04_2422_ch04.qxd 4/9/08 1:09 PM Page 97

Chapter 4

Stalking the Upper-Layer Architecture

You have Telnet and FTP. What else do you need?

—Alex McKenzie, 1975

If the Virtual Terminal is in the presentation layer, where is the power supply?

—Al Reska, 1979

Introduction

Although there has been a strong interest in "the upper layers" over the years, as these quotes indicate, there have been differing misconceptions about them. Since the earliest days of the Net, the upper layers have been a bit mysterious and unclear. The early network software removed concerns of reliability and routing from the applications. And with only a couple of dozen hosts connected by what seemed high-speed lines, one had the illusion of having an environment for distributed computing. All of this sparked the imaginations of those involved to the possibilities as it continues to do today. It seemed that rather than merely an amorphous collection of applications, there ought to be some general structure for organizing upper-layer functions, as there was for the lower layers.

The lower layers had succumbed to organization much more quickly, at least on the surface. By 1975, it was fairly common to hear people talk about transport, network, data link, and physical layers. It wasn't until later that things got sufficiently complex that the rough edges of *that* model began to show. Even then, however, those four layers seemed to capture the idea that the lower two

04_2422_ch04.qxd 4/9/08 1:09 PM Page 98

layers were media dependent, and the upper (or middle) two were media independent and end to end.

The upper layers (above transport) were a different story. No obvious structuring or decomposition seemed to apply. Part of this was due both to the lack of applications and, in some sense, too many. For some time, the network did seem to be able to get by with just Telnet and FTP (mail was originally part of FTP). To find applications on a host, "well-known sockets" were used as a stopgap measure. (There were only three or four applications, so it wasn't really a big deal). It was recognized early that there was a need for a directory, and there were some proposals for one (Birrell et al., 1982). But beyond that, application protocols were very much "point products." Protocols were unique to the application. There did not seem to be much commonality from which one could create a general structure that was as effective as the one for the lower layers, or that easily accommodated the variety of applications and at the same time provided sufficient advantage to make it worthwhile. Unfortunately, early in the life of the Net, upper-layer development was squelched before the promise could be explored. As we will see, it is this event that most likely contributed most to the arrested development of the Internet and left it the stunted, unfinished demo we have today.

The OSI work made a stab at the problem and for a while looked like it was making progress. But although they were able to uncover elements of a general structure, that architecture suffered from early architectural missteps that made application designs cumbersome and from overgenerality that required complex implementations for even simple applications, but mainly it was the internal divisions that killed OSI.

Over the past few years, there has been an opportunity to consider what was learned from these experiences and what they tell us about the nature of the upper layers. Consequently, a much better understanding has surfaced based on a broad experience not only with a variety of applications, but also recognition of similarities between the upper and lower layers. It is now much clearer how the lower layers differ from the upper layers, what the uppers layers do and do not do, how many layers there are, and what goes where. This chapter attempts to bring together these disparate results and put them in a consistent framework. Are all the problems solved? Far from it, but having such a framework will provide a much clearer picture of how we can go forward and will allow new results to be put into a context that makes them more useful. Upper-layer naming and addressing will be considered in the next chapter; here we are concerned with working out the structure. What we are interested in is understanding the relation of "networking" to "applications" or to distributed applications. Networking is not "everything" and cannot be considered to include all of distributed computing. We need to understand how and where one leaves off

04_2422_ch04.qxd 4/9/08 1:09 PM Page 99

and the other starts and the relation of the two. We are not so much interested in being encyclopedic as considering what we have uncovered about "the upper layers" that is both right and wrong.

A Bit of History

The Upper Layer(s) of the ARPANET

It would be difficult to claim that the members of the early Network Working Group started with an idea of upper-layer architecture. Their focus was on building a network. Applications were primarily there to show that it worked! The group had its hands full with much more basic problems: How do you do anything useful with a widely different set of computer architectures? How do you connect systems to something that they were never intended to be connected to? (You made it look like a tape drive.) How do you get fairly complex network software into already resource tight systems? Just implementing the protocols was a major effort for any group. The early ARPANET had much more diversity in the systems connected to it than we see today. Just in the hardware, there were systems with all sorts of word lengths (16, 18, 24, 32, 48, 64, etc.) and at least two varieties of 36-bit words. There were at least a dozen different operating systems with widely disparate models for I/O, processes, file systems, protection, and so forth.[1] Or at least, they seemed to be very different. If there was an architectural direction, it was to provide over the Net access to each of these systems as if you were a local user. (Or as close as 56Kb lines would allow; and 50Kb seemed vast when most remote access was 110 or 300bps!). Furthermore, some of these systems were very tightly constrained on resources. The big number cruncher on the Net was an IBM 360/91, with an outrageous amount of memory: 4MB! (And the operating system was huge! It occupied half!) The general attitude was that an operating system ought to fit in a system with 16KB and still have plenty of room to do useful work. The primary purpose of these early applications was to make the hosts on the Net available for remote use. Thus, the first applications were fairly obvious: terminal

[1] Today, all these systems would be classed as "mainframes." But as one would expect, there were different distinctions then: A few machines were mainframes; number crunchers, systems to submit big batch computation jobs to. Some were timesharing systems whose users specialized in an area of research, in essence precursors to networked workstations. Some were access systems, dedicated to supporting users but provided no computation services themselves; they were early minicomputers. All of them would now fit in a corner of a cell phone.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 100

access, transfer files, and submit jobs for execution. But once it worked (and it worked well), our imaginations ran the gamut of what could be done with a resource-sharing network. But from all of these problems and these initial applications came some very important architectural results in the design of application protocols that still serve us well today, and as one would expect there were also a few missteps along the way.

Early Elegance: Telnet, FTP, and RJE

The early NWG concentrated on three basic upper-layer protocols: a terminal protocol, a file transfer protocol, and a remote job entry protocol. Let's consider each one in turn.

Telnet was the first virtual terminal protocol. The first Telnet was sufficient to demonstrate that the network was usable; but because it reflected so many of the terminal characteristics to the user, it was less than a satisfactory solution. The NWG met in late 1972 and drafted the "new" Telnet.[2] The experience with getting Telnet wrong the first time paid off. With the experience gained from the first attempt, the NWG had a better understanding of what a terminal protocol needed to do and the problems Telnet needed to solve and how to do it effectively. Most important, Telnet was not a remote login application, but a terminal-driver protocol. Remote login is an application built using Telnet (again taking the operating system perspective). The new Telnet protocol had several attributes that are unfortunately still rare in the design of protocols.

The designers of Telnet realized that it was not simply a protocol for connecting hosts to terminals, but it could also be used as a character-oriented IPC mechanism between distributed processes: in essence, middleware. In particular, it could be used (and was) to connect two applications that had been written to interact with humans. Telnet defined a canonical representation of a basic terminal, the *network virtual terminal* (NVT). The NVT (Figure 4-1) defined a rudimentary scroll-mode terminal with very few attributes.[3] The model for a Telnet connection consists of two NVTs connected back to back: The "keyboard" of one is connected to the "screen" of the other and vice versa. The Telnet protocol operates between the two NVTs. The terminal system and the host system convert their local representation into the NVT representation and convert the output of the NVT to their local representation.

---

[2] At this point, calling it "new Telnet" is a bit like calling the *Pont Neuf* the New Bridge.

[3] There was an incredibly wide variety of terminals (probably more than 50) on the market at the time running the gamut from electric typewriters printing on paper, to displays that mimicked the paper printers, to fairly complex storage terminals that handled forms and could display text with bold or reverse video. The NVT did only the simplest of these, although options were created for more complex ones.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 101

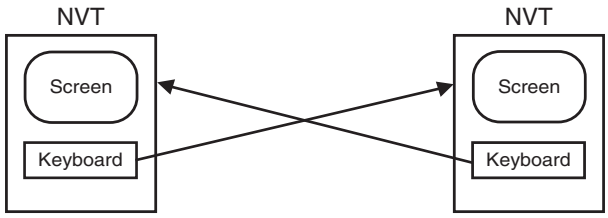

Figure 4-1 The Telnet NVT model.

Telnet defined an innovative symmetrical negotiation mechanism that allowed a request by one to be the response to the other. The mechanism is used by the two users to select and enhance the characteristics of the NVT, such as character echoing, turn off half duplex, message size, line width, tab stops, logout, and so on. The negotiation is structured so that when the connection is established each side announces what it intends to do or not do (by sending the commands WILL/WONT followed by the appropriate Telnet options) and what it intends the other side to do or not do (DO/DONT). The options were encoded such that an implementation that did not understand an option could refuse it without having to "understand" it; that is, just send a WON'T *x*. Each side's announcement becomes a response to the other side. If there is no conflict, one side will announce DO *x*, and the other will announce WILL *x* (the WILL becoming a response to the DO and vice versa). If a conflict occurs, each option defines a scheme for resolution. Telnet is one of the very few symmetrical application protocols. Notice that although the protocol is symmetrical, what is being negotiated is asymmetrical. Different systems did or required different functions. The Telnet negotiation gave them an elegant means to attempt to offload some functions if possible and still get done what was needed.

While of little interest today, the handling of half-duplex terminals shows the subtlety and elegance of Telnet. At the time, terminals that could not send and receive at the same time (that is, half duplex) were still common, in particular IBM terminals such as the 2741, a computer-driven Selectrix typewriter. Consequently, they had to be accommodated, even though most understood that full-duplex operation was displacing them and were much simpler to handle. Most protocol designers took the idea of "turning the line around" literally and assumed that

Why Telnet Is Important?

Most textbooks no longer cover Telnet (undoubtedly because they deem remote terminal support a thing of the past). This is precisely what is wrong with today's networking textbooks. The reason for covering Telnet is not because it provides remote terminal support, but because it teaches lessons in understanding networking problems.

Telnet takes a problem that everyone else saw as asymmetrical (terminal-host) and found an elegant symmetrical solution. That solution made Telnet much more useful than mere terminal support. Telnet also finds an elegant solution to a classic "oil and water" problem (the handling of half/full duplex) that makes both degenerate cases of a more general solution.

We consider Telnet because these concepts are important in the education of network designers (the goal of a university education), even though they might not be for the training of network technicians.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 102

the protocol had to be half duplex. However, the designers of Telnet showed more insight. They realized that the protocol had to manage only the *interface* between the protocol and the remote user as half duplex; the *protocol* could operate full duplex. Hence, Telnet sends indications (Go Aheads) so that the receiver knows when to "turn the line around" (that is, tell the half-duplex terminal it could send). This allowed an application to simply send the indication regardless of whether the other side was full or half duplex, and the receiver either used it or ignored it. Half duplex was subsumed as a *degenerate* case and did not greatly distort the structure of the protocol (as it did with many others). Take, for example, the OSI Session Protocol, which made the protocol half duplex and made full duplex an extended service. Consequently, the minimal Session Protocol requires more functionality than one that uses the full-duplex *option*. Half-duplex terminals could use full-duplex hosts and vice versa. Neither really had to be aware of the other, and the application did not have to be aware of which was being used. As the use of half-duplex terminals declined, the use of the Go Ahead has quietly disappeared.

If Telnet got anything wrong, it was holding fast to a stream rather than record model.[4] The generally accepted wisdom in operating systems at the time was that the flow of data between processes should be streams. "Records" implied fixed-length records. However, the desire to hold to the accepted wisdom for Telnet meant that every character had to be inspected for the Telnet command characters. Telnet commands are a relatively rare occurrence in the data stream. A little record orientation (that is, putting Telnet commands and terminal data in separate "records") so that every byte did not have to be touched to find the relatively rare Telnet commands would have greatly decreased processing overhead.

But all in all, Telnet is a fabulous success both architecturally and operationally, as indicated by its continued use today. Telnet embodies elegant examples of efficient solutions to problems by making them degenerate cases of a more general model (rather than the more typical approach of simply shoving distinct mechanisms together to solve each case, which eventually leads to unwieldy implementations and to designs that are difficult to adapt to new uses).

File Transfer Protocol (FTP) was built on Telnet, partly for architectural reasons and partly for pragmatic reasons (Figure 4-2). FTP uses a Telnet connection to send its four-character commands followed usually by a single parameter

[4] Well, maybe one other. There was a Big Bad Neighbor who insisted that the best way for remote terminals to operate was character-at-a-time transmission and remote echoing by the host across the Net. This proved to be very slow to the users (primarily because of operating system overhead). Although attempts were made to develop a Telnet option to improve efficiency, it never proved workable. But this really isn't an inherent part of the protocol, so it is a footnote.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 103

terminated by CRLF (*carriage return, line feed*). The actual file transfer is done on a separate connection between data transfer processes. The architectural reason (and a good one) to separate the command and data streams is so that commands, especially aborts, do not get stuck behind large file transfers. This connection was generally a fixed offset from the Telnet connection, with one exception, the TIP.

The constraints of the TIPs had a major influence on the nature of FTP. TIPs were a variant of the ARPANET switch (IMP) that had rudimentary host software to connect users' terminals to hosts elsewhere on the Net. Users would open a Telnet connection and would act as the FTP client, typing in the commands directly. Then, using the SOCK (now PORT) command, the user would instruct the remote FTP server to connect to a socket on the TIP to send the file. Because the TIP had no file system, you might be wondering what was the point. Printers and other devices attached to the TIP were hardwired to certain socket numbers. Although this was done on occasion, it was not very popular. Because the TIP software ran as the low-priority task on the IMPs (after message forwarding), it often experienced significant delays.

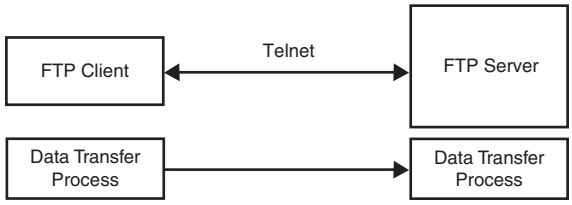

Figure 4-2 The ARPANET FTP model.

FTP defined a rudimentary *network virtual file system* (NVFS) and the basic commands to carry out file transfers and to interrogate a foreign file system. There was such a wide variety in file systems that the NVFS (like the NVT) was restricted to the bare minimum. The main emphasis was on the attributes of the file and saying as little about the nature of the contents as possible. There are basic conventions for the file format (characters, binary, and so on) and for the structure of the file (record or stream). The protocol allowed for checkpoint recovery and third-party transfers.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 104

One Step Forward, Two Back

It is curious that although the original version of FTP (RFC 542) allowed checkpoint recovery and third-party transfers, when FTP was revised for operation with TCP, this capability was removed by what I am told were reactionary factions. An incredulous development to remove useful functionality! We can only hope that these kinds of reactionaries do not continue to put their imprint on an Internet that needs visionaries, not "stick-in-the-muds."

Initially, mail was two commands in FTP. It wasn't until later that it was separated into a distinct protocol (that bears the mark of its origins). Rather than attempt to impose common file system semantics (which would have greatly increased the amount of effort required and the degree of difficulty in implementing it in the existing operating systems), FTP transparently passes the specifics of the host file system to the FTP user. An intelligent method for encoding responses to FTP commands was developed that would allow a program to do an FTP but at the same time provide the ability to give the human user more specific information that might be beneficial to determine what was wrong.

It is hard to say there is anything wrong with FTP per se. One can always suggest things it doesn't do and could, but for what it does, it does it about as well as one could expect, and none of these really break any new architectural ground. More would have been done at the time if it had not been for the constraints on the TIP (there were complaints at the time that the tail was wagging the dog) and schedule pressures to have something to use. (There were small things wrong that illustrate how a temporary kludge can come back to haunt. For example, as noted previously, FTP was kludged to allow TIPs to transfer directly to a specific socket given by the SOCK command. Later, this became a Best Common Practice (!) and a major problem with NATs. The SOCK command was bad architecture; passing IP addresses in an application is equivalent to passing physical memory addresses in a Java program! It was known at the time and should have been removed when the last TIP was removed from the Net.)

Remote Job Entry (Figure 4-3) is an early application protocol that is today obsolete, but in the 1970s submitting a program to run on a remote machine and retrieving the output (usually a printer file) was a major application. This was also the downfall of the early upper-layer architecture. The designers started paying too much attention to architectural elegance and not enough to the users' pragmatic requirements and constraints. It was not hard to see that the job input (yes, the card reader input) and the printer output were files. It was very neat to build FTP on top of Telnet. NETRJE required the ability to send RJE commands and move files. What could be simpler than a Telnet connection for RJE commands and then use FTP to move things around? Simple to describe and easy to implement (if you don't need it). NETRJE put the greatest resource usage where it could least expected to be: the RJE client.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 105

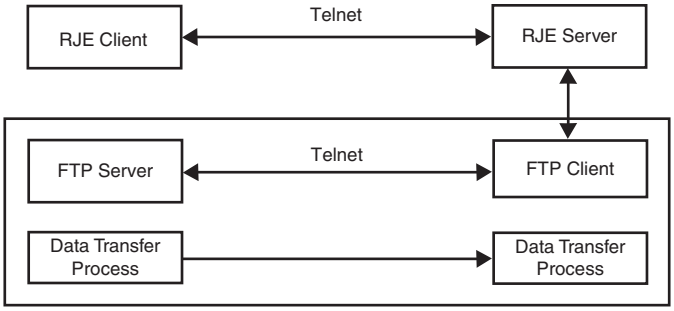

Figure 4-3 The ARPANET RJE model.

Consider a TIP, which did not have the resources to implement an FTP client, but only Telnet. To be a client RJE, a TIP would have to make a Telnet connection to an RJE server, which invoked its FTP client to create a FTP Telnet connection to the FTP server on the RJE client system. The TIP and most potential users of RJE did not have the resources to support an FTP server, which is why they were using RJE in the first place! Therefore, NETRJE never got much if any use. (A competing protocol, which did take into account the realities, did get considerable use. This protocol, CCN RJE [Braden, 1977], first proposed in 1971, puts the load where it belongs. It set up a Telnet connection and then opened data transfer connections for card reader and line printer transfers to sockets, a fixed offset from the Telnet sockets.)

What Was Learned

First and foremost, the upper-layer development of the ARPANET (Figure 4-4), as rudimentary as it was, proved that applications could be built and could be useful. Very useful. Technically, we gained valuable experience with distributed systems. We learned the difficulty of dealing with the subtle differences in the semantics that different systems had for what appeared very similar concepts. We even found some elegant solutions to some sticky problems that could serve as examples going forward. We learned the necessity of striking a balance between overspecifying and keeping it useful. And we learned not to get carried away with elegance: Our triangulation of Clauswitz and Mao struck home. But we also realized that we had chosen to solve specific problems. Although we had a good start, we did not yet understand the fundamental nature of the upper layers. What structure did a resource-sharing network require?

04_2422_ch04.qxd 4/9/08 1:09 PM Page 106

With these accomplishments came much enthusiasm among the people developing and using the early Net. In addition, to these specific protocols that everyone was using, there were numerous projects going on that went far beyond these three applications. To conclude that this was the extent of the use of the early Net would be grossly inaccurate. Distributed collaboration, hypertext systems, and production distributed database systems were all built during this period.

As implementations of Telnet and FTP came into use, people became excited at the possibilities. And it was clear that much more would be needed to make the Net a viable network utility. A group of interested parties formed a Users Interest Group (USING) to develop the necessary protocols. The group began to look at a common command language, a network editor, common charging protocols (not for the network but for using the hosts), an enhanced FTP, a graphics protocol, and so on. This group had an initial meeting in late 1973 to get organized, and a first major meeting in early 1974. However, ARPA became alarmed that this group would essentially wrest control of the network from them and terminated all funding for the work. (Hafner et al., 1996). ARPA would have benefited much more by harnessing that energy. Actually, ARPA had already, in a sense, lost control of the direction of the Net (short of shutting it down). The wide use of the Net might have come much sooner had ARPA encouraged, not squelched, the initiative the users showed.

Rerunning History

Of course, you can't do it. But, it is interesting to conjecture what might have happened had ARPA pursued development of the upper layers? Clearly, the direction USING had laid out to explore new applications should have been pursued. This was a major area, which had hardly been considered; there seemed many application protocols that could be useful. Given the capabilities of the systems, that direction most likely would have led toward something like a Novell NetOS environment. But the advent of the Web raises questions whether that path would have created or inhibited an environment conducive to the explosive growth of the Web and the immense benefits it has brought. (Although, a less explosive spread might have been had some benefits, too, a little less exuberance might have been wise.) It is easy

continues

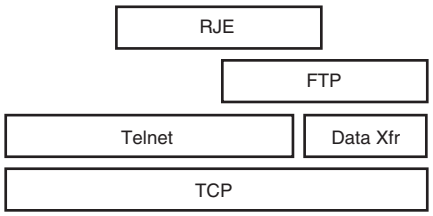

Figure 4-4 The ARPANET upper-layer architecture.

Upper-layer developments in the ARPANET halted for two decades. All subsequent developments have addressed specific protocols and have not considered how they relate to each other, what common elements there might be, or how it could be a distributed resource-sharing utility as early papers described it.

The early ARPANET upper-layer protocols made a greater contribution to our understanding of the design of protocols than to the architecture of the upper layers. But that was to be expected, given that it was a first attempt. There was only so much that

04_2422_ch04.qxd 4/9/08 1:09 PM Page 107

could be done. Developing distributed applications was not the primary rationale of the project anyway. Unfortunately, the innovations in these early protocols were often promptly forgotten by future work: the importance of separating control and data, or the fact that terminal protocols could be symmetrical and more useful, was never utilized by any other protocol, and I suspect never realized by their designers. For example, the early OSI VTP was symmetrical, because of Telnet, but the U.S. delegates from DEC pushed it into a much more cumbersome asymmetrical design in line with their product.

The use of the canonical form (for example, the NVT) was a major innovation both theoretically and practically. It is the tip of the iceberg to understanding of key elements of the theory of application protocols. This was the so-called "n^2 problem." Potentially, hosts would have to support $O(n^2)$ translations from each kind of system to every other kind of system. On the other hand, to define a model that was the least common denominator would have been so limiting as to be essentially useless. A middle ground was taken of defining a canonical (abstract) model of the elements that were to be transferred or remotely manipulated (in this case, terminal or file system elements). For Telnet, this was the NVT and for FTP, an NVFS. The definition of Telnet is strongly tied to the behavior of the NVT. Although the definition of FTP refers to its canonical model less frequently, it is no less strongly tied to the model of a logical file system. The concept was that each system would translate operations from its local terminal or file system in its local representation into the canonical model for transfer over the network while the receiving system would translate the protocol operations from the canonical model into operations on the local representation in its system. This reduced an $O(n^2)$ problem to a $O(n)$ problem. Of course, it also has the advantage that each system only has to implement one transformation from its internal form to the canonical form. It also has the benefit that new systems with a different architecture don't impact existing systems.

There are two unique characteristics to this approach that differed from other attempts. First, the model was taken to be a composite of the capabilities, not the least common denominator. Although there was no attempt to replicate every capability of the terminal or file systems represented in the network, useful capabilities that either were native capabilities or capabilities that could be reasonably simulated were included. Even with this approach, the wide variation in operating systems made it difficult to define every nuance of each operation to ensure proper translation.

continued

to see how a NetOS environment could lead to self-contained islands. On the other hand, by not pursuing upper-layer development, ARPA removed the driver that would have required richer lower-layer facilities, greater security, finishing naming and addressing, a much earlier impetus to provide QoS, and so on. This might have avoided many of the current problems of spam, virus attacks, mobility, scaling, etc. that plague today's Net). Without new applications as a driver, the only impetus the Internet had was to do what was necessary to stay ahead of growth, which through the remaining 1970s and early 1980s was relatively moderate. And most of that pressure was relieved by Moore's law. As discussed later, this turns out to be the critical juncture in the development of the Internet that allowed it to remain an unfinished demo for 25 years and is now the crux of our current crisis.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 108

Translation is the operative word here. Contrary to many approaches, the implementation strategy was not to implement, for example, the NVFS on the host as a distinct subsystem and move files between the local file system and the NVFS (such approaches were tried and found cumbersome and inefficient) but to translate the protocol commands into operations on the local file system and the files from the canonical file format to the local file format. The least common denominator approach was avoided by the simple recognition that there did not have to be a 1:1 mapping between operations on the logical object and the local representation but that the operation in the world of the model might translate into multiple operations in the local environment. In addition and perhaps most significantly, it was also found that the process of creating the abstract model for the canonical form uncovered new understanding of the concepts involved.

The ARPANET application protocols required the use of a single canonical form. One of the widespread complaints about this approach was requiring like systems to do two translations they didn't need, along with the assumption that it is more likely that like systems would be doing more exchanges with each other than unlike systems. Accommodating this requirement, along with a desire to regularize the use of the canonical form, led directly to the syntax concepts incorporated into the OSI presentation layer. However, by the time OSI began to attack the problem, the problem had changed.

A Rose Is a Rose Is a Rose

This was further complicated by the fact that at that point everyone knew their system very well, but knew little of the others, and the systems often used the same terms for very different concepts. This led to considerable confusion and many debates for which the ARPANET and the IETF are now famous. This also made writing clear and unambiguous specifications for application protocols difficult. Even when we thought we had (for example, Telnet, 1973), we were often brought up short when a new group would join the Net and come up with an entirely different view of what the specification said. OSI tried to solve this with FDTs, which are daunting to many developers; the Internet, by requiring two implementations, tends to inhibit innovation.

In the beginning, computers never "talked" to each other; and when they began to, they talked only to their own kind. So, when the ARPANET began making different kinds talk to each other, a lot of kinds had to be accommodated. As one would expect, over time the amount of variability has decreased; not only are there fewer kinds, but also systems tended to incorporate the canonical model as a subset of their system. Also, new network applications were created that had not existed on systems, so its form becomes the local form. Consequently, the emphasis shifted from canonical models to specifying syntax.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 109

Does this mean that the canonical model is no longer needed? We can expect other situations to arise where applications are developed either to be vendor specific or to be industry specific (groups of users in the same industry) in relative isolation that will later find a need to exchange information. The canonical form can be used to solve the problem. Today for example, the canonical model is used to create the *Management Information Bases* (MIBs) or object models for these applications or for interworking instant messaging models.

The ARPANET experience showed that there is some advantage to getting it wrong the first time. The first Telnet protocol was not at all satisfactory, and everyone believed it had to be replaced. But the experience led to a much better design where the conflicting mechanisms were accommodated not by simply putting in both (as standards committees are wont to do) but by creating a synthesis that allowed both to meet their needs without interfering with the capability of the other.

But it has to be very wrong. When FTP was completed in 1973, there was a general feeling that the problem was much better understood and now it would be possible to "get it right." However, it wasn't wrong enough, and it never went through the major revision, although some minor revisions added commands to manipulate directories, and so on.[5]

(While this is an example of the tried–and-true rule of thumb, that you "always have to throw the first one away," this may also be a consequence of "we build what we measure." Unlike other disciplines where the engineering starts with a scientific basis, we have to "build one" in order to have something to measure so that we can do the science to determine how we should have built it. No wonder we throw the first one away so often!)

With the early termination of research on applications in 1974, the early developments were limited to the bare-minimum applications. With the impetus removed to develop new applications that would push the bounds of the network, new insights were few and far between. Perhaps one of the strongest negative lessons from this early upper-layers work was that elegance can be carried too far. RJE using FTP and FTP and RJE using in Telnet led to an impractical solution. We will defer our consideration of applications in the Internet now (more or less keeping to the chronology), and shift our attention to OSI to see what it learned about the upper layers and then return to the Internet to pick up later developments there and see what all of this tells us about the fundamental structure of networks.

[5] It is painful to see kludges we put into FTP to accommodate constraints at the time now touted as "best practice."

04_2422_ch04.qxd 4/9/08 1:09 PM Page 110

The OSI Attempt or "Green Side Up"[6]

Session, Presentation, and Application

Beginning in 1978, OSI was the first of the standards groups intent on getting something out quickly and the first to learn that with a widely diverse set of interests that it could be difficult, if not impossible, to achieve agreement. At the first meeting in March 1978, the group adopted an architecture developed by Charles Bachman, then of Honeywell, that had seven layers. At that time, the characteristics of the protocols for the lower four layers were well-established. Although there was uncertainty about what went in session, presentation, and application (see Figure 4-5), the seven layers in the Honeywell model seemed to make a reasonable working model. It was clear there would be many application protocols. But for the time being, the terminal, file, mail, and RJE protocols formed the basis of the work. What parts of these protocols, if any, went into the session and presentation layers? Or did they all belong in the application layer? And, did other functions belong in session and presentation? For the next three years or so, considerable debate continued, attempting to work out the upper-layer architecture. The upper layers were pretty much a clean slate.

Speeding Up Standards

OSI was intent on getting things done quickly! I know. Many will find this hard to believe, but it is the case. There has been a lot of talk about speeding up the standards process. From years of participating and watching the standards process, it is clear that the vast majority of the time is consumed by people becoming familiar and comfortable with unfamiliar ideas. There is little, if anything, that can speed up such a process. Building consensus simply takes time; and the greater the diversity of interests and the more people involved, the longer it takes. Rushing a consensus or trying to force a particular answer usually destroys the result (for example, SNMPv2). There are no shortcuts. When IEEE 802 started, they were going to produce Ethernet standards in six months; it took three years. The ATM Forum has a similar history. The IETF is a classic example. When it

continues

As with everything else in OSI, there was no consensus on the upper layers, and the disagreement was along the same lines as in the lower layers: the PTTs versus the computer industry. The European PTTs had two point products they wanted operating under the OSI name. And it didn't matter to them whether accommodating them left a path open for future applications. They were a monopoly. Customers had to buy whatever they offered. The computer industry, on the other hand, realized that the upper layers had to lay a foundation for everything to come. So as the computer industry faction began to try to make sense of the upper layers, the European PTTs inserted themselves into defining the session layer. They had been developing protocols for two new services to run over X.25: teletex and videotex. Teletex was billed as e-mail. It was actually telex with some memory and rudimentary editing capability, a far cry from the e-mail protocols

6 The punch line of a politically incorrect ethnic joke from at least the 1970s, possibly before. (I know of a tree nursery company that used the name). The joke is about some guys (insert politically incorrect ethnicity of your choice) laying sod and the foreman having to constantly remind them, "Green side up." The reader can fill in the rest.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 111

that had been ubiquitous in the ARPANET and other research networks for almost a decade.[7] Videotex was more sophisticated: a terminal-based information system with rudimentary graphics capability. Although hypertext had been around for over ten years at that time, it was not widely available. Videotex was targeted at what could be done with the technology of the early 1980s.

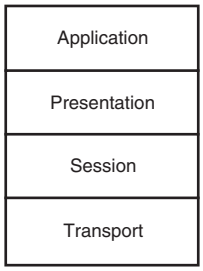

Figure 4-5 The OSI upper-layer architecture.

The PTTs, jumping on the OSI bandwagon of the early 1980s, wanted the teletex and videotex protocols (which were already designed and being built) to be OSI. OSI was just beginning to determine what the upper layers should be. The PTTs basically laid out their protocols and drew lines at various places: This small sliver is the transport layer; here is session layer, there is really no presentation, and the rest is the application layer. These were intended to run over X.25. Throughout the late 1970s and early 1980s, the PTTs argued in every forum they could find that transport protocols were unnecessary. However, OSI was coming down hard on the side of the debate that said X.25 was not end-to-end reliable and a transport protocol was necessary. The PTTs insisted X.25 was reliable. So, Class 0 Transport was proposed by the PTTs so that they would have a transport layer that didn't do anything. And then, when they got to the application layer and

[7] The PTTs have consistently proposed initiatives [videotex, teletex, ISDN, ATM, WAP, and so on] whose market window, if it exists at all, passes before they can create the standard, build the equipment, and deploy it. The sale of these is a credit to their marketing departments or the desperation of their customers.

continued

was reasonably small and its population fairly homogeneous and academic, it developed a reputation for doing things quickly. Now that it has a more diverse participation, it is taking longer than any other standards group has ever taken. (For example, IPv6 has taken 12 years for a small change.) The IETF's participation is still less diverse than OSI's was. It appears that other factors are contributing to the lengthy development time.

If We Build It, They Must Come.

The fundamental market strategy of a PTT. But in this case they didn't. People will not pay to get advertising. The primary, and perhaps only, successful example is the French Minitel. France Telecom justified giving away the terminals with phone service with the argument that between the costs saved in printing and distributing phone directories annually and the revenue from advertisers and information services, such as booking airlines or trains, more than covered the cost. As with the Web, pornography was the first to make money from videotex. However, giving away the terminals makes it hard to justify upgrading the hardware. The French PTT failed to see that technology would change much faster than phone companies were accustom. Although, the design was resurrected in the 1990s and called WAP.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 112

thought no one was looking, they stuck in RTSE to provide end-to-end reliability and said they were doing checkpointing for mail. It was really quite amusing how many pundits and experts bought the argument.

The functions proposed for the session layer that fell out of this exercise were various dialog control and synchronization primitives. There was a strong debate against this. Most people had assumed that the session layer would establish sessions and have something to do with login, security, and associated functions. This came soon after the competing efforts were made a joint ISO/CCITT project. There was considerable pressure to demonstrate cooperation with the PTTs, even if it was wrong. So, the Europeans block voted for the PTT proposal.[8] (So, the OSI session layer was stolen by the PTTs and had nothing to do with creating sessions—something that took many textbook authors a long time to figure out.)

Meanwhile, the Upper-Layer Architecture group had continued to try to sort out what the upper layers were all about. It was fairly Stoic about what had happened with the session layer. Most believed that even if the functions were not in the *best* place, it was close enough for engineering purposes (an argument we hear often in the IETF these days). The session functions were needed, and when presentation and application were better understood, a way could be found to make the outcome of the session layer less egregious. (As it would turn out, the problems it causes are too fundamental, but this does serve a valuable lesson about compromising technical veracity to commercial interests.)

Early on (and almost jokingly), it had been noted that in the applications (or in the application layer) PDUs would have no user data. They would have all PCI, all header. In other words, this was where the buck stops. There was nowhere else to forward user-data. Or more formally, the PDUs contained only information (that which is understood by the protocol interpreting the PDU) and had no user data (that which is not understood by the process interpreting the PDU) for a higher layer. We will also find that this distinction still matters even in the application.

Because the upper-layer problem was less constrained than in the lower layers and clearly part of the more esoteric world of distributed computing, a more formal theoretical framework was necessary to understand the relations among the elements of the application layer. For this, they borrowed the idea of conceptual schema from the database world.

[8] Interestingly, AT&T did not side with the other PTTs and argued that the functions being proposed for the session layer did not belong there but belonged higher up.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 113

From this, it was clear that for two applications to exchange information, the applications needed "a shared conceptual schema in their universe of discourse."[9] If the other guy doesn't have a concept of a "chair," it is impossible to talk to him about "chairs" regardless of what language you use. As it turned out, this was a generalization of the concept of the canonical form developed for the ARPANET protocols, but now with greater formalism. The concept comes directly from Wittgenstein's *Tragtatus*. The conceptual schema defines the invariant semantics that must be maintained when translating between systems with different local schemas.

Database Schemas

In the database world, the conceptual schema was the semantic definition of the information, which might be represented by different structures or syntaxes for storage (called the internal schema) and for presentation to a user or program (called the external schema).

Clearly, if the applications had shared conceptual schemas (that is, semantics), the application layer must provide the functions to support the management and manipulation of these semantics. And wherever there are semantics, there must be syntax. So if the application layer handles the semantics, then clearly the presentation layer must handle the syntax! Wow! Now they seemed to be getting somewhere; maybe there was something to these upper three layers! On the other hand, one cannot manipulate semantics without its syntax. Consequently, one finds that the presentation layer can only negotiate the syntax used by the application. Any actual syntax conversions must be done by the application.

So, the common part of the application layer provides addressing and negotiates the semantics to be used, identified by the application context, and the presentation layer negotiates the syntax identified by the presentation context.

Thus, the presentation layer became the functions to negotiate the syntax to be used by the application. It was envisaged that a syntax language would be used to describe the PDUs used by an application. The syntax language was defined as an abstract syntax and a concrete or transfer syntax. The abstract syntax of a protocol refers to the data structure definitions specifying the syntax of the PDUs in a particular syntax language, whereas the concrete syntax refers to a particular bit representations to be generated by that abstract language (Table 4-1). By analogy, the data structure constructs of a language such as C or Pascal correspond to an abstract syntax language. The data structure declarations in a program written in such a language correspond to the abstract syntax of a protocol, while the actual bit representations generated by the complier represents the concrete syntax.

[9] There was a heavy contingent of logical positivists in the group. What can I say? At least they weren't French deconstructionists.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 114

Table 4-1 *The Distinction between Abstract and Concrete Syntax Is Similar to the Distinction Between a Programming Language and Its Code Generators*

Programming Language		ASN.1
<integer> ::=INTEGER<identifier>;	**Language Definition**	GeneralizedTime ::= [Universal 24] Implicit VisibleString
INTEGER X;	**Statement Definition**	EventTime ::=Set { [0] IMPLICIT GeneralizedTime Optional [1] IMPLICIT LocalTime Optional}
(32-bit word)	**Encoding**	I | L | GeneralizedTime
$012A_{16}$	Value	0203000142_{16}

The presentation protocol provided the means to negotiate the abstract and concrete syntaxes used by an application. Note that presentation only *negotiates* the syntax. It does not do the translation between local and transfer syntax (what the ARPANET called the local and canonical form). OSI realized that such a clean separation between syntax and semantics is not possible. The translation must ensure that the semantics of the canonical model are preserved in the translation. This was referred to as the *presentation context*. Because an application defined its PDU formats in the abstract syntax, an application could use any new concrete syntax just by negotiating it at connection establishment. OSI defined *Abstract Syntax Notation 1* (ASN.1) as the first (and it would appear to be the last) example of such a language. Then it defined the *Basic Encoding Rules* (BER) (ISO 8825-1, 1990) as a concrete encoding. BER was a fully specified encoding of the (type, length, value) form and allowed the flexibility for very general encodings. BER proved to be inefficient in its use of bandwidth and a little too flexible for some applications. With impetus from ICAO, the International Civil Aviation Organization, OSI embarked on two other sets of encoding rules: one to be bandwidth efficient, Packed Encoding Rules (PER), and one to be more processing efficient, Light-Weight Encoding Rules (LWER). PER was done first and archieved both goals. It was 40% to 60% more bandwidth efficient, but surprisingly was roughly 70% more efficient in processing. No need was seen in pursuing LWER, so it was abandoned. Experimentation indicates that PER is about as good in encoding efficiency as is likely, indicated by the fact that data compression has little or no effect on it.

Initially, OSI embarked on developing a number of application protocols, some of which have been alluded to here: OSI variations on Telnet, FTP, and

04_2422_ch04.qxd 4/9/08 1:09 PM Page 115

RJE (JTM, *Job Transfer and Manipulation*). *File Transfer Access Method* (FTAM) and JTM were based on British protocols, and *Virtual Transfer Protocol* (VTP) was based on a combination of proposals by DEC and European researchers. Each had more functionality (to be expected since they came ten years later), but none show any new architectural innovations, including those incorporated in the early ARPANET/Internet protocols, nor were they designed so that initial implementations were inexpensive. The minimal implementation was always fairly large.

Later, OSI developed a number of important areas that the Net had ignored, such as *commitment, concurrency, and recovery* (CCR), a two-phase commit protocol intended as a reusable component; TP (*transaction processing,* which made apparent the limitations of the *upper-layer architecture* [ULA] structure); *Remote Database Access* (RDA); *Remote Procedure Call* (RPC), how to standardize 1 bit; and a directory facility, X.500; and management protocol, *Common Management Information Protocol* (CMIP), which is discussed later. CCITT (ITU-T) also took the lead in developing an e-mail protocol, X.400. The original ARPANET e-mail had been part of FTP and was later extracted into SMTP without major change. Mail was directly exchanged between computers (most were timesharing systems). By the time of the discussions leading up to X.400, most systems were workstations or PCs. Hence, this led to distinguishing servers that received mail on behalf of users: the concepts of message transfer agents and message servers. Initially, X.400 used only the session layer, but as the presentation layer was incorporated, unforeseen problems arose.

For the application layer, OSI recognized that a common connection-establishment mechanism would be required for all applications. One could not expect applications to be "hardwired" to network addresses. Without such a common mechanism, a host would have to be able to interpret all the initial PDUs of all the applications it supported to determine what application to deliver it to.

The common connection-establishment mechanism, *association control service element (*ACSE), provides mechanisms for application layer addressing, application context negotiation, and authentication. Thus, the applications defined by OSI (CCR, TP, file transfer, and so on) in essence only define the behavior of the data transfer phase. ACSE provides the establishment phase.

Well-Known Sockets

The ARPANET had also recognized the requirement for a directory early on, but the need to demonstrate that the network could do useful work outweighed all else, and so well-known sockets were created as a kludge to demonstrate the first few applications. "Doing it right" would have to wait. No one expected that there would be no new applications for 20 years and that we would still be waiting to "do it right." By the early 1980s, well-known sockets were an institution for a new generation, and amazingly enough, one still hears arguments that well-known sockets were an inspired piece of design. A sad commentary on the state of the field.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 116

The other major contribution to the upper layers that OSI did seem to get right was the nature of the application process. At first glance, this looks like a typical standards committee compromise. Being part of ISO, as OSI began to consider applications, it ran into the problem of what was within OSI and what was the purview of other committees, such as database, programming languages, banking, and so on. While on one hand this was the usual turf battle, on the other it raised a very real question of where networking stopped and other aspects of computing started. This quickly devolved into a very esoteric debate over the nature of distributed applications and whether application processes were inside or outside OSI (what is the communications environment). After much debate, the solution that was arrived at was that the application process was on the line (see Figure 4-6), yet another standards committee nondecision. But with consideration, one finds that it is not only the right answer, but that it is also a fairly powerful one.[10]

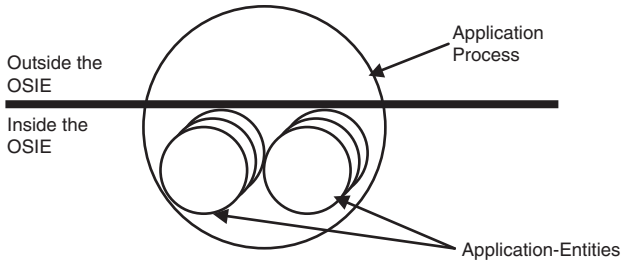

Figure 4-6 An application process consists of one or more application entities, of which there may be multiple instances of each.

What one finds is that it makes much more sense to treat the application protocol(s) as part of the application; or in OSI terms, application entities are part of the application process. If the application protocols are considered distinct entities used by an application, one gets into an infinite regress with respect to the shared state. The parts of the application process, not part of the application entity, are those aspects of the application not involved in communications. The *application process* (AP) consists of one or more *application entities* (AEs), which are instantiations of the application protocols. The application-entities are part of the communications environment (they called it the *OSI environment,* or OSIE) while the rest of the application is not. In other words, the top of the application layer includes the AEs but not the rest of the AP.

[10] Who would have thunk!

04_2422_ch04.qxd 4/9/08 1:09 PM Page 117

Consider an example: A hotel reservation application might use HTTP (an application entity) to talk to its user and one or more remote database protocols (different application entities) to actually make the reservations. The application process outside the OSIE moderates the use of these protocols, doing whatever processing is necessary to convert user input to database requests. Clearly, an application process can have not only different kinds of AEs, but also multiple instances of them, and there could be multiple instances of the AP in the same system. Not all applications would require the full richness of the structure, but some would. The application naming must allow relating an application and its AEs (and their instances) to ensure that communication is associated with the correct application. This turns out to be a powerful and general model and easily describes and supports the requirements of distributed applications.

This also illustrates where the ARPANET approach of using operating systems as a guide turns out not to be sufficiently rich. From this perspective, we can see that all of our early applications (Telnet, FTP, mail) were special cases: The AE and the AP are essentially synonymous; there is little or no AP functionality distinct from the application protocol, and there is generally one per system. So, naming the application protocol was all that was necessary. It is not until the advent of the Web that there are applications in the Internet where the application and the protocol are not synonymous; that is, there is significant functionality clearly not associated with the application protocol. And there are many applications in the same host using the same protocol. One wants to access cnn.com (the AP), not HTTP (the AE). Relying on the operating system model had been a good first cut, but it was not sufficiently general. Again, we see evidence of the harm done by not pursuing new applications into areas that would have taken us beyond the operating system model. Given the nature of the discussions going on in the ARPANET in 1974, I have every confidence that had new applications been pursued, something equivalent to this model would have evolved. It is interesting to note that this distinction arises primarily when considering "user applications" and less so when considering system applications. We definitely had a systems programmers' perspective. Distinguishing application protocols from the application while at the same time recognizing that naming the application protocol first requires naming the application turns out to be a very powerful insight and has implications far beyond its immediate use in the application layer.

That Which Must Not be Named

Why is there no name for this "other part"? The part of the AP not in any AE. Good question. On the one hand, some member bodies insisted that we not describe or name anything outside the OSIE. This was the turf of other committees, and we shouldn't say anything about it. On the other hand, to not do so is just asking for trouble. To draw the distinction too closely would have some pedants insisting on some formal boundary, whereas good implementation would probably dictate a less-distinct boundary between the AE aspects and the AP aspects or even interactions of AEs. Later we will see that this was well founded.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 118

The other aspect of the OSI application layer isn't so much new as just good software engineering. The OSI approach to application protocols allowed them to be constructed from reusable modules called *application service elements* (ASEs) see Figure 4-7. It was clear that some aspects of application protocols could be reused (for example a checkpoint-recovery mechanism or a two-phase commit scheme). Clearly, some part of the application protocol would be specific to the task at hand. As we have seen, a common ASE was required by all applications to create the application connection, ACSE. There was also a connectionless form of ACSE called A-unit-data that complemented the unit-data standards in the other layers. In OSI, an application protocol consisted of ACSE and one or more ASEs bound together by a *control function* (CF), which moderated the interactions among the ASEs. Because ASEs had inputs and outputs like any other protocol state machine, the CF was just a state machine that controlled the sequencing of their interactions. There is nothing unique about this approach. It is just good software engineering. But it is unfortunate that this approach was not pursued; it might have facilitated uncovering some interesting structures in application protocols.

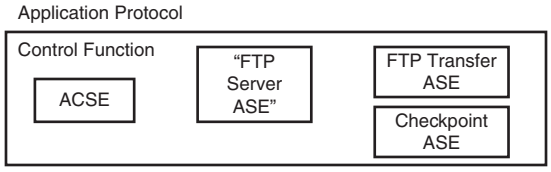

Figure 4-7 OSI application protocols were assembled from modules called application service elements bound together by a control function to moderate interactions among them. All applications protocols used ACSE to set up application connections and for authentication. (There was also a connectionless form of ACSE.)

But as this picture of the upper layers came together in the early 1980s, cracks began to appear in the structure, and as the decade wore on and the applications became more ambitious, the cracks became more severe. By 1983, it had become clear that each session and presentation connection supported a single application connection. There was no multiplexing above transport and no need for addressing in session and presentation. Consequently, there was no need for the session and presentation layers to set up connections serially, and it would incur considerable overhead if it were done that way. To some extent, OSI was even further down the path of too many layers causing too much overhead that had undermined the early ARPANET's attempt. But there was a way out.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 119

Also, it was becoming more and more clear that the functionality of the upper layers decomposed not so much "horizontally" as the lower layers did, but more "vertically" into modules. (In the mid-70s, I had made this observation while investigating how to decompose protocols to improve their processing (others remarked on it as well), but there wasn't enough data to discern a pattern.) This idea was opposed by many who were intent on adhering to the seven-layer structure regardless.[11] It was clear that the implementation of the establishment phase of session, presentation, and application should be considered to be one state machine and the session functional units (that is, the dialog control and synchronization primitives) should be viewed as libraries to be included if required. In 1983, steps were taken by making slight changes to the protocol specifications to allow the connection establishment of all three upper layers to be implemented as a single state machine. This meant that the preferred implementation was to merge ACSE, presentation, and session into a single state machine. This created a much smaller, more efficient implementation, *if* the implementer was smart enough to see it.[12]

This clearly says that the layer structure might need modification. But in 1983, it was too early to really be sure what it all looked like. Just because the connection establishment of the upper three layers could be merged was not proof that the data transfer phases should be. And remember, the CCITT wanted the data transfer phase of the session layer just as it was for teletex and videotex. So, it was not going to support any radical reworking of the upper layers. It also opposed the use of the presentation layer (teletex and videotex were defined directly on top of session) and would not have agreed to a solution that made presentation a *fait accompli.* So in the end, it was felt that this was the best that could be achieved at the time. It was a small modification that made a move in the right direction and allowed much more efficient implementations to be done. It was hoped that it would be possible to work around what was in place, after there was a better understanding. Meanwhile, the upper-layer group continued to work out the details.

Standards Aren't Design Specs

This discussion leads to the oft-cited observation that an architecture does not need to bear a resemblance to the implementation. This is in some sense true. There *are* many "correct" implementation strategies for any given architecture or protocol standard. One should never treat a standard or any protocol specification as an implementation design. However, specifications should not stray too far from the more common implementation strategy. In fact, what one often discovers is that the optimal implementation strategy and the "correct" architecture are often quite close. If not, something is wrong. Of course, recently the tendency has been to do the implementation and then define the architecture. This is equally wrong. Doing the arithmetic before the algebra leads to inefficiency and dead ends, the contra-positive of our philosophical triangulation (Clauswitz and Mao). It is not only important to keep theory close to practice, but also to keep practice close to theory!

11 It was amazing how quickly for some that the seven-layer model went from a working hypothesis to a religion.

12 Most weren't. This became known as the "clueless test" for implementers.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 120

Don't Feed the Animals

As we noted in the preceding chapter, OSI had two 800-pound gorillas in the room: IBM and the PTTs. Many were very concerned about giving either one the upper hand to push their "proprietary" approach. On the one hand, the PTTs had too much clout to reverse their position on session and presentation. Teletex and videotex were just entering the market. On the other hand, going to a five-layer model ran the risk of admitting that IBM's SNA had been right and having IBM force SNA on the committee—it chaired nearly every national delegation. And there was one attempt, known as the Br'er Rabbit episode, to replace the upper layers with SNA's LU6.2. Hence, the illusion of seven layers was maintained.

A Note on Committee Realities

Some readers will think the OSI people were pretty stupid for not seeing these problems coming. They did. Or at least some did. However, if you have ever participated in a standards committee, you know that taking the "right" technical approach seldom counts for much. What does count are implications for products and market position. At the time the structure was being set in the early 1980s, these objections were theoretical, of the form "this could happen." They could not point to specific protocols where it happened. This, of course, allowed so-called

continues

This would not have been too bad if it were the only problem, but other problems arose that indicated that there were more fundamental problems with the upper-layer structure

According to the theory, if an application needed to change the presentation context during the lifetime of the connection, it informed the presentation layer, which made the change by renegotiating it. Different parts of an application might require a different syntax to be used at different times on a connection. However, it quickly became apparent that because the application protocol could invoke session synchronization primitives to roll back the data stream, the presentation protocol would have to track what the application protocol requested of session so that it could ensure that the correct syntax was in use for the data stream at the point in the data when it was rolled back by the session protocol. This added unnecessary complexity to the implementation and was a strong indication that the session synchronization functions belonged above presentation, not under it.

Furthermore, as more complicated applications began to be developed, conflicts in the use of the session layer developed. For example, CCR was defined to provide a generic two-phase commit facility that used session functions to build the commit mechanism. Later, the transaction processing protocol, TP, used CCR for two-phase commits but also made its own use of session primitives necessarily on the *same* session connection. Session had no means to distinguish these two uses, and there was no guarantee that they would be noninterfering. TP would have to make sure it stayed out of the way of CCR, but that violates the concept of CCR (and all protocols) as a black box. In essence, session functional units were not re-entrant and really in the wrong place.

Also, there were the problems caused by relaying in X.400 e-mail. Connections between applications were the ultimate source and destination of data. However, the RM explicitly allowed relaying in the application layer. X.400 (or any mail protocol) may require such relaying. This implies that while the syntax of the "envelope" has to be understood by all the intermediate application layer relays, the syntax of the "letter" needs only to be understood by the original sender and ultimate receiver of the letter, not all the intermediate relays. Because there are far more syntaxes that might be used in a letter, this is not only reasonable, but also highly desirable. However, presentation connections can only

04_2422_ch04.qxd 4/9/08 1:09 PM Page 121

have the scope of a point-to-point connection under the application layer relay. It was not possible for the presentation layer to negotiate syntax in the wider scope of source and destination of the letter, independent of the series of point-to-point connections of the relays. The "letter" could be relayed beyond the scope of the presentation layer on another presentation connection to its final destination. The architecture required the relay to support all the syntaxes required for the envelope *and the letter* even though only the sender and the receiver of the letter needed to be able to interpret its syntax. SMTP avoids this by an accident of history. When mail was first done, there was only one syntax for the letter, ASCII. By the time there were more, *Multipurpose Internet Mail Extensions* (MIME) could simply be added for the "letter," with ASCII required for the envelope.

All of these indicated severe problems with the upper-layer architecture, but the problems were also an indication of what the answer was. And, although it might not be compatible with the seven-layer model in its purest form, it wasn't that far off from its original intent.

continued

pragmatists, those with a vested interest, and the just plain dull to counter these arguments as just speculative prattling. Or arguments of why go to the trouble of doing it right if we don't know it will really be needed. (Sound familiar?) It was not until the late 1980s that work had progressed sufficiently that there was hard evidence. This is often the case in all efforts, in standards and elsewhere. One does not have the hard data to back up real concerns. The demand to accommodate immediate needs regardless of how bad a position it leaves for the future is often overwhelming and much more damaging. Basically, don't violate rules of good design. It might not be clear what will need it, but one can be assured that something will.

What Was Learned

OSI made major progress in furthering our understanding of the upper layers but ran into problems caused by conflicting interests: both from economic interests and in adhering to a flawed model.

Authentication, addressing the desired application, and specifying some initial parameters were generally the concepts associated with session functions. These were embodied in ACSE in the application layer. So without stretching things too far, the theft of the session layer for the PTT teletex and videotex protocols turned the OSI upper-layer architecture upside down. (Right, green side down!) Avoiding the problem, by saying that the use of session was a "pass-through" function, merely added unnecessary complexity to the layers that were passed through. It is hard to argue that there is never a situation where a pass-through function may be the correct solution. However, pass-through functions must be limited to those that do not cause a state change in an intervening layer. This severely limits the possibilities. Furthermore, in general one would want functions located as close to their use as possible. There would have to be a strongly overriding reason for locating another function between an application and a function it uses. Fundamentally, it appears that pass-through functions should never be necessary.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 122

The fact that there is no multiplexing in the session and presentation layers is a strong indication that although there may be session and presentation functions, they are not distinct layers. The presentation negotiation of syntax is best associated as a function of the application connection establishment mechanism, which practically speaking it is. The session functions are actually common modules for building mechanisms used by applications that should be considered, in essence, libraries for the application layer. This also satisfies the previous result and is consistent with the common implementation strategy. Assuming that there are other common building blocks for applications than those found in the session protocol, this would seem to imply that one needs an application layer architecture that supports the combining of modules into protocols.

So, if the architecture is rearranged to fit the implementation strategy, all the earlier problems are solved…except one: the syntax relay. The flow for relaying is independent of the flow of the data for what is relayed. To take the e-mail example, and applying what was developed in the earlier chapter, the letter from an e-mail is a connectionless higher-level flow that specifies the syntax of the data it carries and is encapsulated and passed transparently by the relaying protocol of its layer (with its syntax) via a series of relays. Essentially, we need two layers of application connections to separate the "end-to-end" syntax of the letter and the "hop-by-hop" syntax of the envelope. An interesting result but not all that surprising. We should expect that applications might be used as the basis for building more complex applications.

One might want to build application protocols from modules but also want to build application protocols from other application protocols. To do this about 1990, I proposed to revise ACSE, to make it recursive.[13] To do this required ACSE to also negotiate syntax of these protocols within protocols. As it turned out, the design of the OSI transaction processing protocol using session, presentation, and application layers in the traditional manner was cumbersome and complicated. However, with this extended application layer structure, the design was straightforward and closely reflected the implementation. This model could also solve the mail relay problem by simply making the letter an encapsulated connectionless PDU with its own syntax being sent among application relays with their own application connections and syntax for the envelope.

[13] Writing service and protocol specifications for a recursive protocol is a very interesting exercise. One learns how dependent one has become on thinking that the layer below is different.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 123

OSI improved our understanding of the nature of the upper layers. The recognition of the roles of syntax and semantics in application protocols was crucial. The distinction between abstract and concrete syntax is equally important and allows protocols to be designed such that they are invariant with respect to their encoding. This means the encoding can be changed without rewriting the entire protocol in much the same way that a compiler can change code generators without changing the language. (Remember in Chapter 3, "Patterns in Protcols," this property was needed to further simplify data transfer protocols.) The recognition that application and presentation context were specific cases of a general property of protocols (that is, negotiating policy) was also important. The realization that the field in lower-layer protocols to identify the protocol above was really a degenerate form of presentation context (that is, identified the syntax of the data transfer phase), not an element of addressing, contributed to our general understanding of protocols. In the same way that the presentation layer proves to be a false layer, this too proves to be a false and unnecessary distinction. However, OSI became locked into a particular structure of layers too early (as did the Internet). If ACSE had been the session layer, which is what most had intended, it would be before it was stolen the result would still not be quite right, but it would have been much closer and possibly close enough to get the rest of the solution. However, it was a sufficiently large change that the majority of participants could not accept a shift from seven layers to five, primarily for political/economic reasons and because OSI would be turning its back on protocols that were dead ends anyway (for example, keeping checkpoint recovery out of FTP to protect ancient legacy equipment).

The Main Lesson of OSI

The main lesson of OSI is to never include the legacy in a new effort. We can see how OSI's attempt to accommodate ITU caused many bad compromises to be made in the upper layers and bifurcated the lower layers. ITU hijacked the session layer for services that were a dead end. This so contorted the structure of the upper layers that there was no means to correct the problems without considerable modification of the architecture and the protocols. The ITU insistence on X.25 in the lower layers split them in two. (How X.25 was supposed to accommodate applications of the 1990s is still a mystery.) OSI was two incompatible architectures. All these conflicts undermined the effort and confused its adopters. Every word and punctuation mark in every document was a point of contention. Every minor point was a compromise

continues

Network Management

As the number of private networks grew in the early 1980s, interest grew in how to manage them. The early ARPANET had an excellent network management capability, but it was internal to BBN (MacKenzie, 1975). The stories are legend: BBN calling PacBell to tell them one of its T1 line from Los Angeles to Menlo Park was having trouble and PacBell not believing that the caller wasn't in either Los Angeles or Menlo Park but calling from Boston. The entire network could be observed and largely debugged from its management center in Cambridge, Massachusetts, as long as a switch (IMP) didn't suffer a hard power failure.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 124

Then someone had to be physically there to run the paper tape loader to reboot it. IMP software downloads were distributed over the Net under BBN control. Existing network management was primarily oriented toward controlling terminal networks, and the management systems were very vendor specific (not at all suitable for managing multivendor networks). There was little experience with managing networks of computers.

The major difference in packet networks was one of moving from control to management. Early terminal (and voice networks) viewed these systems as network control, and they did. Most decisions about the network, including routing, were made from the network control center. However, a major characteristic of packet-switched networks was and is that events in the network are happening too fast for human intervention. A human in the loop will make the situation worse, working at cross-purposes to the routing algorithms and flow-control mechanisms. Hence, there is a real shift from "control" to *management*.

Starting in late 1984 under the impetus of the GM *Manufacturing Automation Protocol/Technical and Office Protocols* (MAP/TOP) effort, a major push was made to develop network management for factory and office automation. Coincidentally, earlier in 1984, I had worked out the basics of network management architecture for my employer. GM liked what we had done, and this work formed the basis for the MAP/TOP network management effort. This early attempt was based on the IEEE 802.1 LAN management protocol (IEEE, 1992). The protocol was in use as early as 1985. The protocol had a simple request/response structure with set, get, and action operations and an asynchronous event. The operations were performed on attributes of objects in the device, the precursor of the MIB. While the protocol developed by IEEE 802 was intended to operate over a LAN, there was nothing about the protocol that was specific to LANs. Recognizing that the responses would likely span more than one packet, the request/responses used a transport protocol, while the asynchronous event was sent as connectionless.

IBM was between a rock and a hard place. SNA was a hierarchical network architecture intended to support the centralized mainframe business. SNA was perfectly positioned to complement and avoid the phone company market. But now the packet networks destroyed this division and put

continued
with the old model, consistently weakening any potential of success. There was never a consensus on what OSI was, making it two architectures for the price of three, when it should have been one for half the price.

On the surface, including the legacy seems reasonable and cooperative. After all, one must transition from the old to the new. And it *is* reasonable, but it is wrong. Contrary to what they say, the legacy doesn't really believe it is the legacy. They will point to successes in the marketplace. If people buy it, how can it be wrong? There will be considerable emotional attachment to the old ways, not to mention the vested interest in the many products and services to be obsoleted by the new effort. The legacy will continually be a distraction. Their reticence to change will cause the new effort to fall short of its potential and jeopardize its success. Death by a thousand cuts.

No matter how great the temptation to be reasonable and accommodating, this is one point that cannot be compromised.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 125

IBM and the phone companies in direct competition. The computer industry was endorsing these packet networks with a peer architecture. IBM had 85% of the computer market, so the others were not an immediate threat, although minicomputers and workstations were coming on fast. However, the phone companies were big enough and powerful enough to be a threat. In 1982, IBM endorsed OSI and suddenly SNA had seven layers instead of five. But, the marketing hype said, while OSI was good for data transfer, it didn't do network management. It was well understood that it was impossible to convert a hierarchical architecture to be peer, but IBM wasn't ready to give up that quickly. It needed time to map a path through this two-front minefield. Not only did IBM build good hardware and have good marketing, they also were masters of electro-political engineering. It then embarked on a strategy of stonewalling the development of network management within OSI (which, given the complexity of the issues, was not hard).

It would be hard for anyone to get a management protocol effort started in OSI with the IBM delegates there to bring up all sorts of spurious details to debate. The IBM standards people were focused on OSI and mainframe networking. Their focus in IEEE 802 was 802.5 Token Ring. From their point of view, those software architect types in 802.1 didn't require a lot of attention. The physical layer was where all the action was. (At this time, a data-comm textbook would devote 300 pages to the physical layer and 50 pages to everything else.) It is not surprising that the IBM Token Ring guys weren't watching too close, if they even knew they were there. So what if there were these "kids" from a LAN company working on it. There were a lot of young engineers jumping on the Ethernet LAN bandwagon. Let them play around developing a LAN management protocol. (The IBM Token Ring guys attending 802 didn't make the connection that these "kids'" manager was the *rapporteur* of the OSI reference model and chief architect for a network management product in development and that most of what they were taking into 802.1 was intended for elsewhere. The IBM Token Ring delegates never mentioned the activity to the IBM OSI delegates. And of course, I was highly critical of the ongoing OSI network management work, seeming to support IBM's "concerns," which wasn't hard with IBM mixing it up, when in fact, it was intended to throw them off the scent. I knew that if I took the lead in the IEEE project and the OSI Reference Model, it would get IBM's attention. Hence, IBM was caught flat-footed when the IEEE management protocol was brought into OSI as a fully complete proposal. No development required. Ready to go to ballot. IBM tried all sorts of procedural maneuvers to stop the introduction, but to no avail. There was too much support for it. This broke the logjam on network management, after which the work proceeded at a good pace.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 126

However, experience with the IEEE protocol in mid-1980s showed that although the obvious set/get approach was simple and necessary, it was not going to be sufficient. It was in some sense too simple. It could take a lot of request/responses to get anything done. So, there was already an effort to extend management protocols to do sequences of operations at once. When the IEEE protocol was introduced to OSI in 1985, object-oriented programming was just coming into wide use. The IEEE protocol was generalized to what became the *Common Management Information Protocol* (CMIP). The new features of CMIP were centered on making the MIBs object oriented and including the concepts of scope and filter. MIBs almost always have a tree structure, sometimes referred to as a "parts explosion" or "bill of materials" structure. Scope and filter allowed a request for attributes to specify how much of the tree to search (scope) and filter by a simple relational expression. With CMIP underway, work developing MIBs for all the OSI layers and protocols, as well as definitions for various management applications (such as configuration, accounting, performance, and so on), proceeded apace.

Initially, the intent was to apply scope and filter to both sets and gets. However, it was pointed out that the nature of applying scope and filter to sets was not invertible. (In general, there is no inverse for expressions of the form "for all *x,* such that <relation exp>, replace *x* with *y.*") Hence, scope and filter were restricted to gets only. This is much more powerful than SNMP's GetNext or GetBulk capability and takes less code to implement than lexigraphical order in SNMP.

Somewhat later but overlapping in time, the IETF began work on network management. There were two efforts in the IETF: SNMP, which was essentially equivalent to the IEEE 802 management protocol; and *High-Level Entity Management System* (HEMS), an innovative approach to management protocols that was object oriented and based on a pushdown automata model similar to Postscript (RFCs 1021,1022, 1023,1024; 1987). For some inexplicable reason, the IETF chose to go with the backward-looking SNMP, which turns out to be *simple* in name only. Both CMIP and HEMS implementations are smaller. In addition, SNMP adhered to the "everything should be connectionless" dogma. This decision limited how much information could be retrieved and made getting a snapshot of even a small table impossible. SNMP also limited the nature of the unsolicited event so that devices had to be polled. This decision is hard to explain. In the ARPANET, a proposal to do polling for anything would have been literally laughed out of the room as "old mainframe think." Why it was acceptable at this juncture given that it clearly doesn't scale and networks were growing by leaps and bounds is mystifying. Furthermore, it made little sense. One polls when there is a reasonable expectation of data on each poll. Terminals were polled because most of the time they had characters to send. If new

04_2422_ch04.qxd 4/9/08 1:09 PM Page 127

data is infrequent, event driven makes more sense. Polling for network management would seem to imply that it was assumed most devices would be failing a significant fraction of the time or at least experiencing exceptions. A strange assumption! Dogma was allowed to ride roughshod over the requirements of the problem.

That was not the worst of it. Just as IBM had realized that network management was the key to account control, so did the router vendors. A standard network management protocol that could manage any device from any manufacturer would make routers interchangeable. Hence, some router vendors immediately said that SNMP was good for monitoring but not for configuration (because it was not secure). This was a curious argument. From an academic point of view, it was, of course, true. SNMPv1 security was weak. But practically, SNMP was encoded in ASN.1 (an encoding scheme its detractors delighted in pointing out was overly complex and in moments of exaggeration likened to encryption), whereas the router vendors did configuration over an ASCII Telnet connection protected by passwords sent in the clear! Almost no one had an ASN.1 interpreter, but every PC on the planet had a Telnet program. Practically, SNMP was far more secure than the router vendors' practice at the time. Oddly enough, the IETF and the customers were taken in by this ruse.[14]

This turn of events led to the hurried and botched development of SNMPv2. A small group tried to write the entire new version and then ram it through the IETF with little or no change. This is a well-known recipe for disaster in consensus organizations, and this was no exception. When it was all over, none of the original authors were speaking to each other. New factions had developed around different variations of version 2 (eventually, after a cooling off period, leading to SNMPv3). But by that time, the ruse had achieved the desired affect, and SNMP was viewed as being for monitoring, and the router vendors laughed all the way to the bank.

The other problem in network management was and is the proliferation of MIBs. One would think that each layer could have a MIB that is largely common across all protocols used in that layer. (Actually, it can be done.) OSI actually made a halfway attempt at commonality but did not go as far as it could have, and as it went down in the layers, the commonality disappeared quickly. In the IETF, it was a complete free for all, with a proliferation of MIBs for each technology, for each protocol, and in some cases for different kinds of devices, before they realized there was a problem. And although they did try to introduce some commonality, later the horse was out of the barn by the time the concepts were in place. It is clear that the key to simplifying networking must be in creating greater commonality and consistency across MIBs. What is also clear is

[14] This appraisal of SNMP is not based on 20/20 hindsight but of observations at the time of the events and, in some cases, the events were predicted.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 128

that it is not in the vendors' interest to create that commonality and consistency, at least, as they perceive the market today. If it is going to happen, it is going to have to be built in to the fabric of the architecture.

All in all, the decision to pick SNMP over HEMS has probably set back progress easily by a decade, cost network providers countless millions of dollars in lost productivity and overhead, and slowed or prevented the deployment of new services and applications. By retaining account control and routers not being interchangeable has also increased costs of operations and of capital equipment (all in all, reducing competition). The shortsightedness of this decision ranks along side the decision to do IPv6 as clear steps backward rather than forward. The failure to explore the HEMS approach, especially considering possible cross-fertilization of ideas from CMIP, represents a major lost opportunity.

But what does network management tell us about upper-layer architecture? Actually, quite a bit. Prior to tackling network management, applications had been seen as requiring unique protocols for each application: Telnet, FTP, RJE, mail, and so on. Experience with network management showed us that the variety is in the object models. The range of operations (protocol) was actually fairly limited. Not only are the protocols for management applicable to other "network-like" systems (for example, electric grids, gas, water distribution, airlines), but also a host of other applications can use the same basic object-oriented protocol for performing actions at a distance. It was clear that most application protocols could be viewed as a limited number of operations[15] (protocol) on a wide range of object models (operands or attributes). (It appears that the application protocols come in two forms: request/response and notify/confirm.) Hence, the number of application protocols is really quite small. So in the end, it begins to look like Alex may have been pretty close on the number of application protocols after all; they just weren't Telnet and FTP!

But the problem of the protocol operations being too elemental is still with us. Scope and filter were found to be useful in practice. It is unfortunate that we were not able explore what might have come from using HEMS, which might well have represented that "middle ground" between the elemental "Turing machine-like" structure of protocols, such as SNMP and CMIP, and a language with a full control structure, such as Java, Perl, or XML. It may appear that not

[15] The operators seem to be set/get, create/delete, start/stop, and event, and the protocols seems to be of either a request/response form or a notify/confirm form, where the later could be symmetrical.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 129

many situations fall into the middle ground of needing more complex management command sequences but being too resource limited to support a full language. But then, that might be what the problem is trying to tell us. On the other hand, having not really looked for useful models in that middle ground would tend to lead us to that conclusion. This is an area that deserves exploration.

HTTP and the Web

The development of SNMP took place in the late 1980s, but the application protocol that had the most impact on the Internet did not come from the IETF or from computer science research. HTTP and the development of the World Wide Web came from a completely different community. The history of the Web has been chronicled many times. Developed by Tim Berners-Lee at the particle physics center, CERN in Geneva, it languished for awhile as "just another Gopher" until Marc Andreesen at *National Center for Supercomputer Applications* (NCSA) at the University of Illinois had the idea to put a better user interface on it. A major effort at NCSA was developing tools to facilitate the visualizing of huge amounts of data generated by the center. The rest, as they say, is history.

The Web has been so successful that to much of the world, the Web is the Internet. Our interest here is in the new structures and requirements the Web brought to the Internet. There were basically three:

1. The necessity to distinguish the application from the application protocol
2. The effects of many short transport connections on network traffic
3. (and related to the first) New requirements for addressing deriving from the need to deal with the load generated by many users accessing the same pages

To a large extent, the Web caught the Internet flat-footed. Here was a fairly simple application (at least on the surface), but it was suddenly creating huge demand requiring more sophisticated distributed computing support, which the Internet did not have. The

The First "Web"

This was not the first use of hypertext on the Net. In the early 1970s, Doug Englebart had used the ideas in the *oNLine System* (NLS) for which he also developed the mouse. NLS was also used for the *Network Information Center* (NIC) and was in many ways more sophisticated than the Web, but it did lack graphics and required an entire Tenex to support it. In RFC 100, minutes of a 1970 NWG meeting, it mentions that NLS was ported to an IMLAC, an early graphics terminal. In other words, a personal computer with a mouse running the Web! Again, if ARPA had continued to pursue applications with Moore's law, we might have had the Web a decade sooner.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 130

Web was put in the position a bit like an application finding that it has to do its own memory allocation. Let's quickly review how the Web works, what happens when there are lots of users and lots of pages, and then consider what this implies for the Net.

When a user wants to access a Web page, she types a *Universal Resource Locator* (URL) to her HTTP client (or browser), and it is sent using the HTTP protocol over TCP to the HTTP server. DNS is used to resolve the URL to an IP address. The response returns a page of formatted text and graphics and containing many URLs to other objects on the page. All of these URLs may point to information on different systems. The browser then accesses each URL with a separate request over a new TCP connection. The responses provide more objects to populate the Web page, as well as URLs, which can be selected to access new pages. The Web is a stateless (connectionless) application for accessing and displaying Web pages. Everything is driven by the client. While links may be to any server in the Internet, it is often the case that there is considerable locality in accessing a page. Clearly, as the number of users increases and the popularity of certain Web sites increases, the load caused by large numbers of TCP connections begins to take its toll on both the servers and the network.

The first thing we notice is that the Web user is accessing the Web application. There may be thousands of Web sites on the same host all using HTTP. This is not like Telnet, FTP, or mail, where there is only one application using one application protocol per host. HTTP is just the vehicle for the two parts of the application to communicate. Here we have the first example in the Internet of the distinction between AP and AE that OSI found was necessary.

The first problem confronting the Web was that there was no application name space for the Internet. It essentially had to create URLs. This was not difficult at first, because part of the URL corresponded to the domain name of the host. The browser would just extract the domain name from the URL and do a DNS lookup, get the IP address, and open an HTTP connection and send the whole URL to the HTTP server. But very quickly, things got more complicated. Suppose the owner of the application wants the Web server to be on a server somewhere else?

For example, suppose gaslesscar.com company wants to reserve its own server to support corporate activity and outsource its Web site for customers, www.gaslesscar.com, to a service provider. What does it do? Suppliers still need to connect to gaslesscar.com, whereas customers should be directed to the host of the service provider. The company needs to retain the name recognition. It can be kludged to work by letting the customers first connect to the corporate

04_2422_ch04.qxd 4/9/08 1:09 PM Page 131

site and redirecting them. It works, but it is unnecessary load on the corporate server, it will cause delays seen by the customer, and it raises security issues and a host of other problems. Clearly, URLs had to be directly supported by DNS, and they were. But by the same token, DNS was not designed for this kind of usage.

Clearly, the Web is going to generate a lot of TCP connections and large volumes of data. TCP had been designed on the assumption that connections might be short, a few minutes or even seconds, to very long, a few hours, and that hosts might be setting up new connections on an average of every few seconds (at most, several hundred connections per hour). Connections lasting milliseconds and thousands per minute were not foreseen. This by itself was not a big problem, but the problem of balancing this with applications with much different usage patterns was.

HTTP opens many very short connections and sends only a few bytes of data, even though the total amount of data sent for a page is equivalent of a reasonably sized file transfer. Each HTTP connection never sends enough data to be subject to TCP congestion control, whereas other traffic with longer-lived connections is. This allowed HTTP traffic to unfairly grab an unfair share of the bandwidth, the "elephants and mice" problem. This was the first real encounter with applications that generated heterogeneous and incompatible traffic. The short connections also put a resource strain on TCP control blocks, which cannot be reused for a very long time relative to the time they are used. HTTP1.1 solves both of these problems with persistent connections. By using a TCP connection for more than one request/response interaction, it reduces the turnover in *traffic control blocks* (TCBs) and transfers enough data over the connection to ensure that TCP congestion control comes into effect and HTTP traffic only gets its fair share.

Because the Web page will follow Zipf's law, the same pages will be retrieved for many users. This leads to a requirement for Web caching, either within the client, within the client's subnet, within their ISP, or by a Web hosting service. The client HTTP is modified to send all requests to a cache site regardless of what is indicated in the URL. If it is not there, it may be forwarded to another cache or to the site indicated by the URL. The server can offload subsequent accesses to the page by returning a response with links relocated to another server.

In the case of the Web, a way was found to jury-rig existing structures with minimal additions to meet the Web's needs. Will it be possible to find a band-aid for the next application? How many applications aren't being developed because the structures are not there to support them? When do the band-aids

04_2422_ch04.qxd 4/9/08 1:09 PM Page 132

begin to interfere with each other? This is all arithmetic. What is the algebra? What is the right way to accommodate load leveling and migrating applications in a network? All of these are problems we will encounter with other applications. We have solved the problem for the Web, but we have not solved them for the Internet. Or as the press believes, is the Web the Internet?

Directory- or Name-Resolution Protocols

Another interesting class of distributed applications is the directory- or name-resolution protocols. The need for these was recognized quite early. If one were building a "resource-sharing network," a phrase that appeared in many early networking papers, one would need a means to find the resources. Because operating systems were the guiding metaphor from the early 1970s, the obvious solution was some sort of "directory," a service that could tell the user where things were. The earliest attempt was the XNS Grapevine system (Birrell, A. et al., 1982) developed by Xerox PARC in the late 1970s and extended in Clearinghouse (Oppen and Dalal, 1983). Other commonly known, similar services include DNS, X.500, Napster, Gnutella, distributed hash tables, and so on. All of these have the same basic elements and the same structure with a few variations depending on size, degree of replication, and timeliness of updates. Basically, this is a distributed database problem (see Figure 4-8).

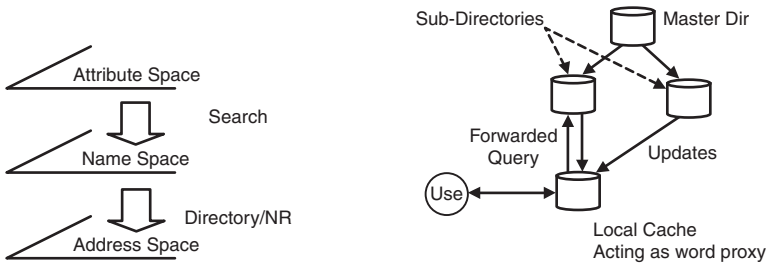

Figure 4-8 Elements of a directory- or name-resolution system.

Name-Resolution Systems

A *name-resolution system* (NRS) consists of a database, usually distributed, that is queried by the user. The database maintains the mapping between two name spaces: one that names what is to be located, and one that names where the object is. (To facilitate the description, I refer to the "what" names as "names" and to the "where" names as "addresses." We will consider names and addresses in more detail in the next chapter.) In some cases, a third name space of attributes and a mapping to the application names is used. This allows the

04_2422_ch04.qxd 4/9/08 1:09 PM Page 133

user to query not just on the name of the object, but also on attributes of the object. In general, this is a separate NRS. This is what *search engines* do. Name-to-address mapping is what *directories* do. Some services replace the application name space with the attribute name space—in effect, treating a string of attributes as the name of the object. This is not always a wise decision. A string of attributes may identify one, none, or many elements, and furthermore, the object(s) ultimately referred to by the string may change with time. This yields a potentially large number of names for the same object. If the service needs to maintain accountability and track access to the objects (as may be required for security purposes), the attribute searches should resolve to a unique name so that accountability is maintained. (Because objects may move, it is necessary to resolve to a name rather an address.)

NRS Structures

The database for an NRS may be centralized, cached, hierarchical (that is, a tree), or a combination of the cached and hierarchical (usually depending on size and the degree of timeliness required). None of these needs to be mutually exclusive. It is straightforward and common for these systems to evolve from one form to another. For small or non-critical databases, there is often just a central database. Cached databases are introduced usually as a local cache to improve performance and reduce overhead at the cost of partial replication. The caches usually have a flat organization but can easily evolve into a hierarchy. With larger databases, the hierarchical structure of the name space is generally exploited to create subdatabases responsible for subsets of the database. The degree of replication (that is, caching) among the resulting tree of databases will depend on the application. For NRSs with significant load/availability requirements, the databases may be fully replicated. It is easy to see how some sites become known for keeping larger portions of the database. Over time, the structure may be regularized with known sites for subsets of the database. There is no requirement with a hierarchical structure that every node in the name space has a distinct subdatabase. These are usually based on size or organization boundaries. Flexibility is more important than following rules. Queries can be sent to the appropriate site with knowledge of the correspondence between the name structure and the appropriate subdatabase.

When Is a Name a Query?

This brings up an interesting convergence that makes distinguishing a query and a name difficult. And you thought they were clearly different! Let me explain.

The OSI Directory Service, also known as X.500, was probably the first to run into this conundrum. X.500 was initially intended to provide the mapping between application names and their network addresses. But as the work developed, it was made a more general-information repository for finding resources of almost any kind. They originally proposed the use of "descriptive names," which was an arbitrary sequence of attribute/values. This looked very similar to a query. A sequence of attribute/value pairs is nothing more than a query in disjunctive normal form. When it was pointed out that accountability was

continues

04_2422_ch04.qxd 4/9/08 1:09 PM Page 134

Two protocols are required for this service:

1. A request/response protocol for querying the database
2. A protocol for distributing updates to the database, if it is distributed

The user sends a query to a member of the NRS. If the member can satisfy the query, it returns a response to the user. If not, an NRS may be designed to respond to the user either by referring it to another database in the system or by forwarding the query to another NRS database. Depending on the level of reliability the NRS tries to maintain, if the query is forwarded, the forwarding database may act as a proxy for the user, keeping track of outstanding requests, or the database forwarded to may respond directly to the user. This latter approach may complicate security.

With a replicated distributed database comes the problem of updating the copies when there are changes. This may either be initiated by the copy (the request form) or by the member when a change occurs (the notify form). The frequency of updates varies widely depending on the frequency of change and the degree of timeliness necessary. Cached systems will tend to age and discard their caches, or perhaps distinguish often used entries and update them. Updates may be done periodically, or initiated by significant events (a new member or a member disappearing), or both. Hierarchical systems tend to periodically update in addition to responding more immediately to change.

required for application names, they retreated to an ordered sequence of attribute/values, called a "distinguished name," which satisfied the requirement of accountability. Distinguished names still have a strong similarity to queries and indicate that the traditional hierarchical pathname is merely a distinguished name where the attributes are "understood." None of the mathematical or philosophical work on naming addresses the difference between a name and a query and whether it matters.

This is one of the things that science and mathematics is supposed to do. When a close similarity is uncovered between two concepts that were thought to be quite distinct, it warrants a more careful consideration. It might turn out to be nothing. Or it may lead one to discover that we have been looking at a collection of problems in entirely the wrong way and lead to reformulation and simplification.

Table 4-2 *Characteristics of NRSs*

Database Organization	Centralized
	Hierarchical
	Cache
	Cache/Hierarchy
Query	Referral
	Forward
	Proxy
Update	Periodic
	Event-driven
	Combination

04_2422_ch04.qxd 4/9/08 1:09 PM Page 135

When new sites come up, they register their information with a local or near-local service, and it is propagated from there as necessary. When sites go down or a resource moves, the contents of the database change, and these changes must be propagated. This is generally accomplished by either the child services requesting an update periodically or by an affected database notifying its neighbors of the changes.

DNS, X.500, or Grapevine are structured like this, choosing specific policies for querying, database organization, and updating. They started from a centralized or cached service and grew to be more hierarchical. Napster (centralized) and Gnutella (cached) basically do the same thing for finding files rather than applications or hosts. Another P2P approach that curiously has gotten a lot of attention is based on *distributed hash tables* (DHTs). This approach differs from the earlier approach only in how a hierarchical application name space is generated. With DHTs, the name of the resource, usually the URL, is hashed. The hash creates a number of buckets where resources may be stored. The sites where resources are stored are arranged according to the subset of the hash space they are responsible for. The resource or information about the resource is stored at the site indicated by the hash of the name. The sites organize themselves in hash order. Using a hash means that the resources will be evenly distributed across the databases. The user accesses a resource by searching in hash order. This may be beneficial in terms of load leveling, but it destroys any locality that may have been embedded in the original name and would have allowed the sites to do some intelligent caching. To add insult to injury, attempts to address this shortcoming have been proposed by adding additional mechanisms to treat the hash value as a hierarchical name (!). A user uses a hierarchy imposed on the hash to find the site with the resource. Essentially the same way, a DNS lookup uses the structure of the domain name to find the correct DNS server. This somewhat begs the question, why not use the original name, which in most cases was already hierarchical and very likely reflected locality to a large degree. It would seem that if one wants to optimize the use of Web pages or files using such name-resolution protocols, one would have much more success using operating system paging as an analog.

What About Peer-to-Peer?

Indeed. What about peer-to-peer? There has been quite a fad surrounding P2P. Claims that it represents new ideas in distributed computing. Much of the hype has centered on its application to music "sharing," also known as intellectual property theft. P2P appears to be the poster child for just how far networking has fallen as a science.

First and least, *peer-to-peer* is bad English, equivalent to *irregardless* as words to show your illiteracy. Peer communication has always said it all. Communication is always with another entity. Peer denotes the nature of the relation. Peer-to-peer is simply redundant. Computer science has never been known for its writing, but there is no point making our illiteracy a neon sign.

"Okay, so you're pedantic," you say. "Whatever you call P2P, it introduces new concepts to networking: systems that are both client and server. Transfers don't have to go through a third party." This has been the case since the day the ARPANET was turned on. For more than a decade, until workstations and PCs became prevalent, most hosts on the Net were both clients and servers. Communication has never been required to go through a third party. One queries DNS and communicates directly, not through DNS. By 1973, even FTP supported third-party transfers! A could transfer a file from B to C without going through A. There is nothing in the protocols or the architecture that imposes such constraints. Any such limitations are only

continues

04_2422_ch04.qxd 4/9/08 1:09 PM Page 136

continued

in the minds of those using the Net. This has more in common with science in 17th-century China, where knowledge of previous discoveries was lost and rediscovered. It is easier to forgive them taking centuries to lose knowledge, whereas it takes only networking a few years.

"But there are these new algorithms for finding where things are." Go back to reading the text. They are just variations on the name-resolution and directory systems we have been proposing and building for a quarter century. To add insult to injury, none of the so-called P2P protocols are peer (symmetric) protocols. They are client/server (asymmetric) protocols.

Why has no one pointed out that not only is there nothing new about P2P, but also what there is, isn't that good? As computer scientists, we should be quite concerned when the primary claim of a fad in our field is for illegal pursuits and is at the same time an embarrassment to our intelligence.

What Distinguishes the Upper Layers

On the surface, distinguishing the upper layers from the lower layers has always been easy. But when the details were examined, it was most often a case of "I can't tell you what it is, but I know it when I see it." It was difficult to find a set of characteristics that was better than "that which is above transport" (a definition used even recently). But as our understanding improved, it appeared that were two characteristics that distinguish the upper layers from the lower layers, regardless of what those layers are:

1. In the upper layers, processing is in units that have semantic significance to the application (that is, incur the least processing overhead/effort for the application); whereas in the middle layers, processing is in units best suited to the resource-allocation requirements; and in the lower layers, the characteristics of the communications media or network technology are dominant.

2. In the upper layers, addressing is location independent. In the lower layers, addressing is location dependent. Or perhaps more precisely, whereas lower-layer addressing is based on the topology of the network, upper-layer addressing is usually based on a sort of "semantic" topology (for example, all network access applications, all software development applications, and so on).

In a sense, the characteristics of the media percolate up, while the characteristics of the application seep down, and both are "filtered" along the way with the differences reconciled when they meet in the middle.

The overriding shift when we move from the lower layers to the upper layers is that semantics becomes important, whereas it was consistently ignored in the lower layers. This is not to say that upper-layer protocols deal only with the semantics and that user-data. (That is, data that is passed transparently without interpretation does not appear in the upper layers; it does.) It is just that the boundaries of the PDUs are not chosen arbitrarily with respect to the application but are chosen to be significant to the application. A couple of simple examples will help to illustrate the point.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 137

Semantic Significance

In the lower layers, message or PDU boundaries are chosen to accommodate the constraints of the media or networking technology. The requirements of the application are rarely noticed (and even less the deeper in the layers one goes). This changes in the upper layers, where "record" and "transaction" boundaries of the application become important. Not only is everything done in terms of these boundaries, but also in most cases, nothing can be done if one does not have a complete "record" or "transaction." Thus, we find checkpoint-recovery protocols that work on records or two-phase commit protocols that work on record or transaction boundaries, determined by the application.

This lesson was learned early in the development of FTP. Checkpoints in FTP are inserted anywhere at the discretion of the host sending the file (the stream model asserting itself). One of the major dichotomies between host operating systems was (and still is) whether their file systems are stream or record oriented. It was noticed that when a stream-oriented host was transferring a file, it inserted checkpoints every so many bytes. If it were transferring to a record-oriented host, the record-oriented host could only recover a failed transfer on record boundaries. If the number of bytes between checkpoints were relatively prime with respect to the record length, the record-oriented host could only recover by transferring the whole file. Some early file transfer protocols made this problem worse by having checkpoint windows (similar to the window flow-control schemes in transport protocols). The sender could only send a window's worth of checkpoints without a checkpoint acknowledgment (not a bad idea in and of itself). The transfer then stopped until one or more checkpoint acks were received. In this case, it was possible for the file transfer to deadlock. The receiver couldn't ack because the checkpoints were not on a record boundary, and the checkpoint window prevented the sender from sending more data until a checkpoint was ack'ed. The fundamental difference was that for a stream-oriented host, the only semantically significant points in the file were the beginning and the end of the file. By inserting checkpoints arbitrarily, it was imposing a policy that was more lower layer in nature than upper layer. The NVFS failed to impose a necessary property on the checkpoint-recovery mechanism.

Similarly, problems could arise in performing mappings between different syntaxes if they were not isomorphic. To take a trivial example, consider the mapping of an 8-bit EBCDIC character set to 7-bit ASCII, a common problem for early Telnet. Early in the ARPANET, new translation tables were deployed in the TIPs, and it was found that it was not possible to generate the line-delete or character-delete characters for systems that used EBCDIC. Nonisomorphic translations must ensure that the semantics important to applications are preserved. While sometimes surprising, translations for character sets are relatively

04_2422_ch04.qxd 4/9/08 1:09 PM Page 138

easy to accommodate and a simple example of the problem. However, when mapping operations on file systems or other such complex operations between different systems, ensuring the invariance is far more subtle. (The point is not so much that a file system is complex, but more that it doesn't take much complexity to create problems.) It can be much less obvious what effects of the operation on a particular system a program is relying on for its successful operation.

The canonical form provides the basis for addressing problems of this sort. The model of the application defines not only its structure, but also the operations that can be performed on that structure. The canonical form, in effect, defines the invariant properties of the operations that must be preserved when mapping from one local system to the canonical form. The canonical form defines the transfer semantics in much the same way that the concrete syntax defines the transfer syntax.

Today with the network being such an integral part of applications, it is less likely that these sorts of problems will occur as often. However, they will turn up as applications that were never intended to be networked (developed in different environments) find that they have to communicate. In these cases, it won't the simple applications, such as terminals and file systems, but complex business applications where teasing out the semantic invariances will be much more difficult.

Location Independence

The difference in addressing had been recognized since some of the earliest research on networking. Very early (circa 1972), it was recognized that it would be highly desirable to allow applications to migrate from host to host and to accommodate such migration would require applications to be named such that their names were independent of their location (that is, what host they were on) (Farber and Larson, 1972). There was a tendency to refer to this as upper-layer "naming" as distinguished from lower-layer "addressing" as recognition of this difference. However, this is not really the case.

Although an address is a name, a name is not necessarily an address. Addresses are assigned to an object so that the object is easier to find. The algorithm for assigning addresses to objects defines a topology (in most cases, a metric topology). Therefore, addresses always represent points in a topology,

04_2422_ch04.qxd 4/9/08 1:09 PM Page 139

whereas names are merely labels. Rather than say a name space is a flat address space, In most cases, a flat name space is simply an address space out of context.

If one considers carefully the nature of "naming" as developed in metamathematics and mathematical philosophy as distinguished from addressing, one is led to the conclusion that in computer science and especially networking, virtually all names are used for *finding* the object. All of our naming schemes are schemes constructed to make locating an object easier (for some value of *easy*) in some context, whether spatial or semantic. (A file system path*name* is structured to make finding the file in our collection of files easy. We use directories to group related files under meaningful names. It isn't addressing in a spatial sense, but it is addressing, rather than just naming.)

In the lower layers, geographical or network topology characteristics are used as the organizing principle for locating objects. In the upper layers, other characteristics are used for locating applications that are seldom related to location. Unfortunately, unlike the lower layers, a crisp set of characteristics commonly used to organize the address space has not yet emerged for application addressing schemes. The most one can say is that characteristics of the applications are used. (One might say that application addresses are organized more by "what" or "who" rather than "where.") In some cases, schemes have reverted back to location-dependent characteristics. However, such schemes preclude any migration of applications being transparent to a user.

Too Fine a Point? …Maybe

In many cases, it will seem that I am merely splitting hairs, and that *is* always a danger. But in many aspects of network architecture, especially addressing, these subtle distinctions have a profound effect on the outcome, a sort of "butterfly effect" in logic. These distinctions can make the difference between a system in which it is easy and efficient to do things and one that is cumbersome and inefficient and even a dead end.

As mentioned earlier, directory schemes, such as X.500, in pursuit of "user-friendly names" initially proposed a "descriptive naming" scheme consisting of the intersection of an arbitrary list of attributes and their values. This characterization begins to confuse the difference between a query and a name. These attributes, in effect, define a topology within which the application is "located." Applications in the same part of the directory are in the sense of this topology "near" each other, at least from the point of view of the application naming space, while almost certainly not from the topology of network addresses.

04_2422_ch04.qxd 4/9/08 1:09 PM Page 140

Conclusions

Although I cannot claim that there has been a thorough investigation of upper layers over the past 35 years, there has been a lot of work. Mostly, we have to draw our conclusions based on generalizing from specific applications, rather than attempts to construct a theory of the upper layers. We were able to show that the division of labor represented by the OSI upper layers works only for the simplest applications. On the face of it, this would seem to indicate that there is no upper-layer architecture, or if there are upper layers, they have nothing to do with what OSI thought session and presentation were. We can say that there is an application architecture, but there is no common *upper-layer* architecture.

While we can characterize the "upper layers," it does appear that the only structure common to all distributed applications is the distinction made by OSI between the application process and the application protocol. The requirement for a common establishment mechanism was an important step in replacing the stopgap left over from the ARPANET demo. Specific application domains will have more detailed structures, and there are probably common application protocol modules that can be used across application domains, but that is probably about it. We have uncovered some useful techniques for structuring application protocols, and as with anything, we have learned some things not to do. In addition, application protocols essentially reduce to defining the means to perform operations at distance on object models, and we can identify that the fundamental operations are read/write/append, create/delete, and probably start/stop. We only lack a model for the control structure that strings them together. Does this indicate that the essential distinction between upper and lower layers is the distinction between transparent (semantic-free) communication and distributed system structures and programming languages?

After looking at naming and addressing in the next chapter, we take a step back to see whether we can make coherent sense of all of this.