. Use manual installation features

     3. Help
     4. Quit

Install 1
Installing new software.

No subsystems selected for installation or removal.
Done.
Is there more software to install? n
sh: ./tagscript: not found
```

You can ignore the above error. It's harmless.

#

Now we're going to re-run *clinst*and watch what happens this time. As you will see, everything works just fine.

```
#./clinst -c 3_3_PI -h client_2 client

Client tree = /d/diskless/PI/client_2, shared tree = /d/diskless/PI/3_3_PI
Enter confirmation (y/Y) :y
Create 1 swap file ........

Ready to install software.

  Choose an item, then press
            :

     1. Automatically install software
     2. Use manual installation features

     3. Help
     4. Quit

Install 1
Installing new software.

Reading product descriptors from /d/diskless/PI/3_3_PI/usr/lib/inst:

    eoe1        4D1-3.3 Execution Only Environment (part 1)
    eoe2        4D1-3.3 Execution Only Environment (part 2)
    nfs         4D1-3.3 Network File System
    diag        Diagnostics
    maint1      Maint1 Product 4D1-3.3.2
    maint2      Maint2 Product 4D1-3.3.2
```

```
Computing disk space changes:


Installing new versions of selected maint1.eoe1_sw subsystems
Removing old versions of selected maint1.eoe1_man subsystems

Installing new versions of selected maint1.eoe1_man subsystems
Removing old versions of selected maint2.eoe2_sw subsystems

Installing new versions of selected maint2.eoe2_sw subsystems
Removing old versions of selected maint2.nfs_sw subsystems

Installing new versions of selected maint2.nfs_sw subsystems
Done.
Is there more software to install? n
#
```

We are now finished building a new client with 3.3.2 updates in it.

Section 8.2 — Booting the Diskless Client.

We have actually gone through this earlier, but for convenience sake, we'll repeat the SGI standard procedure. Once you have finished building the client tree, run */etc/exportfs* on the server just to be certain that the NFS files for the client have been exported. Then go to the client and power it up. Go into monitor mode (number 5) and type *printenv*. You should see several lines of NVRAM variables. We are interested in only three: *netaddr*, *bootfile* and *diskless*. Write down the values for *netaddr* and *bootfile* if you are temporarily using a system as diskless. Assuming that the Internet address for the client_1 is 192.10.26.10 and the server's hostname is "sandy," type the following:

```
>>setenv    diskless 1
>>setenv    netaddr  192.10.26.10
>>setenv    bootfile bootp()sandy:/usr/etc/boot/client_1/unix
>>printenv
```

Once you have compared what you typed with what you thought you typed and found that they agree, type *init* and boot the system. If your system's ethernet transceiver has LEDs you should see a lot of ethernet activity which lasts between three and ten minutes, depending on other ethernet activity. Eventually, you should see the SGI copyright notice appear and after a minute or so, additional messages will appear. That, if everything is okay, is all there is to it. Once you have setup the NVRAM, you shouldn't have to reset it until you want to do something new with the system, such as change it's Internet address.

Section 8.2.1 — Booting Problems and What to do About Them.

Sometimes things don't go right and you can't get the client to boot. Assuming there are no network problems (and that is a fifty page paper in itself), the usual cause is the guest account in `/etc/passwd` has been modified to have a password. Unfortuneately TFTP requires that the guest account be open. If it isn't, then TFTP will not be able transmit the bootfile to your server. Remove the password, if there is one and try again.

Even assuming that you did everything correctly, you may still have a problem. The most common problem occurs when you are trying to use an older system as a diskless workstation. In the case of the 4D/20 and 4D/25, you may happen to have an out-of-date EPROM. It has to be REV E or later. This is part number **070-8003-007(Rev E)**. You can test for this easily by setting the NVRAM *diskless* parameter

as described above and then powering the system off. Then turn the power back on and go back into monitor mode and do another *printenv*. If the *diskless* parameter disappeared, you most likely have an out-of-date EPROM. You can find our what version of the EPROM your PI has by typing *version* while still in the monitor. It will reply with something like the following:

```
>>version
 PROM Monitor Version 3.2 Rev E  (and then date and time)
```

In the case of the ASD systems, they have never been officially released as diskless workstations. However, the newer systems can nevertheless be used as diskless workstations, because their EPROMs support the *diskless* parameter. You can test an ASD system's EPROMs by going into monitor mode, setting *setenv diskless 1*, and then typing *init*. You will see the screen flash and then the boot menu reappear. Select the monitor again (i.e. 5) and type *printenv*. If the diskless parameter appears, and it is set to "1," your EPROMs supports diskless booting.

The correct EPROMs for IP5, IP7, and IP9 systems is hardware level "007." The part numbers are **070-0400-007** and **070-401-007**. It is marked on the label on top of the EPROMs. We suggest that you do not try to look for this unless you are a fully qualified hardware technician. If you are qualified, you'll know where to look. If you aren't, go find somebody who is because you can do many thousands of dollars worth of damage if you aren't careful.

You can test the version of ASD EPROMs the same way you can with the PI type systems. We know that the "005" hardware level does not support diskless, while the "007" level does. We are yet to test the "006," so we don't know if it supports diskless. You can identify "005" level with the monitor *version* command:

```
>>versions
PROM Monitor Version 4D1-4.0A (then time and date)
```

And you can identify the "007" level with the following:

```
>>versions
PROM Monitor Version 3.3 IP5 (then time and date)
```

As far as we can tell, no IP4 systems have EPROMs which supports diskless.

Even if you don't have the correct EPROM, you can still boot one of these older systems as a diskless workstation. The only problem is that you'll have to do the following ***Every*** time you boot it because the parameters will not be stored in NVRAM. First, set the *netaddr* boot parameter in the monitor. This should store in the NVRAM. If it doesn't, you have a very, very old system, and it doesn's support either BOOTP or TFTP as so can not be booted even with this procedure..

Next, you must go back into the monitor mode again and type the following three commands. You will have to do this every time you want boot your system as diskless. Obviously, you would use your sever's and client's host names in place of "server:" and "client_name."

```
>>setenv bootfile bootp()server:/etc/usr/boot/client_name/unix
>>setenv diskless 1
>>boot
```

The reason why you must do this is because the older EPROMs save only the first twenty or so characters of the *bootfile* parameter and nothing of the *diskless* parameter. Therefore, they must be reset each time.

Finally, you must use the *boot* command to go immediately into the booting phase. This way the values for the *bootfile* and *diskless* parameters are retained in RAM. This is critical because several `/etc` files use */etc/nvram* (see the man page) to read the *diskless 1* parameter to set up the file system while your kernel is booting. If you were to use the monitor's *init* or *exit* commands, the EPROM code would try to store the values into NVRAM, fail because it doesn't know how to do it, then clear the RAM. This, naturally, keeps the system from booting correctly. Therefor, you must use *boot*.

Section 8.2.2 — What to do When GET_BOOTP: WHOAMI Fails.

There is one well-known bug which, while irksome, is not really much of a problem once you know about it. It's the "**GET_BOOTP: WHOAMI FAILED addr 04c**" error that might show up a minute or two after the diskless workstation displays the SGI copyright notice while booting. Ninety per cent of the time, this happens because your server isn't running NIS (once known as *yellow pages*) and you do not have *domainname* defined on your server. The problem is that for whatever reason, the supplier of Sun Remote Procedure Calls decided to require that a *domainname* be defined in the code that handles `bootparams.rpc`. The problem is not simple to fix because the changes needed to get around this problem cause far worse problems elsewhere. Thus, what needs to happen is that the supplier of Sun Remote Procedure Calls fixes this problem in their source so that everyone using their code (i.e. just about everybody running NFS on unix) will remain compatible with one another. Until then, we will have to learn to live with this problem. Fortunately, the work around is trivial: Define the server's *domainname* to something—anything. If your diskless workstation has this problem, go the the server and type the following:

```
# domainname                inquire what is domainname
                            if the answer is a null string
# domainname junk           make it "junk"
```

Now reboot the client. The problem is by-passed. You should put the work-around in the server's `/etc/config/network.local` file so that the domainname is automatically named every time you reboot the server. You'll have to read the man page for `network` to find out all the details.

There is a second reason why you might get the WHOAMI problem. And that is either you or some hacker who thought he knew better went into the server's `/etc/bootparams` file and changed something. Change it back to the way *clinst* built it. And if you didn't keep a copy of the original `/etc/bootparams` file as *clinst* built it, you'll have to start from scratch.

Section 8.3 — What to do When Your Clients "Hang."

The reason why the word *hang* is in quotation marks, is because your client most likely isn't hung at all. Some minimal system configurations, particularly 8 MB clients, look like they are hung because they are actually swapping through the ethernet in a very painfully slow manner. This phenomenon, called a "swap storm," is difficult to trigger, but if your application is loading large files, such as bit-mapped graphics files, you can cause the poor system to spend as long a five minutes to swap. There are several fixes to this problem, but before we get into them, let's look at how you to tell if your system is really hung or just simply going through a very painfully slow swap through the ethernet.

- First, simply move the mouse. If your cursor moves even though nothing else seems to happen, your system isn't hung. The graphics engine runs fairly independent of system software and so is usually able to react if the system is still functioning.

- Second, make it a habit to run *gr_osveiw* on systems that have appeared to hang. If, while they are "hung," *gr_osveiw* shows that there is swapping going on (its the yellow segment of the I/O bar), then you are most likely in a swap storm. Note, however, that here is no guarantee that *gr_osview* will show swapping activity at this point. We have seen it happen so fast that *gr_osview* doesn't have time to update before the swap storm hits.

- Third, simply wait. Ususally, this pernicious behavior will cure itself in a few minutes. But, once again, we have seen it take as long as a half-hour. The surest test for this problem is to just let the system sit for an hour. If after that time it hasn't recovered, you probably have a hardware problem. If, on the other hand, it has recovered, then you know for certain that you were the victim of a swap storm.

What to do? Either stop doing whatever it is you're doing to trigger a swap storm or add more hardware. While additional main memory will decrease the severity of a swap storm, there is no guarantee. We have observed swap storms on diskless clients with as much a 32 MB. Admittedly, we trigger such activity by grossly overloading memory with monstrously large bit-mapped RGB images, but we succeeded. The best answer is to add a local swapping disk as described the next section.

Section 8.4 — Adding Local Disks to IRIX 3.3 Diskless Workstations

One caveat before we start on this section, what follows is for IRIX 3.3. The procedure for IRIX 3.2 is a more complicated and what follows will not work with IRIX 3.2.

We are going to assume that you have followed our advice when you built the share tree, because if you had, you would have set the "LOCALDISK" variable to "yes." If you did not, you built the share tree unix kernel without `efs.o`. Since this paper is meant for the system administrator, we will not go through the "guru tricks of the trade" so you can to use *lboot* to fix the problem. In fact, we strongly urge you not to even try because the chances of you making a mistake are extremely high.

Instead, we are going to advise you to erase the share tree and all of the client trees associated with it with *clinst* (read the man page and use the " *-d*" switch). Then, after fixing the appropriate `clinst.dat` file, rebuild the share tree and all of the client trees again. Quite frankly, we ourselves use this procedure when fixing this problem because it is actually faster than to trying to figure what we forgot to do while trying to take a shortcut. It is an old and wise saying which notes: "Short cuts often lead over quicksand."

Section 8.4.1 — IRIX 3.3 Disk Partitions.

Before we get into the details of installing a disk on a diskless workstation, we are going to have to go into how SGI partitions the disk drives. The disks come preformatted and partitioned. As you know, there is a section called "root," another where `/usr` is located, and finally, the `swap` space. These areas of the disk are allocated by consecutive cylinders, and they are called partitions. Normally they are allocated as follows:

Partition	Usage
0	/ (i.e. root)
1	`swap` space
6	`/usr`
7	sum of 0,1 and 6; i.e. all of the disk.

Now run *dvhtool* and dump the partition sizes. This will work even though you are running a diskless system. The *dvhtool* utility only cares that the hardware is in place. Assuming that we've mounted the disk as the first drive on the SCSI interface, type the following. What you type is shown in **bold**.

```
# /etc/dvhtool /dev/dsk/dks0d1vh

Command? (read, vd, pt, dp, write, bootfile, or quit): pt
Command? (part nblks 1stblk type, or 1) 1

current contents:
        part    n_blks  1st_blk type
        0:      32706   2484    efs
        1:      102258  35190   raw
        6:      503424  137448  efs
        7:      638388  2484    efs
        8:      2484    0       volhdr
        10:     640872  0       volume
Command? (part nblks 1stblk type, or 1) [carriage return]

Command? (read, vd, pt, dp, write, bootfile, or quit): quit
#
```

What you see above is the volume header. Note that "vh" is actually partition 8 and "vol"is partition 10. We are interested the `swap` partition, partition 1. Write down the entry for "n_blks" for this partition. In this case it's 102258. Be very careful about this because if there is valuable data on that disk, you can zap it if you make a mistake on this bit of information.

Now shut down the client. Do this normally with *init 0* and then power the client off. It ***Must not be active*** during the next few steps. Move to the server and get into `/usr/etc/boot`. Then edit the appropriate `clinst.dat` file for the share tree that the client is attached to. We are going to remove the swap space on the server, and we are going to use *clinst* to do it. Please do not try to find a short cut, because if you do, the chances of blowing it are quite good. Use *clinst* and be safe. Besides, it only takes a few minutes anyhow.

First, edit the `clinst.dat` file used to build the client tree.

```
#cd /usr/etc/boot
#vi 3_3_PI.dat
```

(edit the file and change the SWAPSIZE *variable to* SWAPSIZE="0")

Now run clinst as though you were building the client from scratch.

```
 ./clinst -c 3_3_PI -h client_1 client
Client tree = /d/diskless/PI/client_1, shared tree = /d/diskless/PI/3_3_PI

Enter confirmation (y/Y) :y
Removing swap file ........

Ready to install software.

 Choose an item, then press
          :
     1. Automatically install software
     2. Use manual installation features
     3. Help
     4. Quit

Install 4
```

Next we have to edit the client's `/etc/brc` file. This is best done from the server and is done as follows:

```
#cd /d/diskless/PI/client_1/etc
#vi brc
```

Now add the five lines that are in bold after line 13. We have also added comments for clarity. It should look like the following:

```
#! /bin/sh
#ident "$Revision: 1.17 $"
#
# This script is responsible for initializing the mounted filesystem table
# kept in /etc/mtab.  It also creates /etc/fstab if none exists.
#

if [ "'/etc/nvram diskless 2 /dev/null'" -eq 1 ] ; then
     /etc/mtab
    /etc/mount -f /
    if [ -d /share ]; then
            /etc/mount -f /share
    fi
# the last parameter on FIRST /etc/swap entry is "n_blks" for the swap partition.
# We used 102258 in this example. Use whatever you got
# when you ran dvhtool ON YOUR DISK.
   /etc/swap -a /dev/dsk/dks0d1s1 0 102258
   /etc/swap -d /dev/swap 0
    if [ -d /swap ]; then
            /etc/mount -f /swap
    fi
else
    rootdev=/dev/root
    usrdev=/dev/usr
    if [ ! -f /etc/fstab ] ; then
            echo "$rootdev /\n$usrdev /usr" | setmnt -f /etc/fstab
    fi
    echo "$rootdev /" | setmnt
fi
```

Now check your numbers in the above. Be very sure that you have them right because when you boot your client, it's going to use those blocks for `swap`, and if you inadvertently overlapped `/usr`, it can be over-written.

Now power on your "super-diskless" workstation!

Section 8.5 — Trouble Shooting.

We tried to indicate throughout this paper where you can wrong each step of the way and how to avoid each pit fall. Hopefully, you took the care to read this entire paper before rushing into the computer room and building your very own network of "super-diskless" workstations. Knowing human nature, some of you tried to out-guess the system. Congratulations to those of you who have succeeded. And to those of you reading this section because you must, don't feel bad. You can't grow unless you try to go beyond your present capabilities. It only natural to stub your toe now and then.

In the case of building diskless share and client trees, some of you fell victim to one of the most alluring traps of them all: You read *clinst* which is nothing more than a shell script. And after doing so, you thought you knew all there was to know about building share and client trees. This in turn caused some of you to "wing it" and get into all sorts of interesting problems. We know, because we get the phone calls.

However, don't consider any of this as a censure. It isn't. In fact, we encourage you to keep on trying, but only ask that you do so in way that doesn't cause panic and mayhem on the part of all the poor folks trying to get work done on their diskless systems. That is to say, use *clinst* and follow instructions for the production systems. Keep your experiments private and unobtrusive.

Most of all we encourage you to continue studying *clinst*. The software engineer who wrote *clinst* is, in our opinion, one of the best there is. It is an excellent example of how shell scripts should be done. Unfortunately, there are very subtle meanings to many of the shell variables which may not be obvious to the casual reader. This was not intentional, but just the natural outcome of a very sophisticated, extremely complex bit of work being made to look easy by a very expert engineer. It just like the way the ballerina makes floating over the stage on her tip toes look effortless: *clinst* makes building diskless share and client trees seem easy. There is much you can learn from reading this very nice piece of work.

Now for those of you have already gotten into trouble. What to do? Well, we have a little list of questions that we run through when the phone rings. Nine times out of ten, one of these gets to the root of the problem.

- First — are you absolutely sure that you used released SGI software? More than one problem has been traced to someone using unreleased software. Unless the tapes have SGI labels on them, be suspicious of them. There is one version of released SGI software which does not work for building diskless share trees. This is IRIX version 3.3.1. A simple mistake inadvertently got inserted in this release of IRIX and it broke *clinst*. You can, however, use IRIX 3.3.0 or 3.3.2 with no difficulties.
- Second — Did you fake it and get caught? There is a surprisingly simple test for this: try to remove the client tree with *clinst*. Before doing anything, *clinst* checks that various tables and files are the still way it built them. If it finds anything amiss, *clinst* will flatly refuse to remove that client tree. If it does, either you or one of your colleagues have be mucking with the diskless workstation stuff. If this is the case, read Section 8.7. And just in case nobody has messed with the files, don't worry about *clinst* zapping your client tree; it asks for confirmation first. Just say "no."
- Third — did you carefully follow the instructions regarding the share tree? One happy soul tried to attach a "almost-the-same" client on a very similar share tree by changing the GFXBOARD parameter in the `clinst.dat` for the share tree while he was building the client tree. He thought he'd save the space of a second share tree. He ended up with a mess that require him to start from scratch. ***Never, never*** change CPUBOARD, MACH, or GFXBOARD parameters of a share tree's `clinst.dat` file once you've build the share tree. These three parameters must remain the same in the share tree and ***all*** client tress build from that share tree.
- Fourth — Did you try to run *lboot* on either the share tree or any of the client trees? If you did and you are having problems, please don't call us — just erase the entire mess (see Section 8.7) and start from scratch. Building diskless workstations is very easy and effortless with *clinst*, so why not use it? And did you save any time by taking a short cut?
- Fifth — Did you use USESRVROOT="yes" in the clinst.dat file? Don't! This feature dosen't work in all situations, and to avoid problems, it was withdrawn in IRIX 3.3.2. If you are using it and experiencing troubles, rebuild your share tree without it and then try again.

- Sixth — Do the versions of IRIX on the server match those on the share trees? While you can have any version of IRIX 3.3 on the server, only 3.3.0 and 3.3.2 (and later versions) can be used in the share tree. However, do not try to run any combination of IRIX 3.3 and 3.2. It may work, but we strongly advise against it. As a rule, the first two digits (i.e. 3.3) must match on the server and the share tree.

Section 8.6 — Before You Call.

We at SGI are very proud of our products and stand firmly behind them. Our support organizations are interested in your problems and are eager to help. From time to time, we find bugs and so might you. If you are having problems with your SGI diskless workstation, we are more than willing to help. However, to do your part, we ask that you do some things before you call us about a mysterious problem regarding diskless workstations.

- First, read Section 8.5 and be able to say in bottom of your heart that you did not try do any of these things.
- Second, print out the server's `/etc/exports` and `/etc/bootparams` files. We mean the server's, not the share tree.
- Third, print out all of the `clinst.dat` files in the server's `/usr/etc/boot` directory. There should be one for each type share tree. Also label them so we know which is which.
- Fourth, make a list of all of the diskless clients you have on the network complete with model , graphics board type, and host name.

Assuming you have a support contract with SGI, call the hot line. Their personnel will work through most the problems over the phone. While doing so, they may request the information listed above, so have it handy. If they have to escalate your problem to software engineering, they will ask that you either mail or fax those listings to SGI so that they can be turned over to engineering.

Section 8.7 — When All Else Fails.

Hopefully, very few of you will ever read this section except out of academic interest. However, there remains the one remaining problem of cleaning up the mess. Assuming *clinst* has decided that it's not going to get involved in clearing out the mess, you will have to the clean up manually. What follows is the computer science equivalent of nuclear war. Do not do this if you can get *clinst* to talk to you. This is, by it's nature, only to be done when all else fails.

- First — Zap the share, swap and client trees. In our example you would *cd* into the "root" directory of each class of diskless trees. For example, `/d/diskless/PI`. Then use *rm -rf ** to wipe it all out.
- Second — Clean up the server's `/etc/exports` files. The share and client tree entries are readily identifiable. Delete them very carefully, one-by-one. Some other stuff may be in there, so be careful that you don't delete anything you want.
- Third — Clean out the unwanted entries in the server's `/etc/bootparams` file. If there are any other share trees that you want to retain, be very careful. This file is very delicate and we insist that you never try to edit an entry. However, since the entire entry for a single client or share tree is always a single line, you can easily delete that line with an editor.

THE END

With direction :

```
>> boot dkip(0,0,8)fx      # MBC or SBC with ESDI
   boot dksc(0,1,8)fx      # SBC with SCSI
```

With a second drive :

```
>> boot dkip(0,1,8)fx      # MBC or SBC with ESDI
   boot dksc(0,2,8)fx      # SBC with SCSI
```

With a remote workstation's disk drive over the network:

```
>> boot -f bootp()server:/stand/fx
```

Note : make sure that netaddr environment variable is set, and that the remote system has the bootp daemon running.

With local Product Distribution tape:

```
>> boot -f tpqic()fx.R2300       # MBC with VME-QIC
   boot -f tpqic()fx.IP4         # SBC with VME-QIC
   boot -f tpsc(0,7,0)fx.IP4     # SBC with SCSI
```

With a remote tape drive :

```
>> boot -f bootp()two:/dev/tape(fx[.IP4])
```

*** To exit sash back to prom monitor

<cntr d> or <cntl c>

*** Copying miniroot into swap space

from local Product Distribution tape :

```
Sash: cp -b 4k tpqic(,,1) dkip(0,0,1)    # MBC/SBC w/ESDI disk & vme tape
      cp -b 4k tpqic(,,1) dksc(0,1,1)    # SBC w/vme tape  & SCSI disk
      cp -b 4k tpsc(0,7,1) dkip(0,0,1)   # SBC w/SCSI tape & ESDI disk
      cp -b 4k tpsc(0,7,1) dksc(0,1,1)   # SBC w/SCSI tape & SCSI disk
```

from a remote server's tape drive : Execute all three commands in this order

```
Sash: cp -b 4k bootp()server:/dev/tape null()
      cp -b 4k bootp()server:/dev/nrtape null()
      cp -b 4k bootp()server:/dev/nrtape dkip(0,0,1)
```

Note : if the disk drive type is SCSI, then replace last line with dksc(0,1,1)!!! Also, make sure that netaddr environment variable is set, and that the remote system has the bootp daemon running.

*** Booting the miniroot

```
Sash: boot -f dkip(0,0,1)unix.R2300     # MBC with ESDI disk
      boot -f dkip(0,0,1)unix.IP4       # SBC with ESDI disk
      boot -f dksc(0,1,1)unix.IP4       # SBC with SCSI disk
```

4D System Administration

Reference Books

*** Operating Systems

- Operating Systems - Design & Implementation	Tanenbaum
- THE XINU APPROACH	Comer
- OPERATING SYSTEM ELEMENTS	Calingaert

*** Unix

- EXPLORING THE UNIX SYSTEM	Kochan/Wood
- Unix The Complete Reference	Coffin
- INTRODUCING UNIX SYSTEM V	Morgan/McGilton
- THE UNIX PROGRAMMING ENVIRONMENT	Kernighan/Pike
- Unix Programmer's Manual Vol I-V	AT&T

*** C

- PROGRAMMING IN C rev I & II	Kochan
- C : The Complete Reference	Schildt
- TOPICS IN C PROGRAMMING	Kochan/Wood
- PROGRAMMING IN ANSI C	Kochan
- Advanced C Tips & Techniques	Anderson & Anderson
- Advanced Unix Programming	Rochkind
- Advanced C Programming for Displays	Rochkind
- THE C PROGRAMMING LANGUAGE rev I & II	Kernighan/Ritchie

*** Shell Programming

- UNIX SHELL PROGRAMMING	Kochan/Wood
- THE UNIX C SHELL FIELD GUIDE	Anderson/Anderson

*** Data Communications and Networking

- DATA AND COMPUTER COMMUNICATIONS	Stallings
- Computer Networks	Tanenbaum
- UNIX Communications	The Waite Group
- INTERNETWORKING WITH TCP/IP	Comer
- Operating Systems Vol II Internetworking with Xinu	Comer
- C Programmer's Guide to Serial Communications	Campbell

*** Miscellaneous

- UNIX SYSTEM ADMINISTRATION	Fiedler/Hunter w/ Kochan/Wood
- UNIX TEXT PROCESSING	Dougherty/O'Reilly w/ Kochan/Wood
- UNIX SYSTEM SECURITY	Wood/Kochan
- UNIX DEVICE DRIVERS	Egan/Texiera
- Postscript Tutorial/cookbook(blue)	Adobe Systems Inc.
- Postscript Reference Manual(red)	Adobe Systems Inc.

Disk and Tape drives supported

Device	*SBC*	*MBC*
Tape	**SCSI & VME-QIC**	**VME-QIC**
Disk	**SCSI & ESDI**	**ESDI**

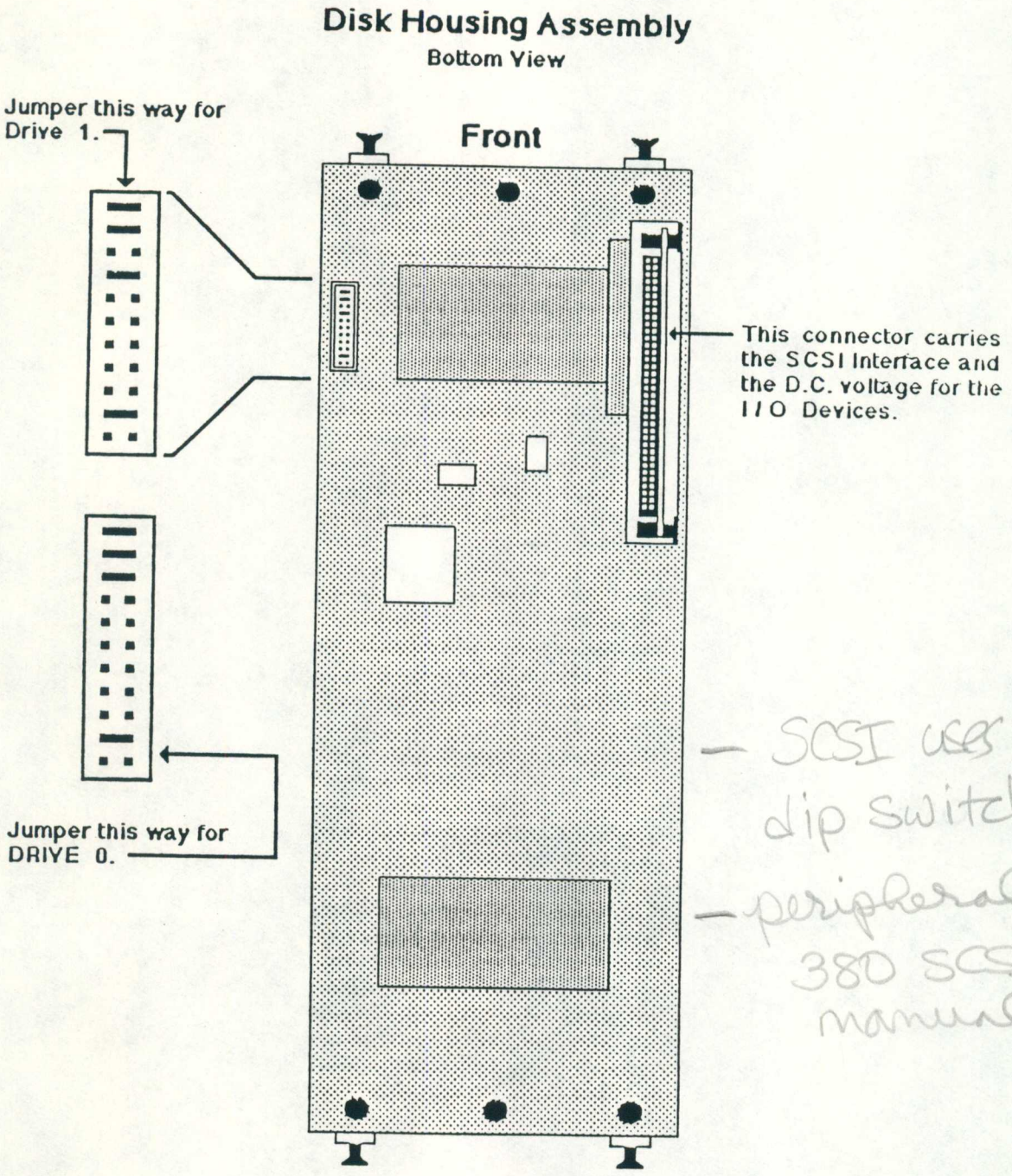
Disk Sizes

Disk Type	Disk Size	
	170	380
SCSI	yes	no
ESDI	yes	yes

1.2 gigabyte announced

DIAGRAM 17: Disk Address jumpers

ESDI only



- SCSI use dip switches
- peripherals 380 SCSI manual

Root filesystem (/root)
Absolute position in tree: /
Disk partition: /dev/dsk/ips0d0s0
Pseudonym: /dev/root

= a file

= a directory