The subsystems that address these topics are interesting systems in their own right and are case studies of managing complexity. Typically, these subsystems are internally structured as client/service systems, applying the concept of this chapter recursively. The next two sections provide two case studies of real-world client/service systems and also illustrate the need for the topics addressed in the subsequent chapters.

4.4 CASE STUDY: THE INTERNET DOMAIN NAME SYSTEM (DNS)

The Internet Domain Name System (DNS) provides an excellent case study of both a client/service application and a successful implementation of a naming scheme, in this case for naming of Internet computers and services. Although designed for that specific application, DNS is actually a general-purpose name management and name resolution system that hierarchically distributes the management of names among different naming authorities and also hierarchically distributes the job of resolving names to different name servers. Its design allows it to respond rapidly to requests for name resolution and to scale up to extremely large numbers of stored records and numbers of requests. It is also quite resilient, in the sense that it provides continued, accurate responses in the face of many kinds of network and server failures.

The primary use for DNS is to associate user-friendly character-string names, called *domain names*, with machine-oriented binary identifiers for network attachment points, called *Internet addresses*. Domain names are hierarchically structured, the term *domain* being used in a general way in DNS: it is simply a set of one or more names that have the same hierarchical ancestor. This convention means that hierarchical regions can be domains, but it also means that the personal computer on your desk is a domain with just one member. In consequence, although the phrase "domain name" suggests the name of a hierarchical region, every name resolved by DNS is called a domain name, whether it is the name of a hierarchical region or the name of a single attachment point. Because domains typically correspond to administrative organizations, they also are the unit of delegation of name assignment, using exactly the hierarchical naming scheme described in Section 3.1.4.

For our purposes, the basic interface to DNS is quite simple:

value ← DNS_RESOLVE (*domain_name*)

This interface omits the context argument from the standard name-resolving interface of the naming model of Section 2.2.1 because there is just a single, universal, default context for resolving all Internet domain names, and the reference to that one context is built into DNS_RESOLVE as a configuration parameter.

In the usual DNS implementation, binding is not accomplished by invoking BIND and UNBIND procedures as suggested by our naming model, but rather by using a text editor or database generator to create and manage tables of bindings. These tables are then loaded into DNS servers by some behind-the-scenes method as often as their managers deem necessary. One consequence of this design is that changes to DNS

bindings don't often occur within seconds of the time you request them; instead, they typically take hours.

Domain names are path names, with components separated by periods (called *dots,* particularly when reading domain names aloud) and with the least significant component coming first. Three typical domain names are

```
ginger.cse.pedantic.edu    ginger.scholarly.edu    ginger.com
```

DNS allows both relative and absolute path names. Absolute path names are supposed to be distinguished by the presence of a trailing dot. In human interfaces the trailing dot rarely appears; instead, DNS_RESOLVE applies a simple form of multiple lookup. When presented with a relative path name, DNS_RESOLVE first tries appending a default context, supplied by a locally set configuration parameter. If the resulting extended name fails to resolve, DNS_RESOLVE tries again, this time appending just a trailing dot to the originally presented name. Thus, for example, if one presents DNS_RESOLVE with the apparently relative path name `"ginger.com"`, and the default context is `"pedantic.edu."`, DNS_RESOLVE will first try to resolve the absolute path name `"ginger.com.pedantic.edu."`. If that attempt leads to a NOT-FOUND result, it will then try to resolve the absolute path name `"ginger.com."`

4.4.1 Name Resolution in DNS

DNS name resolution might have been designed in at least three ways:

1. *The telephone book model:* Give each network user a copy of a file that contains a list of every domain name and its associated Internet address. This scheme has a severe problem: to cover the entire Internet, the size of the file would be proportional to the number of network users, and updating it would require delivering a new copy to every user. Because the frequency of update tends to be proportional to the number of domain names listed in the file, the volume of network traffic required to keep it up to date would grow with the cube of the number of domain names. This scheme was used for nearly 20 years in the Internet, was found wanting, and was replaced with DNS in the late 1980s.

2. *The central directory service model:* Place the file on a single well-connected server somewhere in the network and provide a protocol to ask it to resolve names. This scheme would make update easy, but with growth in the number of users its designer would have to adopt increasingly complex strategies to keep it from becoming both a performance bottleneck and a potential source of massive failure. There is yet another problem: whoever controls the central server is by default in charge of all name assignment. This design does not cater well to delegation of responsibility in assignment of domain names.

3. *The distributed directory service model.* The idea is to have many servers, each of which is responsible for resolving some subset of domain names, and a protocol for finding a server that can resolve any particular name. As we shall see in the following descriptions, this model can provide delegation and respond

to increases in scale while maintaining reliability and performance. For those reasons, DNS uses this model.

With the distributed directory service model, the operation of every name server is the same: a server maintains a set of name records, each of which binds a domain name to an Internet address. When a client sends a request for a name resolution, the name server looks through the collection of domain names for which it is responsible, and if it finds a name record, it returns that record as its response. If it does not find the requested name, it looks through a separate set of referral records. Each referral record binds a hierarchical region of the DNS name space to some other name server that can help resolve names in that region of the naming hierarchy. Starting with the most significant component of the requested domain name, the server searches through referral records for the one that matches the most components, and it returns that referral record. If nothing matches, DNS cannot resolve the original name, so it returns a "no such domain" response.

The referral architecture of DNS, though conceptually simple, has a number of elaborations that enhance its performance, scalability, and robustness. We begin with an example of its operation in a simple case, and we later add some of the enhancements. The dashed lines in Figure 4.10 illustrate the operation of DNS when the client computer named `ginger.cse.pedantic.edu`, in the lower left corner, tries to resolve the domain name `ginger.Scholarly.edu`. The first step, shown as request #1, is that `DNS_RESOLVE` sends that domain name to a *root name server*, whose Internet address it somehow knows. Section 4.4.4 explains how `DNS_RESOLVE` discovers that address.

The root name server matches the name in the request with the subset of domain names it knows about, starting with the most significant component of the requested domain name (in this example, `edu`). In this example, the root name server discovers that it has a referral record for the domain `edu`, so it responds with a referral, saying, in this example, "There is a name server for a domain named `edu`. The name record for that name server binds the name `names.edu.` to Internet address 192.14.71.191." This response illustrates that name servers, like any other servers, have both domain names and Internet addresses. Usually, the domain name of a name server gives some clue about what domain names it serves, but there is no necessary correspondence. Responding with a complete name record provides more information than the client really needs (the client usually doesn't care about the name of the name server), but it allows all responses from a name server to be uniform. Because the name server's domain name isn't significant and to reduce clutter in Figure 4.10, that figure omits it in the illustrated response.

When the client's `DNS_RESOLVE` receives this response, it immediately resends the same name resolution request, but this time it directs the request (request 2 in the figure) to the name server located at the Internet address mentioned in response number 1. That name server matches the requested path name with the set of domain names it knows about, again starting with the most significant component. In this case, it finds a match for the name `Scholarly.edu.` in a referral record. It thus sends back a response saying, "There is a name server for a domain named `Scholarly.edu`. The

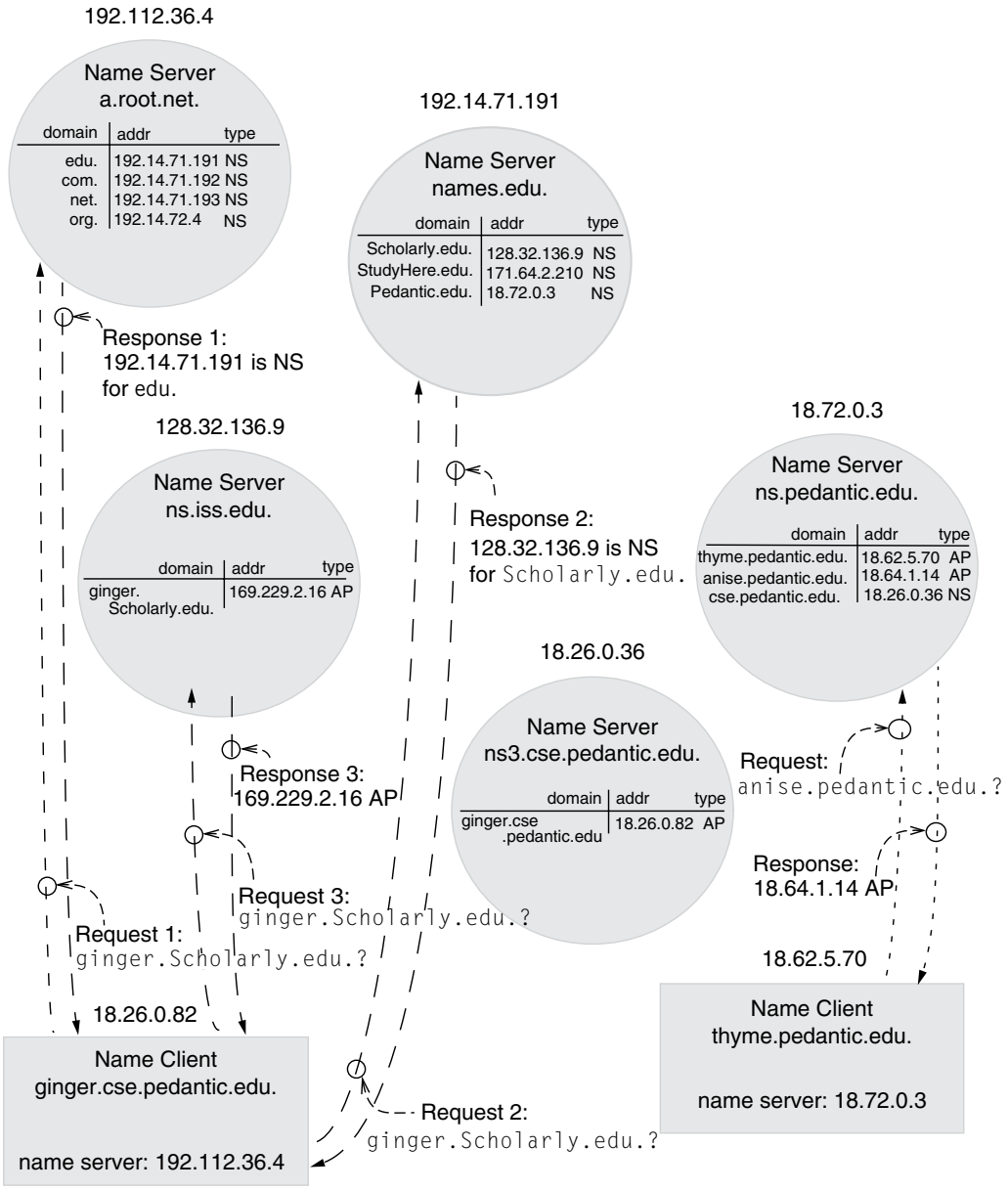

FIGURE 4.10

Structure and operation of the Internet Domain Name System. In this figure, each circle represents a name server, and each rectangle is a name client. The type NS in a table or in a response means that this is a referral to another name server, while the type AP in a table or a response means that this is an Internet address. The dashed lines show the paths of the three requests made by the name client in the lower left corner to resolve the name `ginger.Scholarly.edu`, starting with the root name server. The dotted lines show resolution of a request of the name client in the lower right corner to resolve `anise.pedantic.edu` starting with a local name server.

name record for that name server binds the name ns.iss.edu. to Internet address 128.32.136.9." The illustration again omits the domain name of the name server.

This sequence repeats for each component of the original path name, until DNS_RESOLVE finally reaches a name server that has the name record for ginger.Scholarly.edu. That name server sends back a response saying, "The name record for ginger.Scholarly.edu. binds that name to Internet address 169.229.2.16." This being the answer to the original query, DNS_RESOLVE returns this result to its caller, which can go on to initiate an exchange of messages with its intended target.

The server that holds either a name record or a referral record for a domain name is known as the *authoritative name server* for that domain name. In our example, the name server ns3.cse.pedantic.edu. is authoritative for the ginger.cse.pedantic.edu. domain, as well as all other domain names that end with cse.pedantic.edu., and ns.iss.edu. is authoritative for the Scholarly.edu. domain. Since a name server does not hold the name record for its own name, a name server cannot be the authoritative name server for its own name. Instead, for example, the root name server is authoritative for the domain name edu., while the names.edu. name server is authoritative for all domain names that end in edu.

That is the basic model of DNS operation. Here are some elaborations in its operation, each of which helps make the system fast-responding, robust, and capable of growing to a large scale.

1. It is not actually necessary to send the initial request to the root name server. DNS_RESOLVE can send the request to *any* convenient name server whose Internet address it knows. The name server doesn't care where the request came from; it simply compares the requested domain name with the list of domain names for which it is responsible in order to see if it holds a record that can help. If it does, it answers the request. If it doesn't, it answers by returning a referral to a root name server. The ability to send any request to a local name server means that the common case in which the client, the name server, and the target domain name are all three in the same domain (e.g., pedantic.edu) can be handled swiftly with a single request/response interaction. (The dotted lines in the lower right corner of Figure 4.10 show an example, in which thyme.pedantic.edu. asks the name server for the pedantic.edu domain for the address of anise.pedantic.edu.) This feature also simplifies name discovery because all a client needs to know is the Internet address of any nearby name server. The first request to that nearby server for a distant name (in the current example, ginger.scholarly.edu) will return a referral to the Internet address of a root name server.

2. Some domain name servers offer what is (perhaps misleadingly) called *recursive* name service. If the name server does not hold a record for the requested name, rather than sending a referral response, the name server takes on the responsibility for resolving the name itself. It forwards the initial request to a root name server, then continues to follow the chain of responses to resolve the complete path name, and finally returns the desired name record to its client. By

itself, this feature seems merely to simplify life for the client, but in conjunction with the next feature it provides a major performance enhancement.

3. Every name server is expected to maintain, in addition to its authoritative records, a cache of all name records it has heard about from other name servers. A server that provides recursive name service thus collects records that can greatly speed up future name resolution requests. If, for example, the name server for `cse.pedantic.edu` offers recursive service and it is asked to resolve the name `flower.cs.scholarly.edu`, in the course of doing so (assuming that it does not in turn request recursive service), its cache might acquire the following records:

```
edu                    refer to names.edu at 198.41.0.4
Scholarly.edu          refer to ns.iss.edu at 128.32.25.19
cs.Scholarly.edu       refer to cs.Scholarly.edu at 128.32.247.24
flower.cs.Scholarly.edu          Internet address is 128.32.247.29
```

Now, when this name server receives, for example, the request to resolve the name `psych.Scholarly.edu`, it will discover the record for the domain `Scholarly.edu` in the cache and it will be able to quickly resolve the name by forwarding the initial request directly to the corresponding name server.

A cache holds a duplicate copy, which may go out of date if someone changes the authoritative name record. On the basis that changes of existing name bindings are relatively infrequent in the Domain Name System and that it is hard to keep track of all the caches to which a domain name record may have propagated, the DNS design does not call for explicit invalidation of changed entries. Instead, it uses expiration. That is, the naming authority for a DNS record marks each record that it sends out with an expiration period, which may range from seconds to months. A DNS cache manager is expected to discard entries that have passed their expiration period. The DNS cache manager provides a memory model that is called *eventual consistency,* a topic taken up in Chapter 10 [on-line].

4.4.2 Hierarchical Name Management

Domain names form a hierarchy, and the arrangement of name servers described above matches that hierarchy, thereby distributing the job of name resolution. The same hierarchy also distributes the job of managing the handing out of names, by distributing the responsibility of operating name servers. Distributing responsibility is one of the main virtues of the distributed directory service model.

The way this works is actually quite simple: whoever operates a name server can be a *naming authority,* which means that he or she may add authoritative records to that name server. Thus, at some point early in the evolution of the Internet, some Pedantic University network administrator deployed a name server for the domain `pedantic.edu` and convinced the administrator of the `edu` domain to install a binding for the domain name `pedantic.edu.` associated with the name and Internet

address of the pedantic.edu name server. Now, if Pedantic University wants to add a record, for example, for an Internet address that it wishes to name archimedes.pedantic.edu, its administrator can do so without asking permission of anyone else. A request to resolve the name archimedes.pedantic.edu can arrive at any domain name server in the Internet; that request will eventually arrive at the name server for the pedantic.edu domain, where it can be answered correctly. Similarly, a network administrator at the Institute for Scholarly Studies can install a name record for an Internet address named archimedes.Scholarly.edu on its own authority. Although both institutions have chosen the name archimedes for one of their computers, because the path names of the domains are distinct there was no need for their administrators to coordinate their name assignments. Put another way, their naming authorities can act independently.

Continuing this method of decentralization, any organization that manages a name server can create lower-level naming domains. For example, the Computer Science and Engineering Department of Pedantic University may have so many computers that it is convenient for the department to manage the names of those computers itself. All that is necessary is for the department to deploy a name server for a lower-level domain (named, for example, cse.pedantic.edu) and convince the administrator of the pedantic.edu domain to install a referral record for that name in its name server.

4.4.3 Other Features of DNS

To ensure high availability of name service, the DNS specification calls on every organization that runs a name service to arrange that there be at least two identical replica servers. This specification is important, especially at higher levels of the domain naming hierarchy, because most Internet activity uses domain names and inability to resolve a name component blocks reachability to all sites below that name component. Many organizations have three or four replicas of their name servers, and as of 2008 there were about 80 replicas of the root name server. Ideally, replicas should be attached to the network at places that are widely separated, so that there is some protection against local network and electric power outages. Again, the importance of separated attachment increases at higher levels of the naming hierarchy. Thus, the 80 replicas of the root name server are scattered around the world, but the three or four replicas of a typical organization's name server are more likely to be located within the campus of that organization. This arrangement ensures that, even if the campus is disconnected from the outside world, communication by name within the organization can still work. On the other hand, during such a disconnection, correspondents outside the organization cannot even verify that a name exists, for example, to validate an e-mail address. Therefore, a better arrangement might be to attach at least one of the organization's multiple replica name servers to another part of the Internet.

For the same reason that name servers need to be replicated, many network services also need to be replicated, so DNS allows the same name to be bound to several Internet addresses. In consequence, the *value* returned by DNS_RESOLVE can be a list of (presumably) equivalent Internet addresses. The client can choose which

Internet address to contact, based on order in the list, previous response times, a guess as to the distance to the attachment point, or any other criterion it might have available.

The design of DNS allows name service to be quite robust. In principle, the job of a DNS server is extremely simple: accept a request packet, search a table, and send a response packet. Its interface specification does not require it to maintain any connection state, or any other durable, changeable state; its only public interface is idempotent. The consequence is that a small, inexpensive personal computer can provide name service for a large organization, which encourages dedicating a computer to this service. A dedicated computer, in turn, tends to be more robust than one that supplies several diverse and unrelated network services. In addition a server with small, read-only tables can be designed so that when something such as a power failure happens, it can return to service quickly, perhaps even automatically. (Chapters 8 [on-line] and 9 [on-line] discuss how to design such a system.)

DNS also allows synonyms, in the form of indirect names. Synonyms are used conventionally to solve two distinct problems. For an example of the first problem, suppose that the Pedantic University Computer Science and Engineering Department has a computer whose Internet address is named `minehaha.cse.pedantic.edu`. This is a somewhat older and slower machine, but it is known to be very reliable. The department runs a World Wide Web server on this computer, but as its load increases the department knows that it will someday be necessary to move the Web server to a faster machine named `mississippi.cse.pedantic.edu`. Without synonyms, when the server moves, it would be necessary to inform everyone that there is a new name for the department's World Wide Web service. With synonyms, the laboratory can bind the indirect name `www.cse.pedantic.edu` to `minehaha.cse.pedantic.edu` and publicize the indirect name as the name of its Web site. When the time comes for `mississippi.cse.pedantic.edu` to take over the service, it can do so by simply having the manager of the `cse.pedantic.edu` domain change the binding of the indirect name. All those customers who have been using the name `www.cse.pedantic.edu` to get to the Web site will find that name continues to work correctly; they don't care that a different computer is now handling the job. As a general rule, the names of services can be expected to outlive their bindings to particular Internet addresses, and synonyms cater to this difference in lifetimes.

The second problem that synonyms can handle is to allow a single computer to appear to be in two widely different naming domains. For example, suppose that a geophysics group at the Institute of Scholarly Studies has developed a service to predict volcano eruptions but that organization doesn't actually have a computer suitable for running that service. It could arrange with a commercial vendor to run the service on a machine named, perhaps, `service-bureau.com` and then ask the manager of the Institute's name server to bind the indirect name `volcano.iss.edu` to `service-bureau.com`. The Institute could then advertise its service under the indirect name. If the commercial vendor raises its prices, it would be possible to move the service to a different vendor by simply rebinding the indirect name.

Because resolving a synonym requires an extra round-trip through DNS, and the basic name-to-Internet-address binding of DNS already provides a level of indirection, some network specialists recommend just manipulating name-to-Internet-address bindings to get the effect of synonyms.

4.4.4 Name Discovery in DNS

Name discovery comes up in at least three places in the Domain Name System: a client must discover the name of a nearby name server, a user must discover the domain name of a desired service, and the resolving system must discover an extension for unqualified domain names.

First, in order for DNS_RESOLVE to send a request to a name server, it needs to know the Internet address of that name server. DNS_RESOLVE finds this address in a configuration table. The real name-discovery question is how this address gets into the configuration table. In principle, this address would be the address of a root server, but as we have seen it can be the address of any existing name server. The most widely used approach is that when a computer first connects to a network it performs a name discovery broadcast to which the Internet service provider (ISP) responds by assigning the attacher an Internet address and also telling the attacher the Internet address of one or more name servers operated by or for the ISP. Another way to terminate name discovery is by direct communication with a local network manager, to obtain the address of a suitable name server, followed by configuring the answer into DNS_RESOLVE.

The second form of name discovery involves domain names themselves. If you wish to use the volcano prediction service at the Institute for Scholarly Studies, you need to know its name. Some chain of events that began with direct communication must occur. Typically, people learn of domain names via other network services, such as by e-mail, querying a search engine, reading postings in newsgroups or while surfing the Web, so the original direct communication may be long forgotten. But using each of those services requires knowing a domain name, so there must have been a direct communication at some earlier time. The purchaser of a personal computer is likely to find that it comes with a Web browser that has been preconfigured with domain names of the manufacturer's suggested World Wide Web query and directory services (as well as domain names of the manufacturer's support sites and other advertisers). Similarly, a new customer of an Internet service provider typically may, upon registering for service, be told the domain name of that ISP's Web site, which can then be used to discover names for many other services.

The third instance of name discovery concerns the extension that is used for unqualified domain names. Recall that the Domain Name System uses absolute path names, so if DNS_RESOLVE is presented with an unqualified name such as `library` it must somehow extend it, for example, to `library.pedantic.edu`. The default context used for extension is usually a configuration parameter of DNS_RESOLVE. The value of this parameter is typically chosen by the human user when initially setting up a computer, with an eye to minimizing typing for the most frequently used domain names.

4.4.5 Trustworthiness of DNS responses

A shortcoming of DNS is that, although it purports to provide authoritative name resolutions in its responses, it does not use protocols that allow authentication of those responses. As a result, it is possible (and, unfortunately, relatively easy) for an intruder to masquerade as a DNS server and send out mischievous or malevolent responses to name resolution requests.

Currently, the primary way of dealing with this problem is for the user of DNS to treat all of its responses as potentially unreliable hints and independently verify (using the terminology of Chapters 7 [on-line] and 11 [on-line] we would say "perform end-to-end authentication of") the identity of any system with which that user communicates. An alternative would be for DNS servers to use authentication protocols in communication with their clients. However, even if a DNS response is assuredly authentic, it still might not be accurate (for example, a DNS cache may hold out-of-date information, or a DNS administrator may have configured an incorrect name-to-address binding), so a careful user would still want to independently authenticate the identity of its correspondents.

Chapter 11 [on-line] describes protocols that can be used for authentication; there is an ongoing debate among network experts as to whether or how DNS should be upgraded to use such protocols.

The reader interested in learning more about DNS should explore the documents in the readings for DNS [Suggestions for Further Reading 4.3].

4.5 CASE STUDY: THE NETWORK FILE SYSTEM (NFS)

The Network File System (NFS), designed by Sun Microsystems, Inc. in the 1980s, is a client/service application that provides shared file storage for clients across a network. An NFS client grafts a remote file system onto the client's local file system name space and makes it behave like a local UNIX file system (see Section 2.5). Multiple clients can mount the same remote file system so that users can share files.

The need for NFS arose because of technology improvements. Before the 1980s, computers were so expensive that each one had to be shared among multiple users and each computer had a single file system. But a benefit of the economic pressure was that it allowed for easy collaboration because users could share files easily. In the early 1980s, it became economically feasible to build workstations, which allowed each engineer to have a private computer. But users still desired to have a shared file system for ease of collaboration. NFS provides exactly that: it allows a user at any workstation to use files stored on a shared server, a powerful workstation with local disks but often without a graphical display.

NFS also simplifies the management of a collection of workstations. Without NFS, a system administrator must manage each workstation and, for example, arrange for backups of each workstation's local disk. NFS allows for centralized management; for example, a system administrator needs to back up only the server's disks to archive the file system. In the 1980s, the setup also had a cost benefit: NFS allowed organizations