PRENTICE HALL

Patterns in Network Architecture

A Return to Fundamentals

JOHN DAY

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Patterns in Network Architecture

This page intentionally left blank

Patterns in Network Architecture

A Return to Fundamentals

JOHN DAY

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data:

Day, John D., 1947-

Patterns in network architecture : a return to fundamentals / John Day.

p. cm.

ISBN 0-13-225242-2 (hbk. : alk. paper) 1. Computer network architectures. 2. Computer networks—Design. 3. Computer networks—Philosophy. 4. Internet—History. I. Title.

TK5105.52.D39 2007

004.6—dc22

2007040174

ISBN-13: 978-0-132-25242-3
ISBN-10: 0-132-25242-2
Text printed in the United States on recycled paper at Courier Westford, Massachusetts.
Second printing December 2008

Editor-in-Chief
Karen Gettman

Acquisitions Editor
Chuck Toporek

Managing Editor
Gina Kanouse

Senior Project Editor
Kristy Hart

Copy Editor
Keith Cline

Indexer
Erika Millen

Editorial Assistant
Romny French

Cover Designer
Chuti Prasertsith

Composition
Gloria Schurick

Proofreader
San Dee Phillips

To Heinz von Forester,
who taught me how to think
and
To Dite, Kathleen, and Kinmundy,
who gave me a foundation for life

This page intentionally left blank

Contents

Preface

There is something fascinating about science. One gets such wholesale returns on conjecture out of such a trifling investment of fact.

—Mark Twain, *Life on the Mississippi*

The Seven Unanswered Questions

This didn't start out to be a book.

It started out simply as an attempt to distill what we know about networks after 35 years of beating on the problem. What principles, rules of thumb, guidelines, and so on could be distilled from what we had seen independent of politics, and religion, and even the constraints of technology. What could be said with as little qualification as possible? Were there a few constraints that could be introduced that would allow us to do a great deal more? What did we really know that didn't change? What some might call science.

Over the years, I saw ideas go by that had not been pursued, directions taken that didn't seem quite right; sometimes little things, sometimes not so little (but always affected by politics, market interests, group think, and sometimes just the imperfect state of our understanding). But the ideas were points that had the potential to be those subtle inflections on which much bigger things hinged. Usually they were sloughed off with a fatalistic "Awww! Simplifying here would only increase complexity elsewhere." But would it?

As I pursued this seemingly quixotic quest, patterns began to assemble themselves that I had not seen before. Patterns that lead to a major collapse in complexity. The structure of networks turns out to be much simpler than we imagined. There are far fewer protocols. And capabilities such as multihoming, mobility, and scaling turn out to be a consequence of the resulting structure, not complexities to be added. No cumbersome mechanisms are required. The increased orthogonality and regularity of the structure makes the solutions to other problems easier and straightforward. On the surface, what emerged appears not that different from what we had been doing. And upon first reflection, some are likely to think, "Sure, we all knew that." But deeper, it is very different and requires a cognitive shift that isn't always easy to make. And this shift is made more difficult because not all the concepts key to making the transition are common knowledge.

In addition to just codifying principles and rules of thumb, there were a few key unsolved problems that were at the crux of a better understanding, problems that needed the kind of unfettered thought impossible in the heat of product development or standards deliberation.

I have often said, only half jokingly, that "the biggest problem with the ARPANET was we got too much right to begin with." Meaning that for a project for which there had been no prior experience, for which there was considerable doubt it would even work, there was some brilliant work and some brilliant insights to the point that it was "good enough," and there was no overwhelming need to address the problems it did uncover (which really just says we weren't pushing hard enough on the edges). One of the most striking phenomena in the early ARPANET was the number of times that when presented with what appeared to be a dichotomy, an "oil and water" problem, they found an elegant simple synthesis that wasn't either extreme but in which the extremes were "merely" degenerate cases (and that at the same time told us something we hadn't previously understood).[1]

As one would expect with any first attempt, some were mistakes, some things were unforeseen, some shortcuts were taken, some areas went unexplored, and so forth. But even so, the network worked much better than anyone had any reason to expect. Almost immediately, the ARPANET went from a subject of research to a necessary and useful resource.[2]

During its development, a constant guiding metaphor was operating systems. We always looked to operating systems to provide insight to the solution of problems and for what should be built. (Many involved in that early work have attributed the success of the ARPANET in great part to the fact that it was built by people with operating system, not communications, backgrounds and have lamented that it is no longer the case.) By 1974, with the network essentially operational, there was great excitement about what could be done with the Net.[3] (It was really a small version of the excitement we saw in the early 1990s

1 I say "they" because I was just a "junior" grad student at the time, and while I was there, I can take no credit for these insights but could only hope that I learned from watching them emerge.

2 This book isn't yet another history of the Net (although there is a lot of that here). I have found that one cannot give an honest explanation of why things are the way they are based solely on technical arguments.

3 Contrary to recent characterizations that we saw the use of the Net as "conversational," nothing could be further from the truth. We saw it as a heterogeneous resource-sharing facility, and that was the impetus for experiments and production distributed systems such as Englebart's NLS, National Software Works, CCA's Datacomputer, the NARIS land-use management system that utilized a distributed database spread of over the United States invisible to its users, processing ERTS satellite images across multiple systems, heavy use of Rutherford High Energy Lab and the UCLA 360/91 by U.S. particle physicists, and so on, all prior to 1976.

when everyone else discovered the Net.) However, there were some outstanding problems that we knew about; some expediencies we had taken (as one must always do in any real project) that needed to be fixed. They were as follows:

- **Replacing NCP.** Probably foremost in our minds coming out of the ARPANET was the realization that the Host-Host Protocol would not scale to a large network, where *large* was a few thousand hosts. The separate control channel shared by all hosts was a bottleneck. The protocol was overly complex and tied a little too closely to the nature of the IMP subnet. *What sort of protocol should replace it?*

- **Cleaning up the structure.** Given that operating systems loomed large in the early thinking and Dijkstra's THE paper (1968) was only few years old, it was natural that layering was used to organize the functionality. However, it is difficult to say that the initial implementation of the ARPANET was layered very cleanly. There was still a lot of beads-on-a-string in the design.[4] The interactions of the Host-Host Protocol and the IMP subnet were less than clean. But by 1974, the idea of physical, data link, network, and transport layers—probably best reflected in the implementation of CYCLADES with its clean separation of CIGALE and TS—was becoming well accepted. Beyond that, there was less certainty. And later, we would find that the lower four layers weren't quite "right" either but were a bit more complicated. But we couldn't say we had a good understanding of what layers were.[5] *What was the right architecture for heterogeneous resource-sharing networks?*

- **The upper layers.** We had just scratched the surface of what applications could be developed. We had three basic applications, once again using operating systems as our guide. We simply replicated the services of an operating system in a network. One we nailed (Telnet); one needed more work (FTP); and one we blew (RJE). Not a bad record. There was a general sense that there was more "structure" in the upper layers we had not yet been able to tease out. Even though some thought that Telnet and FTP were all you needed,[6] some people had all sorts of ideas for other applications. We needed a better understanding of what applications would be

4 This is not an uncommon state of affairs in science that the first step in the transition from one paradigm to another still has a foot in both. "Beads-on-a-string" refers to the phone company model of networking, as exemplified by X.25, ISDN, ATM, and MPLS, that existed prior to 1970 and still exists today.

5 Actually we still don't, as textbook authors like to point out (and if the ongoing architecture discussions are any indication).

6 Mail was two commands in FTP.

useful, how the upper layers were structured, and how they worked with the rest of the system. And as it would turn out, this is a place where our operating system model failed us. These first three applications are all examples of a special case. Oddly enough, this might have been a critical juncture in the development of the Net…or lack thereof. *What did the upper layers look like?*

- **Application names and directory.** Early in the development, the model of operating systems told us that we should have application names and network addresses. As with operating systems, application names would be location independent, whereas addresses would be location dependent. In fact, I remember my mild disappointment when it was announced that well-known sockets would be used as a stopgap measure, rather than defining application names and a directory. It was understandable. Coming up with a naming scheme and building a directory would have taken considerable time. We had only three applications and only one instance of each of them in each host. Application names and a directory weren't really *needed* immediately. Eventually, we would have to go back and do it right before there were too many applications. *What did naming and addressing look like in networks?*

- **Multihoming.** In 1972, Tinker Air Force Base joined the Net and took us at our word that the Net was supposed to be robust. They wanted redundant network connections. Upon hearing this news, I distinctly remember thinking, "Ah, great idea!" and a second later, thinking, "O, *#@*, that isn't going to work!" By making host addresses IMP port numbers (i.e., naming the interface not the node), the routing algorithm couldn't tell that these two addresses went to the same place: our first really fundamental mistake.[7] *But* the solution was immediately obvious! Using the operating system model, it was clear that we needed a logical address space over the physical address space. We needed separate address spaces for nodes and for interfaces. The only trouble was it wasn't clear what these address spaces should look like. It was well understood from operating systems that naming and addressing was a hard problem fraught with pitfalls. Get it right and many things are easy; get it wrong and things are hard, inefficient, and maybe impossible. And we knew the difference between getting it right and getting it wrong could be subtle. *We needed to proceed carefully. What was the nature of this "logical" addressing?*

---

7 Well, not really. It would be a mistake if supporting redundant connections had been intended, but it hadn't. It was hard enough just building a network that moved data. But this is an indication of how quickly the Net began to be considered "production."

- **Location-dependent addresses.** And furthermore, it wasn't at all clear what *location dependent* meant for network addresses. It was a simple problem in operating systems. Location dependence of memory addresses was easy and well understood. It was also well understood for cities built on grids. But data networks were seldom regular grids. What location dependent meant in a general mesh network without being route dependent was far from clear. It couldn't be tied to the graph of the network because that changed too often. It needed to be some sort of abstraction of the graph that indicated where without indicating how to get there. *But how to abstract an arbitrary graph was less than obvious. What does location dependent mean in a network?*

- **Adopting connectionless.** The ARPANET was primarily a connection-oriented network. The ARPANET IMP subnet had more in common with X.25 than with IP. This was a reasonable conservative choice for a first attempt, when we had no idea how it would actually work or how a network was supposed to be built and a somewhat built-in assumption that the network had to be as reliable as possible. Experience with reassembly, flow control, and such showed that a tightly controlled deterministic network had major problems. The insight that less control (less reliable) would be more effective came as an intriguing surprise, but an insight that made a lot of sense. The experience of CYCLADES with the use of connectionless datagrams in a network that essentially created reliable communications with unreliable mechanisms was elegant, simple, and convincing.[8] However, a better understanding was needed of how the connectionless model behaved. Because it had been used only at low bandwidth in relatively small networks, a better understanding was needed of how it would work as it was scaled up. After all, it is seldom the case that the pure form of anything works well in the real world. The simplicity and elegance of the new paradigm of connectionless looked promising. It also provided concepts for a replacement for the Host-Host Protocol. We also needed a deeper understanding of the difference between the connection model and the connectionless model. Even with our excitement for connectionless, we had to admit that there did appear to be times when connections made sense. However, I must admit it took some of us a long time to admit that (me included). *What were the properties of the connectionless model and its relation to connections and how would it scale in a production system? Was there a single model that would encompass both as degenerate cases?*

---

8 The connection-oriented packet-switching model is a straightforward, even obvious, approach to the problem, whereas the connectionless model is an inspired shift in thinking.

These were the major issues facing networking as we transitioned from the ARPANET to the Internet. What happened next? Let's consider how the Internet rose to the challenge of these problems.

Converging on TCP

There were basically four contenders for replacing NCP:

(1) **XNS - Sequence Packet**, which was similar to the (2) **CYCLADES TS** protocol. A packet-sequenced, dynamic window transport protocol with multiple PDU types, establishment, release, ack, and flow control. Both XNS SeqPkt and CYCLADES separated the transport and network functions, analogous to TCP and IP.

(3) **Delta-t**, developed at Lawrence Livermore Lab, was a radically new idea in protocols with a more robust timer-based synchronization mechanism that essentially eliminated connection establishment and used separate PDU types for ack and flow control. Delta-t also separated the transport and network functions. And, of course…

(4) **TCP**, a byte-sequenced, dynamic window transport protocol with a single PDU format and control bits to distinguish various state changes. It also allowed the two simplex channels to be released separately. In its initial version, TCP did not separate the transport and network functions.

A few unique features of TCP stirred some discussion:

- The single PDU format was supposed to streamline processing rather than require the additional code to parse several different PDU types. It was expected that this would save both per-packet processing and code space. Given the low speeds of processors, this was a very real concern. At the time, this looked like a move toward simplicity, but with more understanding of protocols it turns out it isn't.[9] In addition, treating the control bits as control bits in the implementation creates more code complexity. The recommendation for current implementations is to treat them as if they were an opcode. In fact, looking at traffic statistics in the Net today, it is clear that syns, fins, and acks are treated as different PDU types (i.e., the number of 40-byte packets).
- The single PDU format also had the advantage of piggybacking acks. Calculations at the time showed that piggybacking reduced overhead by 35%

[9] It is hard to believe, but in 1974, there had been very few data protocols designed, and they all looked very different. More so than they do today.

to 40%. This savings occurred because at the time the vast majority of traffic on the Net was character-at-a-time echoing of Telnet traffic by BBN Tenexes (the then-dominant system on the Net). However, because there aren't many Tenexes on the Net today, the savings today is negligible, well under 10%.[10]

- For a 1974 environment where one size fit all, TCP had marginal advantages in some areas, for others it posed significant burden; for example, bandwidth constraints were still common, making the header size problematic for some environments. Today its advantages have disappeared. Its inability to adapt easily to a wider range of operations are an obstruction to meeting the requirements of a modern network. Delta-t or TS would probably have been a better choice. They were not only well-suited for the environment at the time (delta-t was used for years within the DoE), but both could also have been easily adapted to modern demands without significantly changing their structure.

As shown in Chapter 3, "Patterns in Protocols," the general structure of this class of protocols naturally cleaves into a pipelined data transfer part, loosely coupled with a more general-purpose computational half that requires synchronization for the bookkeeping associated with error and flow control. The single PDU format complicates taking advantage of this structure and complicates making the protocol adaptable to the requirements of different applications, leading to an unnecessary proliferation of protocols. The single PDU format makes less sense. TCP was very much optimized for the characteristics of the mid-1970s.

Why was TCP chosen? There are many reasons. At the time, with the exception of the delta-t synchronization mechanism, the differences among the four protocols were not that great. And overwhelming arguments could not be made for any of these protocols; that is, none was the overwhelming choice. None of the arguments mentioned above was understood then. And, it was expected whatever the choice, it would be used for a few years in this research network and replaced. After all, NCP was a first attempt in building a network. TCP was our first attempt in this new direction. No one expected that we would get it right the first time. At least, one more attempt would probably be needed to "get it right." However, probably the foremost factor in the choice was that the Internet was a DoD project and TCP was paid for by the DoD. This reflects nothing more than the usual realities of interagency rivalries in large bureaucracies and that the majority of reviewers were DARPA contractors.

10 Do the math. Twenty-character input and 40 characters on output were accepted averages for terminal traffic at the time.

Splitting out IP (nothing new for addressing). Splitting IP from TCP seemed a necessity. The transport protocol and IP do very different functions (as will become clear in Chapter 6, "Divining Layers"). The only unfortunate aspect in the creation of IP was that nothing was done about the multihoming problem. IP continued to name the interface. But this was understandable. IP was split out in 1975, soon after the problem was recognized. Although we understood what the multihoming problem was and theoretically what its solution was, there was still much about addressing that was unclear. More theoretical and practical work was necessary. However, it did put us in the uncomfortable position of an *Internet* address naming a *subnetwork* point of attachment.

NCP is phased out. Finally, after eight years of development, TCP was deployed in 1982. The Internet did its first (and nearly last) flag day switch from NCP to TCP. In the same time frame (late 1970s, early 1980s), the (in)famous BBN 1822 Host–IMP hardware interface was being phased out in favor of a standard interface. For hosts connecting to a packet switch, the choice was, in most cases, IP over X.25; for others, it was the new-fangled Ethernet. NCP had served well for more than a decade, much longer than anyone expected.

Saltzer on addressing. In 1982, Jerry Saltzer at MIT published one of the most cited papers on naming and addressing in computer networks. Saltzer (1982) outlined how a network must have application names, which map to node addresses, which map to point of attachment addresses, which map to routes. These are all of the necessary elements of a complete addressing architecture.[11] The only missing piece then is figuring out what location-dependent means in a graph. While everyone cites this paper and agrees that it is the right answer, there have been no proposals to implement it. But in all fairness, Saltzer doesn't provide much help with how his abstractions might be applied to the existing Internet or what *location dependent* means in a graph.

Host table gets unwieldy—DNS but no application names or directory. From the beginning of the ARPANET, the *Network Information Center* (NIC) had maintained a text file of the current list of hosts and their corresponding IMP addresses. Every few weeks, the latest version of the file was downloaded. Then weeks became every week, became every other day, and by 1980 or so it was becoming hard to manage manually as a simple text file. This was bound to happen with the Internet continuing to grow. So now was a good time to take the first step to resolving some of the addressing problems, by putting a scheme of application names and a directory in place. But there were still only three applications in the Net, and each host had only one of each. There was still no

---

11 There is only one refinement we will need (and will turn out to be crucial, see Chapter 5) that did not exist or was very rare when Saltzer wrote, so it is not surprising that he did not consider it.

real need for all the trouble of a directory. And everyone was quite comfortable with the way it had been done for the past 15 years.[12] So, DNS was created essentially as a hierarchy of distributed databases to resolve synonyms for IP addresses, replacing the old host table. This approach was partly due to the strong attachment to the idea of naming hosts that was begun with the ARPANET (even though a careful analysis of naming in networks shows that naming hosts is not relevant to the addressing necessary for communications). As long as there were well-known sockets and only one occurrence of an application in each host, DNS was all the "directory" that was needed: a means to maintain a user-friendly form of the IP address. Even though there had been discussions of a directory since the early 1970s, an opportunity to show some vision was lost. Already the attitude of introducing no more change than necessary to address the current problem had set in. Was this prudent engineering, shortsightedness, protecting the status quo, or a bit of all three?

Congestion collapse. In 1986, the Internet encountered its most severe crisis. The network was suffering from congestion collapse. The classic congestion curve of increasing throughput followed by a nosedive became a daily occurrence. Long delays caused by congestion led to timeouts, which caused retransmissions that made the problem worse. Although the connectionless model had become the *cause célèbre* early in the 1970s, the ARPANET was fundamentally a connection-oriented network (unless Type 3 messages were explicitly used). Even after the move to IP, many host attachments to packet switches and routers were made with BBN 1822 or X.25, both of which flow controlled the host. As more and more hosts were attached by connectionless LANs with no flow control, and as 1822 and X.25 were phased out, there was less and less flow control in the network. The only flow control that existed was in TCP. But TCP flow control was intended to prevent the sending *application* from overrunning the destination *application,* not with preventing congestion somewhere in the network. Congestion collapse was inevitable. No one had ever experimented with the properties of connectionless networks as they scaled up.[13] Now it had to done on-the-fly.

This was a major crisis. Something had to be done and done quickly. The Internet was basically unusable. But the crisis was much deeper than simply keeping an operational network up and running. Control theory going back to Weiner said that feedback should be located with the resource being controlled.

12 No wonder there were people who thought it was *supposed* to be done this way. Fifteen years ago in computing is nearly ten generations—ancient history!

13 There had been calls for experimental networks, and some small ones had been built, but not large enough to investigate these problems. They were too expensive. No one was willing to fund simulations of large networks. Not to mention that there were detractors who questioned whether such simulations would be meaningful.

But congestion could happen at any switch in the network. To include congestion control would essentially mean going to a connection model, not a connectionless model. First, it was known that connection-oriented designs did not work that well and had bad survivability properties. Second, for the past 15 years, the networking community had been fighting off the phone company giants in debates over connectionless and connections (see Chapter 3). We couldn't admit defeat, and we didn't think we were wrong.[14] Many believed there was a middle ground, a synthesis, but so far no one had been able to find it. All proposals seemed to fall into one extreme or the other. In any case, there certainly wasn't time for new theoretical insights. Something had to be done quickly.

Van Jacobson proposed a congestion-avoidance scheme to be inserted into TCP. It consisted of the now well-known slow-start, doubling the congestion window with every round-trip until congestion is detected (and then exponential backoff). Essentially, congestion avoidance creates congestion and then backs off. This solution maintained the connectionless model and provided a quick fix to the congestion problem, while researchers tried to understand how to do congestion control and maintain the seminal properties of a connectionless network. Furthermore at this point, it was much easier to change the TCP implementations than to redesign all the switches. Perhaps as important, this juncture also signals a qualitative shift in networking from flow control being discrete counting of buffers to continuous control theory mechanisms. However, after the crisis was past, there was such relief that no one went back to try to understand what a full solution might look like. And with an all-too-human trait, rationales appeared to justify why this was the "right" solution. There were without doubt several reasons: the "it works don't change it" attitude;[15] the strong adherence to the end-to-end principle; pressure from the outside to adopt connection-oriented solutions; and so on. But congestion collapse had been put behind us so that today there is a consensus that congestion control *belongs* in TCP. But wasn't it a stopgap? Could the conditions that led to congestion collapse occur again? What would it take? Perhaps, a killer app that generated large amounts of traffic, but didn't use TCP? What if the bulk of traffic on the Net were not using TCP? Like with, say, video?

SNMP. The ARPANET had always had good network management,[16] but it was a function internal to BBN that was running the Net. In the early 1980s, as

---

14 And we weren't.

15 At the time, few of the networking people involved had a strong background in control theory, very few were comfortable with the issues, and so there was greater reticence to start changing something so large that was working.

16 The stories are legend: BBN calling Pacific Bell to tell them their T1 line from Santa Barbara to Menlo Park was having trouble and Pacific Bell not believing that they weren't calling from either Santa Barbara or Menlo Park, but from Boston.

more corporate networks were created, network management had become a topic of concern. By the mid-1980s, experience with the IEEE 802.1 management protocol had shown that the elemental "Turing machine" approach,[17] although simple and straightforward, was inadequate. It was also clear by this time that the key to network management was less the protocol and more the object models of the systems to be managed. The Internet community pursued two approaches: a simple Turing machine-like, polling[18] protocol, SNMP without object-oriented characteristics; and a more sophisticated extensible object-oriented, event-driven protocol, HEMS. It is probably significant that unlike the ARPANET, which came up with innovative solutions to problems, the Internet of the late 1980s took a step away from innovation by adopting SNMP. There was strong emphasis at the time on the apparent simplicity, supposedly leading to smaller code size and shunning concepts that were seen as too esoteric.[19] As it turned out, SNMP implementations are larger than either HEMS or CMIP.[20] Its rudimentary structure and lack of object-oriented support, along with a red herring that we will look at in Chapter 4, "Stalking the Upper-Layer Architecture," has proven to be a major obstacle to the development management in the Internet.

The Web. In the early 1990s, the Web began to take off. The Web had been around for a while, but was basically just another version of Gopher. Until NCSA at the University of Illinois extended it with a browser. One of the major efforts of the supercomputer center was investigating how to present data more effectively. As part of that, one of their programmers hit upon the idea of putting a GUI on the Web that made any object on the page "clickable." The Web took off and put new requirements on the Net.

The Web becomes the first major new application on the network in 20 years, and as one would expect it created a number of new problems. First of all, this is the first application that did not come from the operating system metaphor. For the Web, the protocol and the application are not one and the same. There may be more than one application using the Web protocol and more than one instance of the same application at the same time on the same host. With no application naming structure in place, the Web had to develop its

---

17 Everything is done with Set and Get on attributes.

18 The use of polling in SNMP has always been perplexing. In the ARPANET, polling was seen as a brute-force approach that didn't scale and represented mainframe think. It was an anathema. It would never have been considered, and anyone proposing polling in those days would have been laughed out of the room.

19 Push-down automata, object-oriented, and so on. There was a strong anti-intellectual attitude then (and still is to some extent) that real programmers "don't need no book learning." They innately know how to design and write code.

20 The OSI management protocol, which was event-driven and was object-oriented.

own naming scheme, the now ubiquitous URL. However, once again, this did not lead to consideration of the deeper structure of what this was saying about the requirements for naming. Instead, there was considerable interest in extending the existing scheme with the work on Universal Resource Names.

With network management, we again see the focus on the short term and how to fix a specific problem, but little focus on what this is telling us about the general problem.

IPng. In the early 1990s, the Internet was growing by leaps and bounds. At the rate things were going, there was going to be a shortage of IP addresses, although of greater concern was the growing router table size. The IAB embarked on a program to determine a course of action. After a thorough process considering the pros and cons of a new protocol effort or adopting an existing protocol, they recommended a two-pronged approach of conservation and replacing IP with the OSI version called CLNP. Conservation consisted of IANA tightening the number of addresses handed out, the use of private addresses, instituting CIDR to facilitate aggregation of routes, and forcing most requests for addresses through the major providers to reinforce the move to CIDR.

The years of isolation between the Internet and OSI had done their job. The proposal to adopt an OSI protocol precipitated a huge uproar, which led to the IAB reversing itself, and the IPng process was begun to select a new protocol. The requirements for an acceptable IPng were drafted, which among other things required that the address continue to name the interface, not the node (even though it had been known since 1972 that a network address, let alone an *internetwork* address, should not name a *subnet* point of attachment). Basically, the only problem the resulting IPv6 solves is lengthening the address. In particular, it did nothing to arrest the growth of router tables and nothing to solve 20-year-old deficiencies in the addressing architecture.[21] And what it does do, it makes it worse. Furthermore, the transition plan to IPv6 called for network address translation (NAT). As it turned out, owners of networks liked NATs for other reasons. Once one had a NAT and private address space, there was little reason to adopt IPv6. Had the IPv6 group chosen to fix the addressing problem and come to grips with the fact that IPv4 was not an *Internet* protocol, they could have fixed the problem and avoided the use of NATs.

Why did the IETF not fix a problem that had been known for 20 years? Several reasons:

---

21 It pains me to watch the IETF resorting to spin for IPv6 to cover up its inadequacies. It used to know how to call a lemon, a lemon.

1. CLNP did fix it, and there was a strong attitude that if OSI did it, the Internet wouldn't.[22]

2. Very few people in the IETF (maybe a dozen or so out of about 1,000) understood the problem.[23] What should be named in a network architecture was not taught in universities. In fact, even today one will be hard pressed to find a networking textbook that covers this topic.

3. There was a belief that any multihoming would be to different providers,[24] which would either have no peering point or they would be so distant that it would unnecessarily complicate the routing, if not be impossible. There were also excuses about addresses being provider-based, but this is an artifact of naming the interface and misses the point of Saltzer's paper that point of attachment addresses are "physical addresses" but node addresses are "logical addresses."

Internet traffic is self-similar. In 1994, a paper was published by a group at Bellcore showing that measurements of Internet traffic on various Ethernets exhibited self-similarity. Some found this a revelation—that this was the first inkling that traffic was not Poisson—when, in fact, this fact had been known since the mid-1970s.[25] This observation created huge interest, and a lot of researchers jumped on the bandwagon. There was more than a little infatuation with the idea that the Internet was described by the hot new idea of fractals, chaos, the butterfly effect, etc. Although not reported in that paper, there were immediately deep suspicions that it wasn't Internet traffic *per se* or Ethernet traffic that was self-similar, but that the self-similarity was an artifact of TCP congestion control. This was later verified. TCP traffic is more strongly self-similar than UDP traffic, and Web traffic is somewhat less self-similar than TCP traffic. The lower self-similarity of Web traffic is most likely a consequence of the "elephants and mice" phenomenon. But interestingly enough, the result that TCP congestion control was causing chaotic behavior did not precipitate a review of how congestion control was done. The general view of the community seemed to be that this was simply a fact of life. This is in part due to the ideas being currently in vogue and the argument being made by some that large systems all exhibit self-similar behavior, so there is nothing to do.

---

22 Of course, there were very logical rationales for not changing it that sounded good if one didn't look too closely, but it doesn't change the underlying reaction.

23 This argument plays out on an IETF list every few months. Some still arguing that they should be able to take their *address* wherever they go. Nothing has been learned in the past 15 years.

24 Which is only sometimes the case in the real world.

25 The problem was that bursty traffic required a new approach to modeling. No one had come up with one (and still haven't).

That brings us to roughly the early 1990s, to the time frame when I started this exercise, just as the IPng was heating up.[26] The seven unanswered questions we started with were still unanswered and in the back of my mind (as they always had been). It was not my intention to try to solve them. It is a daunting list. But as each pattern emerged, it was measured against whether they contributed to solving them. I was looking for a clear understanding of where we were. However, three issues had to be looked at. Two of the issues experience had shown could wreck an architecture if not confronted and solved. We have already touched on them: finding a meaningful synthesis of connection and connectionless, and working out naming and addressing (and in particular what location dependent means). The religious war over connections and connectionless had been at the root of too many disasters. A true synthesis was desperately needed. And, of course, just looking at the seven unanswered questions, you can see that a number of issues all revolve around a clear understanding of naming and addressing. The third arose from my experience with hundreds of protocol designs over more than 30 years, seeing the same things over and over. I wanted to separate mechanism and policy as we had in operating systems—just to see what would happen.[27]

Keep in mind that this wasn't my job, my thesis, or my research grant. This was just something I did in my spare time. The initial foray was very productive. Separating mechanism and policy revealed patterns I hadn't seen before and renewed interest in patterns I had seen 15 years earlier (but at the time did not seem to go anywhere). By 1994, the outlines of the model presented here were clear. There weren't seven layers or five layers, but a single layer of two protocols along with optional information that recursed. The limitations of technology and our focus on differences had hidden the patterns from us. This collapse in complexity immediately solves a long list of problems.

Although there were some key problems to solve, it was never a case of finding just anything that solved them. They were threads to pull on in untangling the knot confronting us. Merely finding something that would work was not enough. The solution had to fit into a larger "theory." If it didn't, either the solution or the theory needed to change. I quickly learned (and was often

26 I remember being at IETF meetings where IPng was under heavy discussion and having just had these fundamental insights, but having not as yet completely worked it through.

27 Along the way, I picked up a fourth coming out of my frustration with the fact that although we revel in the idea that network traffic is bursty, we then do everything we can to get rid of the burstiness and what I saw as a missing piece: We have a body of literature on ack and flow-control strategies but not on multiplexing (except as a physical layer phenomenon). Although I have made significant progress on this topic, it isn't covered in this book because it just isn't an "architecture" problem.

reminded) that it was more important to go where the problem told me, rather than to do what I thought was best. (Some readers will think I have completely lost it; others who have had the experience will know precisely what I mean.)

In the mid-1990s, however, no one believed there was any reason to look at "new architectures." And in any case, I wasn't done yet, so I just kept mulling over the patterns. Sometimes I put the work down for a year or more. Then some new insight would reveal itself and I would dive into it for a while. Sometimes I would see the pattern the problem was showing me, but it was so at odds with conventional directions that I wouldn't fully embrace it. But there would be continuing hints that doing what the problem was saying would be better. Finally, my resistance would collapse and further simplifications and insights resulted.[28]

What emerged was a much simpler model of networking. A complexity collapse. We knew the outlines of what addressing had to be fairly early. Jerry Saltzer gave us the basics in 1982. But a slight extension to Saltzer to accommodate a case that didn't yet exist yielded a result that dovetailed neatly with the emerging structure of protocols (i.e., it repeated). The results were reinforcing each other. This was getting interesting. This would happen more and more. Someone would remark about something that was hard to do, and it turned out to be straightforward in this model. When capabilities that were not specifically designed in turn out to be supported, it is usually an indication you are on the right track.

The problem of location dependence was much harder. It had always been clear that addresses had to be location dependent, but route independent. It took years of reading and thinking. But slowly I came to the conclusion that for addresses to be location dependent in a meaningful way, they had to be defined in terms of an abstraction of the graph of the network. Looking for mathematical tools for abstracting graphs led to topology and the conclusion that an address space has a topological structure. Throughout the 1990s, I talked to people about this, and by the late 1990s, I had a way to go and an example.

Later, an off-handed teaching question about a detail of protocol design led to revisiting fundamentals that we all knew, and this turned out to shed new light on the structure and further simplification.

So, does this book solve all of our problems? Hardly. But it does lay out the fundamental structure on which a general theory of networking can be built. It does give us a place to stand outside the current box we find ourselves in and see what we have been missing. It turns out that it wasn't so much that what was missing was huge, but it was *key* to a simple solution. I have tried to strike a balance between readability and formality. But one of my goals here has been to try

28 This was the case with the structure of error- and flow-control protocols.

to find the minimal set of concepts necessary to represent the problem. This model is very close to being that. This is a fundamental model. Much of what we have done over the past 30 years is still quite applicable. But this model gives us a much better basis for reasoning about networks independent of any particular network or technology. My hope is that this will spark insights and ideas by others, and I look forward to them.

As noted earlier, several concepts that are key to understanding this model are not generally known. We will rely heavily on what Seymour Papert[29] calls the only concepts that make computer science worth learning: problem decomposition, abstraction, and recursion. Abstraction has fallen into disuse for the past couple of decades, but we will put it to good use here. Furthermore, the architecture we are led to requires a considerable cognitive shift. Therefore, this book is organized to take the reader from what we know to a new way of looking at things. To bridge the gap, so to speak. Even so, this will not be easy for the reader; there is some hard thinking ahead.

We first start with a return to fundamentals, to remind us of the minimum assumptions required for communication and for the tools for working with abstractions. In Chapters 2 and 3, we look at the familiar world of protocols and separating mechanism and policy. Here, new patterns emerge that indicate there are probably only three kinds of protocols, and then later we find that one of them is more a "common header" than a protocol. We are also able to make considerable progress in resolving the conflict between connections and connectionless.[30]

In Chapter 4, we review our experience with "upper layers" and learn some things that we did right and some things to avoid. As strange as it might sound, we find some key concepts here that will be useful in constructing our fundamental model, while at the same time concluding that there is no "upper-layer architecture." Then in Chapter 5, we take a hard look at that ever-difficult and subtle topic, "Naming and Addressing." We give special emphasis to Saltzer's 1982 paper expanding on it slightly, noting how the current infatuation with the "loc/id split" problem is a dead end. By the time we reach Chapter 6, we have a pretty reasonable picture of the problem and the elements we will need and can consider the problem of assembling them into layers. Here we embark on a simple exercise that any of us could have done at any time over the past 30 years only to find it yields the structure we have been looking for. (What a revolting department!) This chapter is key to everything.

---

29 I wish I could cite a reference for this. Seymour assures me he said it, but he can't remember where, and I can't find it!

30 We don't address the problem of connectionless scaling because this isn't strictly an architectural problem, although the structure presented here facilitates a solution.

In Chapter 7, “The Network IPC Model,” we do the unpleasant task of assembling all the pieces we have uncovered in the previous six chapters into the elements of the new model and consider its operation. This entails emulating Johnson’s harmless drudge as we define all the concepts required. Messy work, but it has to be done. We consider how new nodes join a network and how communication is initiated. Chapter 8, “Making Address Topological,” returns us to naming and addressing to consider the problem of what *location dependent* means and how to make useful sense of the concept. In Chapter 9, we look at how “Multihoming, Multicast, and Mobility,”are represented in this model and some new results that are a consequence of this model. In Chapter 10, “Backing Out of a Blind Alley,” we take stock, consider the process that led to seven fundamental issues going unsolved for more than a quarter century, and look to the future.

Acknowledgments

This Preface can't end without expressing my immense appreciation to the long list of people who have contributed their time and effort to this book and to my thinking, all of the people whose ear I have bent over the years working through these ideas. The list in its entirety is far too long, but let me hit the high points: Sue Hares, Lyman Chapin, Margaret Loper, Charles Wade, Glenn Kowack, Geneva Belford, Fred Goldstein, George Schimmel, William Zimmer, Sue Rudd, Chris Williams, Fernando Gont, Sharon Day, and a special thanks to Lynn DeNoia for asking the important questions. The reviewers and friends who had to endure so much: Jonathan Smith, Michael O'Dell, Pekka Nikkander, Ibrahim Matta, Tony Jeffree, and Joel Halpern. Catherine Nolan, Mark Taub, Keith Cline, and Chuck Toporek at Prentice Hall for tackling a different kind of book.

And of course, my wife, Meg, whose love and support sustained me throughout this project (although, I think she was never quite sure it would ever end).

—John Day, Lake Massapoag, 2007

About the Author

John Day has been involved in research and development of computer networks since 1970, when they were 12th node on the "Net." Mr. Day has developed and designed protocols for everything from the data link layer to the application layer.

Also making fundamental contributions to research on distributed databases, he developed one of two fundamental algorithms in the updating of multiple copies. He also did work on the early development of supercomputers and was a member of a development team on three operating systems. Mr. Day was an early advocate of the use of *Formal Description Techniques* (FDTs) for protocols and shepherded the development of the three international standard FDTs: Estelle, LOTOS, and extending SDL. Mr. Day managed the development of the OSI reference model, naming and addressing, and a major contributor to the upper-layer architecture; he also chaired the US ANSI committee for OSI Architecture and was a member of the Internet Research Task Force's Name Space Research Group. He has been a major contributor to the development of network management architecture, working in the area since 1984 defining the fundamental architecture currently prevalent and designing high-performance implementations; and in the mid-1980s, he was involved in fielding a network management system, 10 years ahead of comparable systems. Recently, Mr. Day has turned his attention to the fundamentals of network architectures and their implications (as discussed in this book).

Mr. Day is also a recognized scholar in the history of cartography, on Neolithic Korea, and on Jesuits in 17th-century China. Most recently, Mr. Day has also contributed to exhibits at the Smithsonian and a forthcoming chapter in *Matteo Ricci Cartographia*.

This page intentionally left blank

Chapter 5

Naming and Addressing

Did I ever tell you that Mrs. McCave
Had twenty-three sons and she named them all Dave?
Well, she did. And that wasn't a smart thing to do.
You see, when she wants one and calls out, "Yoo-hoo!
Come into the house, Dave!" she doesn't get one.
All twenty-three Daves of hers come on the run!

This makes things quite difficult at the McCaves'
As you can imagine, with so many Daves.
And often she wishes that, when they were born,
She had named....

[There follows a wonderful list of Dr. Seuss names she wishes she'd named them, and then concludes with this excellent advice.]

But she didn't do it and now it is too late.

—Dr. Seuss, Too Many Daves

Introduction

Many years ago when I started to work on the addressing problem, I remembered the opening lines to a Dr. Seuss story that I had read to my children far too many times. I thought it would make a good introductory quote for naming and addressing. So I dug into my kids' books to find it. Of course, I couldn't do that without reading the whole story through to the end for the great list of names she wished she had called them. But I had forgotten how it ended. I hit that last line and wondered whether Dr. Seuss had been sitting in all those addressing discussions and I just never noticed him! There was never more appropriate advice on naming and addressing than that last line.

The problem of addressing has confounded networking from the beginning. No other problem is so crucial to the success of a network; is so important to get right early and at the same time is so subtle, so philosophical, and so esoteric. No matter how you approach it. Once defined, it is difficult to change, and you may find yourself in the same situation as Mrs. McCave. If it is wrong and must be changed, the longer it takes to realize it, the more painful (and costly) it will be to change. If it is really wrong, the use of the network becomes cumbersome and arcane and eventually useless. Trying to fix it piecemeal as problems arise, only prolongs the agony, increases the cost, and increases the pain when the inevitable finally comes. But if it is right, many things become easier, and you scarcely realize it is there.

Why Do We Need Naming and Addressing?

The short answer is: to know where to send data. However, the more considered answer is a little longer (but amounts to the same thing). One of the major efficiencies of networks is that every source does not have to be directly connected to every destination. If they were, only the simplest networks would be feasible, and addresses would always be a local matter. But by allowing nodes in the network to act as intermediates to relay messages from sources to destinations, we must at least distinguish them with names, and as the network grows we can greatly decrease the cost of the network at the "mere" expense of adding addresses to the protocols and routing to the network.[1] We need to distinguish messages from each other. For simple networks, the mechanisms are deceptively simple, and simply enumerating the nodes is sufficient. But as the size and complexity of the network grows, naming and addressing begins to show itself as a subtle maze with all sorts of traps, quagmires, and dead ends. The protocol designer begins to wonder whether he has unwittingly signed a pact with the devil. But it is too late to turn back. And one is left wondering how engineering suddenly became so philosophical.

There are basically two separate problems that we must consider: 1) What objects need to be named to effect communications, and 2) the nature of the names and addresses used to label these objects. But before diving into the theory of addressing, let's consider how we got here so that we have a better understanding of why the theory is being asked to answer certain questions.

1 The "multidrop" technologies accomplish a similar reduction in cost for "star" topologies and also require addressing mechanisms.

How the Problem Arose

Naming and addressing had never been a major concern in data communications. The networks were sufficiently simple and of sufficiently limited scope that it wasn't a problem. Most early networks were point-to-point or multidrop lines, for which addressing can be done by simple enumeration. Even for large SNA networks, it was not really an issue. Because SNA is hierarchical with only a single path from the leaves (terminals) to the root (mainframe), enumerating the leaves of the hierarchy (tree) again suffices.[2] In fact, addressing in a decentralized network with multiple paths, like the early ARPANET or even the early Internet, can be accommodated by enumeration and was. But everyone knew the addressing problem was lurking out there and eventually it would have to be dealt with.

The ARPANET was a research project that wasn't expected by many to succeed. No one expected the ARPANET to ever be large enough for addressing to be a major problem, so why worry about an esoteric problem for which at the time we had no answers. As it was, there were an overwhelming number of major technical problems to solve which were a lot more crucial. Just being able to route packets, let alone do useful work with it, would be a major achievement. After all, it was research. It was more important to be focused on the few specific problems that were central to making the project work. Addressing was distinctly a lesser issue. Of course, to everyone's surprise the ARPANET was almost immediately useful.

Because the initial design called for no more than a few tens of switches connecting a few hosts each, addressing could be kept simple. Consequently, there were only 8 bits of address on the *Interface Message Processors* (IMP). Host addresses were the IMP number (6 bits) and the IMP port numbers (2 bits). Each IMP could have a maximum of 4 hosts attached (and four 56K trunks). IMP numbers were assigned sequentially as they were deployed.

Although a maximum of 64 IMPs might seem a severe limitation, it seemed like more than enough for a research network. There was not much reason for concern about addressing. Once the success of the ARPANET was accepted, the address size of NCP was expanded in the late 1970s to 16 bits to accommodate the growth of the network. (*Network Control Program* implemented the Host-to-Host Protocol, the early ARPANET equivalent of TCP/IP.)

---

2 SNA could even enumerate the routes, because the hierarchy kept the number from growing too fast. But if you don't understand why, it can lead to problems. There was a network company that many years ago tried to use the SNA approach for nonhierarchical networks (after all if it was used by IBM, it must be right!) and couldn't figure out why the number of routes exploded on them.

It was clear that the one aspect of naming and addressing that would be needed was some sort of directory. ARPA was under a lot of pressure to demonstrate that the network could do useful work; there certainly was not time to figure out what a directory was and design, and implement such a thing. And for the time being, a directory really wasn't necessary. There were only three applications (Telnet, FTP, and RJE), and only one each per host. Just kludge something for the short term. A simple expedient was taken of simply declaring that everyone use the same socket for each application: Telnet on socket 1, FTP on 3, and RJE on 5.[3] Every host would have the same application on the same address. This would do until there was an opportunity to design and build a cleaner, more general solution. Hence, well-known sockets were born. (Strangely enough, while many of us saw this as a kludge, discussions among the people involved revealed that others never saw it that way. An unscientific survey indicates that it may depend on those who had early imprinting with operating systems and those that didn't.)

If there was any interest in naming and addressing during that period, it was more concerned with locating resources in a distributed network. How does a user find an application in the network? By the mid-1970s, several efforts were underway to build sophisticated resource sharing systems on top of the ARPANET (the original justification) or on smaller networks attached to the ARPANET. David Farber was experimenting with a system at UC Irvine that allowed applications to migrate from host to host (Farber and Larson, 1972); and another ARPA project, the National Software Works, was trying to build an elaborate distributed collaboration system on top of the ARPANET (Millstein, 1977). These projects raised questions about what should be named at the application layer and how it related to network addresses, but outstripped the capability of systems of the day.

The problem of naming and addressing had been a factor in the development of operating systems. The complexity of process structure in some operating systems provided a good basis for considering the problem (Saltzer, 1977). Operating system theory at the time drew a distinction between location-independent names and the logical and physical levels of addresses. This distinction was carried into networking and generalized as two levels of names: 1) location-independent names for applications and 2) location-dependent addresses for hosts.

[3] When "new Telnet" was defined, socket 23 was assigned for debugging and experimenting with the new design until the old Telnet could be taken out of service and new Telnet moved to socket 1. Telnet is still on socket 23.

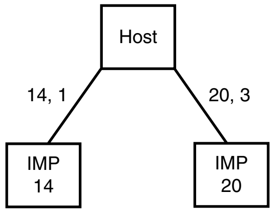

Figure 5-1 Because ARPANET host addresses were the port numbers of the IMPs (routers), a host with redundant network connections appears to the network as two separate hosts. Routing can't tell the two lines go to the same place.

The general concept was that the network should seem like an extension of the user's interface. The user should not have to know where a facility was to use it. Also, because some applications might migrate from host to host, their names should not change just because they moved. Thus, applications must have names that are location independent or as commonly called today, portable. The binding of application names to processes would change infrequently. These applications would map to location-dependent addresses, a mapping that might change from time to time. Network addresses would map to routes that could change fairly frequently with changing conditions of the network. That was the general understanding.

Using switch port numbers for addresses was not uncommon. After all, this is basically what the telephone system did (as did nearly all communication equipment at that time). However, although this might have been acceptable for a telephone system, it causes problems in a computer network. It didn't take long to realize that perhaps more investigation might be necessary. Very quickly, the ARPANET became a utility to be relied on as much or more than an object of research. This not only impairs the kind of research that can be done, it also prevents changes from being made. (On the other hand, there is a distinct advantage to having a network with real users as an object of study.) But it also led to requirements that hadn't really been considered so early in the development. When Tinker Air Force Base in Oklahoma joined the Net, they very reasonably wanted two connections to different IMPs for reliability. (A major claim [although not why it was built] for the ARPANET in those days of the Cold War was reliability and survivability.) But it doesn't work quite so easily. For the ARPANET, two lines running to the same host from two different IMPs, have two different addresses and appear as two *different* hosts. (See Figure 5-1.) The routing algorithm in the network has no way of knowing they go to the same place. Clearly, the addressing model needed to be reconsidered. (Because not many hosts had this requirement, it was never fixed, and various workarounds were found for specific situations.) Mostly, the old guard argued that it didn't really happen often enough to be worth solving. But we were operating system guys; we had seen this problem before. We needed a logical address space over the physical address space! The answer was obvious; although it would be another ten years before anyone wrote it down and published it. But military bases were rare on the Net, so it was not seen as a high-priority problem. Also, we all knew that this was a hard subtle problem, and we needed to understand it better before we tried to solve it. Getting it wrong could be very bad.

Background on Naming and Addressing

The problems of naming and addressing remained an interesting side issue for the Net, not a problem crucial to survival for many years. There weren't too many places to learn about naming and addressing. In the early days of computer science, there was considerable emphasis on mathematical logic, the predicate calculus and related subjects. Some aspects of naming are taken up there in some detail. As previously mentioned, there had been some work done in the context of operating systems. The postal system and the telephone system solved this problem on a global scale; and although both are large systems, they are also simpler in significant ways. Most of the network is hierarchical, and the part that isn't was strongly geographical with a single provider. They didn't have to consider multicast, migrating applications, multihoming, or until recently, mobility.

Foundations of Mathematics and Naming

As we have said, the problems of naming and addressing have a tendency to get philosophical. What to name, the relation among various names and the objects they refer to, and the structure that such names should have and what constructs they can support are all issues to be considered. It doesn't take long before it can begin to sound like counting angels on the head of a pin. However, experience has shown that subtle distinctions can often make the difference between a simple but rich and efficient naming scheme and a scheme that becomes complex and cumbersome and may not even work. So, perhaps we should consider those aspects before we go too much further. Because we are concerned with naming and addressing in computers and networks of computers, we will not discuss the full scope of naming issues that have been taken up by philosophy. We will only provide a taste of these issues and limit ourselves to those aspects of the mathematics that apply most directly to our problem.

Modern considerations of naming derive from the work on the foundations of mathematics and symbolic logic. This work got significant attention in the late 19th century with the interest in the foundations of mathematics and the work of Gottlieb Frege, with major contributions coming from the work of Bertrand Russell and Alfred North Whitehead, Ludwig Wittgenstein, Rudolf Carnap, and others who became known as the Vienna Circle. Primarily, they were concerned with two problems: 1) creating a strictly axiomatic basis for all of mathematics and 2) the means to create purely logical language to describe the world. Both projects failed. The first because Kurt Gödel proved the "incompleteness theorem," or in essence "no matter where you start, there is some place you can't get to from here." And the second by Wittgenstein, who in his *Tractatus Logico-Philosophicus* made it clear that most of what philosophy had been talking about for the past 2,000 years could not be stated with sufficient precision to prove any conclusions. And all those things that could were tautologies, which say nothing. However, in the process of getting to these conclusions, considerable insights were made into the nature of language, the foundations of mathematics, symbolic logic, and so on.

Much of this work related to constructing a precise logical language. Consequently, one of the major considerations was precisely determining the relation of names to their meanings and how these meanings came about. Frege, in his essay "On Sense and Meaning" (1892) defined a name as follows:

> A proper name (word, sign, sign combination, expression) expresses its sense, means or designates its meaning. By employing a sign we express its sense and designate its meaning.

Here and in the *Basic Laws of Arithmetic* (1884), Frege goes on to develop the concept of a name to correspond closely to what one intuitively thinks of as a noun clause. As alluded in the definition, a name can be an expression. Frege also introduced variables into these expressions and the concept of bound and unbound variables, although the use of these terms did not come until later. Frege distinguishes simple and complex complete names. Simple names are what we would term constants; complex names are expressions. A complete name has all of its variables bound to constants. For Frege, an incomplete name (i.e., one with unbound terms) is a function. Frege uses these concepts and a unique notation in an attempt to derive the fundamental rules of arithmetic. However, he only came close. As his book went to press, Frege received what is now a famous letter from Russell advising him of a problem Russell had encountered in his own attempt with Whitehead to put mathematics on a completely logical footing (the set of all sets that do not contain themselves, leading to the Russell paradox). Frege had missed the paradox that stumped Russell for quite awhile and whose solution is still debated by mathematicians. Although the damage was not irreparable, Frege never revised his book to fix the problem.

Twenty some years later, the young Ludwig Wittgenstein took issue with Frege and to some extent Russell in his work that revolutionized mathematics and philosophy, the *Tractatus Logico-Philosophicus* (1922). We have already touched on the *Tractatus* in Chapter 1, "Foundations for Network Architecture," but here let's look more closely at what it says about names. Right off the bat, Wittgenstein takes issue with Frege:

> **3.142** Only facts can express a sense, a set of names cannot.
>
> **3.143** Although a propositional sign is a fact, this is obscured by the usual form of expression in writing or print. For in a printed proposition, for example, no essential difference is apparent between a propositional sign and a word. (This is what made it possible for Frege to call a proposition a composite name.)
>
> **3.144** Situations can be described but not given names.

An early 20th-century flame, W goes on to give a much restricted definition of a name, which corresponds to what we will call here a *primitive name:*

> **3.202** The simple signs employed in propositions are called names.
>
> **3.203** A name means an object. The object is its meaning. ('A' is the same sign as A.)
>
> **3.22** In a proposition a name is the representative of an object.
>
> **3.26** A name cannot be dissected any further by means of a definition: it is a primitive sign.
>
> **3.261** Every sign that has a definition signifies via the signs that serve to define it; and the definitions point the way.
>
> Two signs cannot signify in the same manner if one is primitive and the other is defined by means of primitive signs. Names cannot be anatomized by means of definitions. (This cannot be done to any sign that has a meaning independently and on its own.)

W is nailing things down pretty tight, defining a name as essentially a label for an object. This is a denotative approach to naming. He goes on to point out that names by themselves say very little:

> **3.3** Only propositions have sense; only in the nexus of a proposition does a name have meaning.

> **3.314** An expression has meaning only in a proposition. All variables can be construed as propositional variables. (Even variable names.)
>
> **3.3411** So one could say that the real name of an object was what all symbols that signified it had in common. Thus, one by one, all kinds of composition would prove to be unessential to a name.
>
> **4.0311** One name stands for one thing, another for another thing, and they are combined with one another. In this way the whole group—like a *tableau vivant*—presents a state of affairs.
>
> **4.23** It is only in the nexus of an elementary proposition that a name occurs in a proposition.

So, W comes full circle or would seem to. The meaning of a name can only be determined when it occurs in a proposition (i.e., in context). Further, all expressions must reduce to a primitive name, and these expressions do not affect the name. Where is W headed with all of this? Right here:

> **5.526** We can describe the world completely by means of fully generalized propositions, i.e., without first correlating any name with a particular object.
>
> **6.124** The propositions of logic describe the scaffolding of the world, or rather they represent it. They have no 'subject-matter'. They presupposed that names have meaning and elementary propositions sense; and that is their connection with the world. It is clear that something about the world must be indicated by the fact that certain combinations of symbols-whose essence involves the possession of a determinate character-are tautologies. This contains the decisive point. We have said that some things are arbitrary in the symbols that we use and that some things are not. In logic it is only the latter that express: but that means that logic is not a field in which we express what we wish with the help of signs, but rather one in which the nature of the natural and inevitable signs speaks for itself. If we know the logical syntax of any sign-language, then we have already been given all the propositions of logic.

The hope had always been that logic could resolve important questions in philosophy. What W has done here and will wrap up between here and the famous statement 7 says that names are arbitrary labels and all statements in logic are tautologies. They say nothing about the real world.

What Happened Next? A More Organic View

For those who are curious, W did not rest with the *Tractatus.* He was still troubled by its implications. Twenty years later he published his thoughts again, and this time changed his view considerably, taking a more connotative model of language that is closer to how organisms seem to actually acquire language. Oddly enough, his point of departure was St. Augustine:

1. "When they (my elders) named some object, and accordingly moved towards something, I saw this and I grasped that the thing was called by the sound they uttered when they meant to point it out. Their intention was shown by their bodily movements, as it were the natural language of all peoples: the expression of the face, the play of the eyes, the movement of other parts of the body, and the tone of voice which expresses our state of mind in seeking, having, rejecting, or avoiding something. Thus, as I heard words repeatedly used in their proper places in various sentences, I gradually learnt to understand what objects they signified; and after I trained my mouth to form these signs, I used them to express my own desires." (Augustine, *Confessions,* I. 8)

These words, it seems to me, give us a particular picture of the essence of human language. It is this: The individual words in language name objects-sentences are combinations of such names. In this picture of language, we find the roots of the following idea: Every word has a meaning. This meaning is correlated with the word. It is the object for which the word stands.

38. Naming appears as a queer connection of a word with an object. And you really get such a queer connection when the philosopher tries to bring out the relation between name and thing by starting at an object in front of him and repeating a name or even the word "this" innumerable times. For philosophical problems arise when language goes on holiday. And here we may indeed fancy naming to be some remarkable act of mind, as it were a baptism of an object. And we can also say the word "this" to the object, as it were address the object as "this"-a queer use of this word, which doubtless only occurs in doing philosophy.

43. For a large class of cases-though not for all-in which we employ the word "meaning" it can be defined thus: the meaning of a word is its use in the language. And the meaning of a name is sometimes explained by pointing to its bearer.

275. Look at the blue of the sky and say to yourself "How blue the sky is!"—When you do it spontaneously-without philosophical intentions—the idea never crosses your mind that this impression of color belongs only to you. And you have no hesitation in exclaiming that to someone else. And if you point at anything as you say the words you point at the sky. I am saying: you have not the feeling of pointing-into-yourself, which often accompanies "naming the sensation" when one is thinking about "private-language." Nor do you think that really you ought not to point to the color with your hand, but with your attention.

293. If I say of myself that it is only from my own case that I know what the word "pain" means—must I not say the same of other people too? And how can I generalize the one case so irresponsibly? …

Not only has his thinking changed to such an extent that he now considered that names are conventions among people, not arbitrary labels that can be applied willy-nilly, but he is also considering that the senses that one applies a name to may be different for different individuals (something borne out by cognitive psychology and neurophysiology). The world is far less deterministic that even the *Tractatus* allowed.

Although there had been suspicions to the contrary before this point, mathematics had always been considered a science. There was a belief that it was a universal language with which the world could be completely and precisely described, which would in turn lead to answering many long-standing questions, including some outside the traditional realm of science and mathematics. After all, much of its use was in the service of science, and science made many statements and solved many problems about the real world with mathematics. W has now slammed the door on this view. Logic and, by the constructions of Frege and Russell, mathematics say nothing about the real world and can't. Mathematics is not a science. Mathematicians were operating in the world of Platonic ideals, believing that these truths that they derived were independent of human thought. Although refined by other logicians and mathematicians in the intervening 80 years, the structure and limitations erected by W have remained, circumscribing how far mathematics can go in answering questions that affect people.

But although this was a failure on one level, it was precisely what was required 30 years later when it became possible to build logic machines and get the fledging field of computer science off the ground. The concepts of primitive name, simple and complex, complete and incomplete names were precisely the foundations necessary for constructing the logical languages required for computers, where now these languages could be used in propositions that said something real about a virtual world. It also provides the basis for a theory of naming for networks and distributed system, but provides little help with any fundamentals for addressing. We need a mathematical characterization of "locating" objects.

Naming and Addressing in Telephony

Addressing in the telephone system developed from the bottom up. Initially, telephone systems were isolated islands. Telephone numbers corresponded to numbers on the switchboard, which corresponded to the wires that ran to the phones.[4] Enumeration worked again. The scope of the address space was limited to the island or central office called an exchange; that is, telephones in different exchanges might have the same number. When a phone system outgrew what could be handled by a single central office, trunks were used to link central offices. Each exchange was given a unique identifier, and this number was tacked on the beginning of the number for the telephone: the beginning of hierarchical addressing. Connections between islands required an operator.[5] With the advent of automatic dialing and long distance, it was necessary to add

4 My first phone number was 61.

5 Remember those old movies, *Operator, get me New York, Pennsylvania 6-5000.*

another level to the hierarchy, and area codes were created. But the fundamental semantics of the phone number never changed: It was the number of the wire that ran to the phone. There was really no attempt at structuring the assignment of numbers within an exchange, there might be some similarity in the exchanges used for a single city, but overall the structure of the address space was roughly geographical. This had more to do with conserving the amount of relay equipment than attempting to logically structure the phone numbers.

Over time, as telephone engineers found ways to hack the system to provide specialized services, the semantics of the telephone number got confused. There are strong indications that the phone companies didn't quite understand what they were getting in to. Although normal phone numbers were physical layer addresses, the label of a wire, the definition began to get confused: 800 numbers are application addresses being location independent, whereas 411 and 911 are simply well-known names for specific applications. (Most in phone company circles did not realized this, of course; they were still just phone numbers.) Initially, cellular phone numbers were network addresses, a good unique identifier as the phone was handed off from cell tower to cell tower. But as soon as roaming was provided, they became application addresses (because they were now location independent). Customers had become familiar that when they moved within a city their phone number did not need to change. Although exchanges had begun as exclusively geographical, this began to break down over time with improvements in switches and customer demand. Roaming just served to convince customers that they could move anywhere in the country and not change phone numbers. Because 800 numbers and initially cell phones were such a small population, the mapping from the application address to a network or physical layer address could be a special case. As Signaling System 7 was deployed in the 1980s, it enabled these changes during the 1990s, and the telephone system moved to rationalize its addressing architecture.

Naming in Operating Systems

Much more theoretical work has been done on naming than on addressing. As luck would have it, we are much more interested in addressing than naming. Almost everything in computer science is addressing of one form or another, not naming. There has been very little theoretical work done exploring the properties of addresses, no systematic exploration of addressing. Much of this was because computing systems were so resource constrained. Most of the work has been very pragmatic in the context of solving a specific problem. So, we have some idea of what works or under what conditions it works or what doesn't, but we have very little idea if this is the best we can do.

One of the few theoretical treatments of this subject tempered by implementation of a production system (i.e., it satisfies our philosophical triangulation) is the work of J. H. Saltzer on *Name Binding in Computer Systems* (1977).[6] This is what university-level computer science should be and isn't much of the time. This work develops the theory of naming and addressing in operating systems and programming languages in a general and implementation-independent manner. It is does the "algebra" first. Although space does not allow a detailed review of the paper, we do see that roughly three levels of naming are required in operating systems. Saltzer provides a framework for the sharing of data and programs in a computing environment. Although Saltzer does not consider the problems of naming and addressing in computer networks, many of the concepts that will be needed are discussed. These might be characterized as follows:

1. A name space that allows sharing among independently running programs
2. A name space that allows programs to logically refer to their variables regardless of where they are in memory
3. A name space that represents the program in memory
4. A path from the processor to the memory

The first has a "universal" scope of the whole computer system and encompasses all files (program or data) that are executing or may be executed on that system. This name space allows one to unambiguously refer to any programs and data files on the computer and in some systems, such as Multics, objects within these. The second provides a name space that allows the programmer to logically construct programs independent of memory size and location. This space creates a virtual environment that may assume resources that exceed those of the underlying real computer. This logical environment is then mapped to a real computer where the operating system provides the facilities that create the illusion of the virtual environment. (For example, virtual memory provides location independence and the illusion of greater memory than actually exists, and processor scheduling gives the illusion of a multiprocessor system.) The hardware then provides a path from the processor to the appropriate memory location.

For the naming of files and programs, a hierarchical approach was adopted rather quickly, consisting of a root directory, subdirectories, and finally primitive names. This was called a *pathname* because it defined a path through the directory structure. If a file was moved in this structure, its primitive name remained the same, but its pathname changed.

6 This might seem like ancient history here, but I highly recommend that you dig out this reference.)

X.25 and the ITU

In the mid-1970s, the PTTs rushed to get in the packet-switching business. Mostly to defend their turf because organizations that weren't telephone companies were building networks than because they thought it was a good business opportunity. After all, data traffic would never come close to the kind of volumes as voice traffic! The PTTs proposed a network design along the lines of the ARPANET or NPLnet using a new protocol, X.25, as their answer to the research networks. X.25 addresses have the same semantics as a telephone (no surprise). The structure of an X.25 address is similar to that for telephones, consisting of a country code, followed by a network number and DTE (host) number. But the allowances for growth were very small, allowing only ten networks per country. A distinct "group-id" field in the X.25 header identifies particular connections from this DCE. The address is the name of the interface over which all connections with that DTE pass.

The "East Coast elite" screwed up the ARPANET addressing because they were from Boston. In Boston, there is only one way to get anywhere, and so it is easy to confuse that a route and an address are the same thing. If they had been from the Midwest where everything is on a grid and there are many paths between two points, they would have known that a route and an address are two entirely different things.

It isn't true, but it makes a good story!

The Evolution of Addressing in the Internet: Early IP

Or Is it?

In New England, the way to get some place is to take out the map, find the destination, and trace a route back to where you are. Follow the path. Not unlike Internet routing.

In the Midwest, the address gives you a good idea where the destination is relative to where you are. You start in that direction, using addresses along the way to indicate whether you are closer or farther from the destination. Interesting. Forwarding without routing.

As previously discussed, the origin of the Internet's convention that addresses name interfaces derives from the implementation of the original IMPs. Although this was common practice for the small data networks of the time, it is basically the same as the telephone company. Using the telephone example was a reasonable first approximation, and it wasn't at all obvious how the street address example contributed anything to the solution (although there was a nagging sense that it should). Unlike telephone addresses, ARPANET addresses were only route dependent for the last hop. (In the phone system, there were multiple routes above the exchanges, although automatic rerouting is relatively recent.) It was clear that computers would have different requirements than telephones. We have already seen the problem of dual homing. But it was realized the problems of naming applications that were seen in operating systems would be more complex in networks.

The development of TCP and IP began in the mid-1970s to fix problems with the original Host-to-Host Protocol. As far as addressing was concerned, the only immediate problem that had to be dealt with was that there weren't enough of them. So, the IP specification expanded the address to 32 bits and slightly generalized the semantics of the address so that it named the "interface" rather than an IMP port.

The problem continued to be discussed. John Shoch published an important paper (Shoch, 1978). (Shoch's paper had been circulating within the ARPANET community for over a year before it appeared in print.) Shoch recognized (as so often scoffed at) that

> Taxonomies and terminologies will not by themselves, solve some of the difficult problems associated with the interconnection of computer networks; but carefully choosing our words can help us to avoid misunderstanding and refine our perceptions of the task.

Shoch posited that three distinct concepts were involved: names (of applications that were location independent), which were "what we seek"; addresses (that were location dependent), which indicated "where it was"; and routes (which were clearly route dependent), which were "how to get there." Shoch made clear what many had been thinking but didn't know quite how to say. At the time, Schoch was working at Xerox PARC with Robert Metcalfe on the development of Ethernet and related projects. Shoch points out in his paper how the naming in networks parallels what is found in computing systems: Namely, that applications had names that were independent of memory location and made sense to human users, whereas programs used virtual memory addresses that allowed their code to be placed anywhere in memory and were mapped to the actual physical memory location (routing) by the hardware. It seemed to make a lot of sense.

A few years later (1982), the other most often cited paper on network addressing appeared, Jerry Saltzer's (RFC 1493) "On the Naming and Binding of Network Destinations." This is a most curious paper. Saltzer sets out to apply to networks the same principles he applied to operating systems and makes a major contribution to the problem. Saltzer notes that there are four things, not three, in networks that need to be named (just as there were in operating systems): services and users, nodes, network attachment, and paths. Saltzer carefully lays out the theoretical framework, defining what he means by each of these. After noting some of the issues pertinent to the syntax of names, Saltzer observes:

> The second observation about the four types of network objects listed earlier is that most of the naming requirements in a network can simply and concisely be described in terms of bindings and changes of bindings among the four types of objects. To wit:

> 1. A given service may run at one or more nodes, and may need to move from one node to another without losing its identity as a service.
>
> 2. A given node may be connected to one or more network attachment points, and may need to move from one attachment point to another without losing its identity as a node.
>
> 3. A given pair of attachment points may be connected by one or more paths, and those paths may need to change with time without affecting the identity of the attachment points."

It would appear that Saltzer is suggesting that we name the objects and track the mappings (i.e., the bindings) between them. Notice the parallel between this list and Saltzer's list for operating systems earlier in this chapter.

> Each of these three requirements includes the idea of preserving identity, whether of service, node or attachment point. To preserve an identity, one must arrange that the name used for identification not change during moves of the kind required. If the associations among services, nodes, attachment points and routes are maintained as lists of bindings this goal can easily be met.

Again Saltzer is pointing out a very important property (i.e., that the names given to objects must be invariant with respect to some property across the appropriate scope). In particular, service or application names do not change with location, node names do not change for attachment points within the scope of their location, and attachment points do not change as the ends of their routes.

This expands a bit on Saltzer's words, but it seems reasonable to assume that Saltzer recognized that names would not be assigned once and for all. And if they could change, there must be rules for when and how they could change. In fact, he states quite rightly that even if a name is made permanent, this "should not be allowed to confuse the question of what names and bindings are in principle present." He then reviews that "to send a data packet to a service one must discover three bindings" [given the name of a service]:

1. Find a node on which the required service operates
2. Find a network attachment point to which that node is connected
3. Find a path from this attachment point to that attachment point

From Saltzer's description, there is a name for each of these four and tables that maintain the bindings between the names:

1. Service name resolution, to identify the nodes that run the service
2. Node name location, to identify attachment points that reach the nodes found in 1
3. Route service, to identify the paths that lead from the requestor's attachment point to the ones found in 2

Saltzer then illustrates his points with a couple of examples that for Saltzer present problems in applying his model. He then concludes that regardless of what one may think of his analysis, "it seems clear that there are more than three concepts involved, so more than three labels are needed…." And finally, in his summary, he points out there is a strong analog between what he has described and the concepts found in operating systems.

This seems to answer our first question of what has to be named: Applications require location-independent names. This is Schoch's *what*. This allows the application to be moved without changing its name. That name maps to a node address that indicates *where* the node is and the application can be found, with each router maintaining a forwarding table that maps an address to a "next hop" (i.e., next node address). But then Saltzer lumps the next step in with routing. He clearly knows that a point of attachment address is needed, but he doesn't clearly distinguish how it differs from a node address. As noted previously, it was obvious that the solution to the multihoming problem was that a *logical* address space was needed over the *physical* address space. But then Saltzer follows the operating system model too closely and notes that there is a mapping of applications to nodes, a mapping of nodes to points of attachment, and then a mapping to routes as a sequence of points of attachments and nodes.

Saltzer misses a case that is unique to networks and key to understanding: In networks, there can be multiple paths (links) between adjacent nodes. Saltzer can't be faulted for missing this. Multiple paths to the next hop were rare or nonexistent when he was writing. Let's supply the answer.

After selecting the next hop, the router must know all the node address to point of attachment address mappings of its nearest neighbors so that it can select the appropriate path to send PDUs to the next hop.

Routes are sequences of node addresses from which the next hop is selected. Then the router must know the mapping of node address to point of attachment address for all of its nearest neighbors (the line in Figure 5-2) so that it can select the path to the next hop.

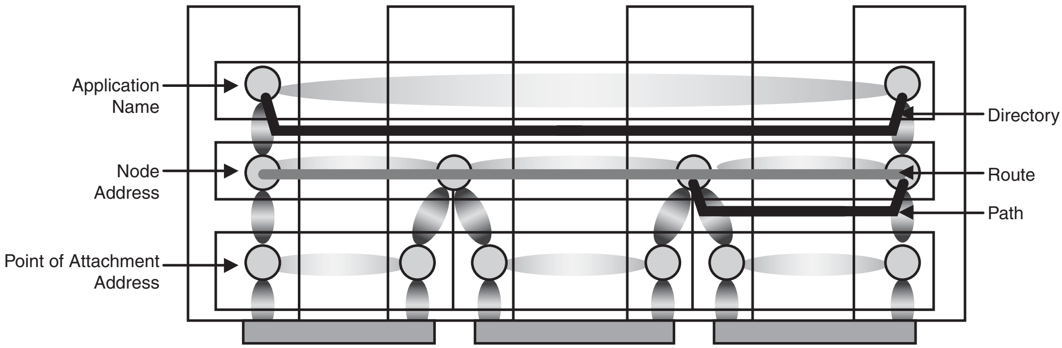

Figure 5-2 Addressing for a network requires at least an application name, a node address, and a point of attachment address. Directory maps application names to node addresses, routes are sequences of node addresses, and multiple paths between adjacent nodes require mappings between node addresses and point of attachment addresses.

"Routing" is a two-step process. A route is a sequence of node addresses. The next hop is chosen to the next node address. Then the mapping of local point of attachment addresses to the point of attachments of nearest neighbors for the next hop is needed to select which path to the next hop is selected. Looking at the figure, we see these bindings:

1. *Directory,* mapping of application names to node addresses to find where the application is. This is an example of the name-resolution or directory protocols discussed in Chapter 4, "Stalking the Upper-Layer Architecture."

2. *Routes,* as a sequence of node addresses calculated by the routing algorithms to generate the next hop

3. *Paths,* selected from the mapping node address to point of attachment address of the nearest neighbors (i.e., next hops)

Interesting! 1 and 3 are the same mapping! The path is also an example of a name-resolution service, just like the directory. The path database is smaller than the directory database, and the syntax of the names are a bit different, but the same mapping nonetheless. They both track name mappings that are "one hop" from each other (relative to their layer).

It was clear that a network address (i.e., node address) needed to be location dependent and application names should be able to be location independent. What about *point-of-attachment* (PoA) addresses? Traditionally, the PoA corresponds to the data link layer address. From the point of the view of the nodes, it doesn't matter. All the nodes (routers) require is that PoA addresses of nearest neighbors are unambiguous. All PoA addresses don't have to come from the same address space and probably won't. Different protocols in different layers of less scope are possible and allowable. Any two connected nearest neighbors will have addresses from the same address space. (They have to because both ends of the communication use the same protocol, by definition.) But not all PoAs on the same router or host must be from the same address space. Whether a PoA address space will be flat or location dependent will depend on the protocols and scope of the PoA layers. Location dependence is a property that facilitates scaling within a layer by reducing the complexity and combinatorial properties of routing.

But what is curious about this paper is that Saltzer lays out the answer very clearly. When addressing is discussed in networking meetings, this paper is cited by almost everyone. The paper is almost revered. But the Internet architecture has no application names and no node addresses (a well-known socket is at best a suffix for a network address, and URLs show signs of backing into being a form of application name *within* http). The Internet has only PoA names, and routes. Saltzer says clearly that PoAs and routes are not enough. It is clear that the fundamental problem with Internet addressing is that it is missing half the necessary addressing architecture. Why then has the Internet not taken Saltzer's advice, especially given how Saltzer lays out the principles so clearly?

The XNS architecture developed at Xerox PARC for networks of LANs, and later used by IPX for Novell's NetWare product, had a network address that named the system, not the interface. This was the first commercial architecture to fix the addressing problem created by the IMPs. But, Xerox's decision to keep the specifications proprietary limited its early influence. At the same time, the decreasing cost and increasing power of hardware reduced the need to fix the problem in IP.[7] Later this same solution would be picked up and used by OSI.

The deployment of IP overcame the address space problems of NCP. Thirty-two bits of address space was more than enough. However, IP retained the semantics of the IMP port address and named the interface (see Figure 5-3). The primary reason for this is unclear. IP was first proposed in about 1975 and changed very little after that first draft. The only known problem at that time was with the semantics of the address, as exemplified by the dual-homing problem described earlier. The Saltzer analysis shows that multihoming isn't supported for routers, let alone hosts. But because the Net was small enough

7 Once again, Moore's law perhaps causes more trouble than it helps by allowing us to ignore the scaling problems of the address space for so long that the network grew so large that solutions became more daunting. It is curious, given the DoD sponsorship of the early Internet, that there was not more pressure to fix such a fundamental capability. Worse, users had come to believe that addresses could be used as names. "Experts" demanded that IP addresses not change no matter where they were attached to the network: a fine property of names, but not of addresses.

without multiple paths between adjacent nodes, it wasn't a problem that Moore's law couldn't solve. (And when multiple paths did arise, it caused problems but band-aids were found for them.) The problems of multicast and mobility were many years off. It was understood that a change would be necessary, as was our repeated caution about the importance of getting addressing right. No one felt they really understood addressing well enough. It seemed prudent that a more complete understanding was necessary before making the change. We still didn't understand what location dependence meant in a network. It seemed prudent not to do anything until there was a better understanding of what to do. Even in the early 1980s, when NCP was removed and IP became the only network layer protocol, the Internet was still for the most part a network of universities and R&D organizations, so such a major change was still something that could be contemplated.

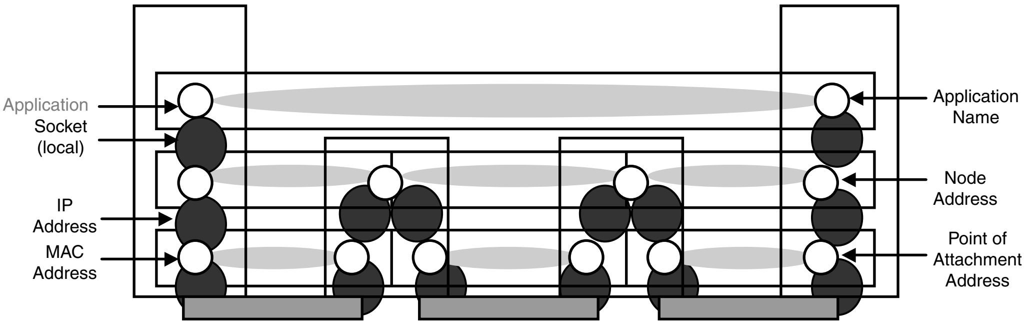

Figure 5-3 Mapping Saltzer's concepts to the Internet shows that half the required identifiers are missing (application names and node addresses) and one is named twice (point of attachment).

When IP was defined, some structure was imposed on IP addresses by dividing the address space into blocks of Class A, B, and C (Figure 5-4). (As other authors do, we will ignore the existence of Class D and E addresses for now.) The classes of IP addresses are intended to be assigned to networks with different numbers of hosts: Class A for the really big ones, Class B for the middle-size ones, and Class C for the really small ones. And of course, within a Class A network, Classes B and C can be used to provide a rudimentary form of location dependence.

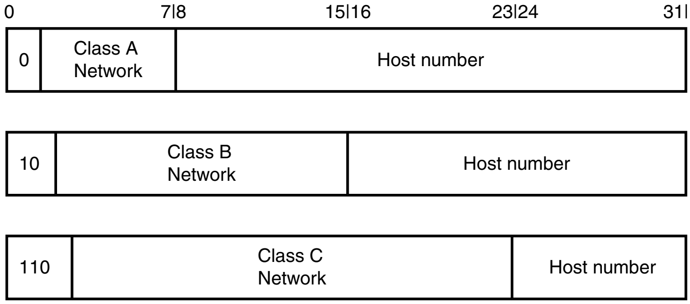

Figure 5-4 IP address format.

But these were allocations of size, and although they might be used to impose location dependence within a given network, no consideration was given to doing it across networks. Blocks of IP addresses were for the most part handed out in the order requested. 128.89 might be on the East Coast of the United States, and 128.90 might be in Hong Kong. So in fact, IP addresses were more like names than addresses. There was no structure or plan to assigning the network part of an IP address. It was assumed that addresses would be assigned in a location-dependent manner within the networks (an assumption made unnecessary by Moore's law) and that the number of networks would remain relatively small. There was no planning for tens of thousands of networks organized into tiers of providers.

As the problems of configuring networks for large organizations grew, subnetting was introduced. Subnetting takes part of the host-id portion of the address and uses it to represent subnets within the Class A or B address (or Class C, but they are pretty small for subnetting). This provides topological-dependent addresses within an organization; outside the organization, however, it is of no help.

OSI and NSAPs

Using the experience from the ARPANET and early Internet, OSI made some major strides in working out the theory of naming and addressing. It also made some major mistakes. (Although there are several interesting aspects to the OSI addressing concepts.) The amount written on it is fairly voluminous and impenetrable. We will consider the basics as briefly as we can and only elaborate on concepts or lessons that we need to carry forward. First, let's dispense with what OSI got wrong: The Europeans were intent on making X.25 the OSI answer to the network layer and not using any experience from the United States, even if it was improving on the lessons learned in the Internet. Consequently, they forced into the OSI architecture fundamental constructs to reflect X.25. As an example, in OSI an (N)-connection is defined to be shared state among (N+1)-entities, not the shared state among (N)-entities. But in spite of such fundamental problems, it was possible to resurrect the beginnings of a fairly reasonable addressing architecture, even if the errors did cause the definitions to get a bit convoluted at times.

OSI spent considerable time developing a theoretical framework for the architecture. This was not the "seven-layer model." But an earlier section of the reference model that defined the common elements that all layers would have. The understanding was that there were common elements but different functions in each layer, in line with the Dijkstra concept of a layer. This effort was beneficial because it was an attempt at an "algebra" that clarified the nature of the problem provided insight into the solutions. It is unfortunate that politics could not be kept out of it. However, it seldom helped those who tried to use the standards because the standards seldom reflected the insights that had been gained. (The U.K. delegation insisted that any "tutorial material" should not be included. It seemed that they were intent on making the documents as difficult to use as possible.) There are two aspects of this theory: the general architecture as it relates to addressing and the specifics of addressing in the network layer.

Terms, Terms, Terms

Entity might seem like a pretty innocuous term. It was supposed to be. There was great fear that the *model* not specify implementation. Therefore, any term such as *process, program, task, procedure,* and so on that might be construed as specifying how it must be implemented was unacceptable. I have noticed recently that others have been driven to the same term.

The general OSI architecture consists of (N)-layers. (Of course, in the specific architecture constructed from this theory, the maximum value of N was 7.) Each system in the network contains elements of these (N)-layers, from 1 to 7. The intersection of an (N)-layer with a system is called an (N)-subsystem. Within each (N)-subsystem, there is one or more (N)-entities (Figure 5-5). An (N)-entity is the protocol machine for that layer. A (N)-subsystem could contain more than one (N)-entity (e.g., different groups of users) or (N)-entities of more than one kind (i.e., different protocols). In other words, an (N)-subsystem is all the modules in a system relating to a particular layer, protocol machines, management, buffer management, and so on. Having a term for everything in a system associated with a given layer proves to be quite useful.

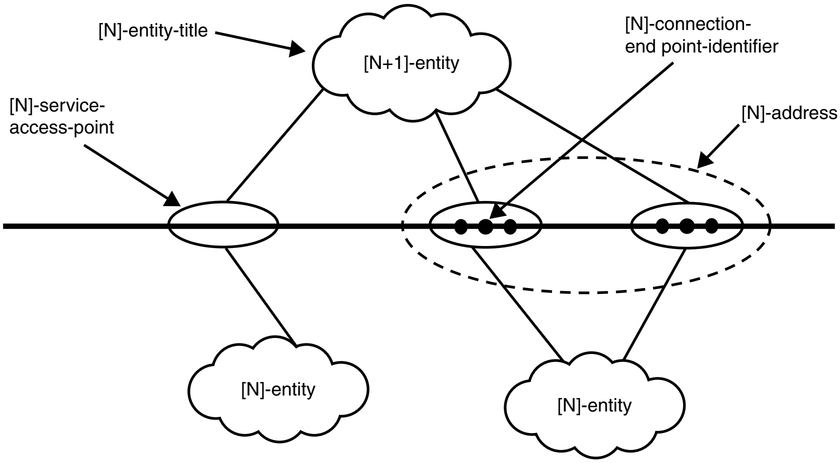

Figure 5-5 Entities, service access points, and identifiers.

As mentioned, an (N)-connection was defined to be "an association requested by an (N+1)-entity for the transfer of data between two or more (N+1)-entities." In other words, an (N)-connection went from one (N+1)-entity (in an (N+1)-layer) down to an (N)-entity across to an (N)-entity in another system and up to the (N+1)-entity in the remote system. (Pushing this definition were the Europeans attempting to legislate the X.25 view.) This tightly binds the shared state in the (N)-entities to the shared state in the (N–1)-entities. But it is important that it be possible to decouple the two, so that the shared state at (N–1) can be lost without affecting the shared state at layer N. This definition makes that difficult.

Later realizing that they needed a name for the relation between the (N)-entities (what the definition of a connection should have been), they defined an (N)-association as "a cooperative relationship among (N)-entity-invocations."[8] Yes! In OSI, associations were connections, and connections were what association should be. But then I have never known a standards organization yet whose arrogance didn't get it into this sort of doublespeak.

The (N)-connection crossed the boundary between an (N+1)-layer and an (N)-layer at an (N)-service access point or (N)-SAP. (N)-SAP-address identifies an (N)-SAP. (This is why one encounters the term *SAP* in other standards.

---

8 Quite correctly, OSI tried to distinguish between type and instance. A protocol in a subsystem was the type, whereas a specific flow or connection using that protocol would be an instance or instantiation of the protocol. One connects to TCP (type), but each state machine along with its TCB represents an instance of TCP. So when the dust settled, the (N)-entity was the type, and the (N)-entity-invocations were the instances.

Notice how a SAP tries to be a port or interface.) An (N)-SAP was bound to one and only one (N)-entity at a time. If an (N)-entity needed to have an identifier, it was called an (N)-entity-title. (The pedants said it couldn't be called a "name" because addresses were also names.) An address was a location-dependent name. So, the term *title* was used for location-independent names. Associated with an (N)-SAP-address were one or more (N)-connection-endpoint-identifiers whose scope was the (N)-subsystem. An (N)-CEP corresponded to a single connection to an (N)-entity. The (N)-SAP-address was supposed to be an X.25 DTE address. The (N)-CEP-identifier corresponds to what many protocols or IPC facilities call port-ids, whereas for the PTTs it was the X.25 group-id. (Group-ids are similar to ATM virtual path-ids or MPLS tags. All three of these derive from the same telephony lineage). So, an (N)-SAP was really a port, an interface.

This constraint along with the definition of connection caused a number of problems. It implied that all the bindings between (N)-entities in a system had to be preallocated before a connection request was made. This, of course, makes dynamic assignment and resource allocation essentially impossible. By 1983, it was already believed that the reference model was too far along to be changed. So rather than simply fix the definition of connection and make the structure simpler, a level of indirection was created[9]: An (N)-address was defined as a set of (N)-SAP-addresses. But worse, the OSI "address" also identifies the interface. The one thing that most were trying to avoid. (In a committee, *consensus* never means that issues are resolved, only that progress can continue until someone finds a reason to raise the issue again.)

Another problem was discovered in how we thought we would build addresses. Initially, it was assumed that an (N)-address would be formed from an (N–1)-address and (N)-suffix, allowing addresses from a higher layer to infer addresses at lower layers. This was a fairly common approach found in operating systems. It can be found in early versions of the OSI reference model see, for example, ISO TC97/SC16/N117 (1978) or N227 (1979) and in the Internet today. It is a bad idea in networks. And why it is a bad idea is clear from its use in operating systems. Constructing names in this manner in operating systems has a name. They are called *path*names, and therein lies the problem. It defines a path. It defines a single static path within the system and then to the application when, in fact, there may be multiple paths that it should be possible to choose dynamically. It can be done, but essentially one must ignore that it has been done. Recognizing that it is a lot of redundancy for very little gain and may compromise security. It works in an operating system because there *is* only one

9 Yes, there is nothing in computer science that can't be fixed with a level of indirection. (sigh)

path *within* the operating system from one application to another. This is exactly what we wanted to avoid from our analysis of Saltzer. Hence, any addressing scheme that, for instance, creates a network address by embedding a MAC address in it has thwarted the purpose of the addressing architecture. There can be a relation, but the relation cannot be tied to the path. This is still considered a quite normal approach to take to forming addresses.

However, all was not lost. Or more to the point, the problems in the network layer were much more complicated. The U.S. delegation was insistent that there would be a connectionless network protocol that built on the experience of IP, and the Europeans were intent that the future of networking would be a connection-mode protocol (i.e., X.25) and that connectionless would as limited as possible. They attempted to work out an architecture of the network layer that could accommodate both. The resulting standard, called the *Internal Organization of the Network Layer* (IONL), shed considerable light on what the two warring factions were wanting and provided technical insights (ISO 8648, 1987). Although the language of the document can be quite impenetrable to the uninitiated, every configuration described in it has since turned up in one form or another. The IONL was a very useful exercise in working out how real-world situations would be handled within an architecture. The Europeans had to admit that X.25 was only an interface to the network (after all, it was the title of the Recommendation) and as such only provided access to a subnetwork. It was finally worked out that the primary function of the network layer was to make the transition between the subnetwork-dependent protocols and provide a service that was independent of the subnetwork technology. To do this could require up to three sublayers depending on the configuration and the underlying media:

- A *Subnetwork Access Protocol* (SNACP) is a protocol that operates under constraints of a specific subnetwork. The service it provides may not coincide with the network layer service.
- A *Subnetwork Dependent Convergence Protocol* (SNDCP) operates over a SubNetwork Access protocol and provides the capabilities assumed by the SNICP or the network layer service.
- A *Subnetwork Independent Protocol* (SNICP) operates to construct the OSI network layer service and need not be based on the characteristics of any particular subnetwork service.

Although a lot of this structure may seem (and was) politically motivated, there were several major technical insights. For our purposes, the most important of which was that there was a "subnetwork PoA" (an SNPA or "the wire") that had an address with a scope that had to span only the particular subnet. A system might have several SNPAs that mapped to an NSAP address. The NSAP address as constructed by the IONL was, in fact, the (N)-entity-title. The (N)-directory, or in the this case the N-directory (N for network) (i.e., the routing information) maintained a mapping between the SNPA-addresses and the NSAP-address. This mapping provides a level of indirection between the physical addressing of the wire and the logical addressing of the network. This level of indirection provides the flexibility required for addressing to accommodate all the configurations and services necessary. This is repeated later, but it is worth observing now:

A network address architecture must have at least one level of indirection.

Like operating systems, there needs to be a transition between logical and physical addressing. As we have seen earlier from our interpretation of Saltzer in a network, two transitions are required: one in the network layer between SNPAs and NSAPs, between route dependence and route independence but both location dependent; and again between NSAPs and application entity titles, between location dependent and location independent.

The NSAP addressing structure attempted to solve two problems: accommodate a wide variety of existing address formats and set out a location-dependent address space. The address format of an NSAP is shown in Figure 5-6.

AFI	IDI	DFI	Org	Reserved	Rowing Domain	Area	System	Sel
1	2	1	3	2	2	2	6	1

Figure 5-6 OSI NSAP format for the United States

The address space is organized by countries. The country codes are assigned by an ISO standard. Each country is then allowed to organize its own space. In the United States, a rather elegant solution was found that avoids a requirement for an active centralized authority. There is an existing ANSI standard of organization identifiers. These are used after the country code. To get an assignment of NSAP addresses, one merely has to get an organization-id (which many companies would already have for other purposes), the organization-id goes after the country code the rest of address space can be used by the organization. This creates a provider independent address.

The AFI specifies the format of the IDI and the addressing authority responsible for the IDI. The AFI could select X.121, ISO DCC, F.69 (telex), E.163 (PSTN), E.164 (ISDN), ISO 6523-ICD, or Local. The DFI contains the country code; Org is the ANSI organization identifier. Routing Domain and Area are the topological routing information. The Reserved field was to allow for another level of the routing hierarchy if it was required. The System field is six octets so that an Ethernet address can be used. If this is interpreted too literally it will force the NSAP to name the interface, not the network entity as intended. (Groan. In a committee, it is sometimes difficult to keep people from wanting to do it wrong.) Although this format incorporates location-dependent elements, it does not indicate *where* in the topological structure of the network the address is. It doesn't help determine "which way" to send a PDU or if two destinations are "near" each other. This address is location dependent more in the sense of Boston than Chicago!

This address space reflects the growing understanding of addressing. The IP address space was mostly concerned about identifying networks and hosts without much concern for their relative position in a topology. At this point, although it was understood that something analogous to a "Chicago address" would be useful, no one had any idea how to do such a solution. It really wasn't understood that addresses needed to be topological (in the mathematical sense). With the NSAP address space, there is more concern that a topology is reflected in the address space by including the DFI or country identifier and organization identifier. However, this topology is not completely satisfactory either. This scheme assumes that the routing domains are below the level of organizations. This would be the case for large companies but hardly for smaller ones. Similarly, there are cases where being able to group several small countries under a single regional domain would be useful and conversely, breaking up larger countries into multiple domains would also be useful. Or was the address format the result of a compromise between the "X.25 faction" and the "IP faction"? This raises the question of what is the relation between provider-based addresses and provider-independent addresses. Clearly, provider-based addresses reflect the topology of the provider's network. What does a provider-independent address space reflect? The usual reaction is to immediately leap to a geographic approach. But is this the only one? Are there others that are not totally geographic in nature?

There were other minor problems: The format assumes that organizations are a proper subset of countries. (Although one could assume that a company's presence in another country has a different value for these fields.) The only other problem with the address format is the selector field, which supposedly identifies the protocol in the layer above. The OSI Architecture group had taken the position that it was counter to the architecture for an (N)-protocol to identify an (N+1)-protocol. A horrid layer violation. At the time, this was seen as relating to addressing. So rather than a field in the PCI, the Network Layer group made it a field in the address. Neither solution actually can be used to identify the upper-layer protocol, regardless of whether it is a layer violation. Such a field can only identify one occurrence of a protocol in the layer above bound to that address. (Admittedly, this does not happen often, but as with many other "rare" events, when it does it can make things cumbersome if the addressing has not been done right.) There are configurations where more than one instance of the same type of protocol bound to the same network address is necessary. As we saw in Chapter 3, "Patterns in Protocols," one could argue that we weren't seeing the problem correctly, that the field identifies the syntax of the protocol. However, we will find later that both interpretations are incorrect and such a field is unnecessary.

But all in all, OSI progressed the state of the art and tried to take Saltzer's advice, even if the ill informed stuck a MAC address in the NSAP. It recognizes PoA addresses, node addresses, and as we shall see later, application names extending Saltzer's scheme in an important way.

Communism is the longest most torturous path from capitalism to capitalism.

—Joke that circulated in Eastern Europe at the end of the 1980s

Addressing in IPv6

So let's consider the addressing architecture for this new IP in some detail. The IPv6 addressing specification is very emphatic: "IPv6 addresses of all types are assigned to interfaces, not nodes." However, it then observes that since any interface belongs to a single node, a "unicast address may be used as an identifier for the node"—a painful example of having heard the words but not understanding their implication. We will assume that a *node* is synonymous with a *system* and assume an interface is generalized from the IMP port from which it originated; that is, an interface is the path from the bottom of the IP layer through any lower-layer protocols to the physical media connecting to another system.

One exception to this model is granted to allow multiple physical interfaces to be assigned the same address as long as the implementation treats these as a single interface when presenting it to the IP layer. In other words, parallel interfaces or spares can be treated as a single interface. This would seem to indicate that this is a degenerate form of anycast address—and another kludge to make up for not having node and PoA addresses.

The Various Address Types

Although IPv6 supports a number of address formats, the format we are most interested in will be the Aggregatable Global Unicast Address. This is what most people will think of as an IPv6 address. But before we do that, let's dispense with anycast and multicast addresses and a couple of other address types that are unique to IPv6, the link-local and site-local addresses.

There are three types of IPv6 addresses (RFC 2373, 1998):

- **Unicast.** An identifier for a single interface. A packet sent to a unicast address is delivered to the identified by that address.
- **Anycast.** An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to an anycast address is delivered to one of the interfaces identified by that address.
- **Multicast.** An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to a multicast address is delivered to all interfaces by that address.

Anycast addresses. Anycast addresses are syntactically indistinguishable from unicast addresses. According to RFC 2373, a unicast address is turned into an anycast address by having multiple interfaces assigned to it. This is not quite the case. The nodes to which the interfaces belong must be explicitly configured to be aware of this. So, in fact, it is not multiple assignment that makes it an anycast address, but configuring the nodes to know that it is multiply assigned (an enrollment phase function). The RFC imposes two constraints on the use of anycast addresses: They cannot appear as the source address in any IP packet (reasonable); and they cannot be assigned to hosts, only to routers (less so). This latter constraint is perhaps the most odd because considerable use could be made of anycast addresses in applications. The subnet prefix of an anycast address is the longest prefix that identifies the smallest topological region of the network to which all interfaces in the set belong.

How this is supposed to work is not quite clear. For different nodes to be configured to be aware that multiple interfaces have the same address requires protocol to be exchanged. No such protocol has yet been defined. Clearly, any use of this facility must be stateless because successive uses may not yield PDUs being delivered to the same destination. This is another kludge to get around not having node and PoA addresses.

Multicast addresses. Multicast addresses include two subfields: A flags subfield that has 3 unused bits and a single bit that indicates whether this group address is permanently assigned; and a scope field that currently defines whether the scope of this group address is the local node, the local link, the local site, the local organization, or global. Permanently assigned multicast addresses have global scope; that is, the scope field is ignored. IPv6 defines a multicast address as "an identifier for a set of interfaces." There will be more to say on the nature of anycast and multicast "addresses" in Chapter 9, "Multihoming, Multicast, and Mobility."

Link- and site-local addresses. A link-local address essentially consists of the 10-bit format identifier in the high-order bits and a 64-bit interface identifier in the lower-order bits, and 59 bits of nothing in the middle. This address form is for "local" use only. The RFC suggests that link local addresses "are designed to be used for addressing on a single link for purposes such as auto-address configuration, neighbor discovery, or when no routers are present." The use of the term *link* implies that they are intended to be used on, for example, a single LAN segment (i.e., within a single subnet).

A site-local address, although similar to the link-local form, was to correspond to what private address space was in IPv4 (e.g., net 10). The subnet identifier distinguishes the multiple subnets within the same "site."

In 2003, there was a movement within the IPv6 working group, over considerable objections, to delete site-local addresses from the specification. There were strong feelings against the use of private address space within the IETF. Some believed that this "balkanized" the Internet, which it does, and contradicted some mythic ideal of the "spirit of the Internet." Engineering on belief rather than empiricism is always dangerous. As we have seen, NAT and private address space only break protocols in an incomplete architecture and primarily indicate bad design choices. Or to paraphrase Buckminster "Bucky" Fuller, NATS only break broken architectures.[10] As it turns out, private address space is a natural part of any complete architecture and poses no dangers and, in fact, has many benefits.

However, the removal of private address space from IPv6 would seem to represent a very large deterrent for corporate adoption. Although NATs do not provide complete security, they are an important element in securing and exercising control over a subnet. It is hard to imagine corporate IT directors giving up this simple measure to be replaced by elaborate and as yet unproven IPv6 security mechanisms. Once again, the IETF seems to have cut off its nose to spite its face.

In addition, address formats are defined for carrying NSAP and IPX addresses. (Although there is little expectation that these will ever be used.)

IPv6 also allocates two special addresses: 0 and 1 (or to be precise in the IPv6 notation, 0:0:0:0:0:0:0:0 and 0:0:0:0:0:0:0:1). The unspecified address is 0 and

10 Bucky said, "Automation only displaces automaton."

"indicates the absence of an address." The unspecified address can never be used as a destination but may appear as the source address for a sender who does not have an address yet. (It is not clear what you do with such a PDU (you can't respond to it), but that is not important. The loopback address is 1 and is used by a system to send a PDU to itself. It may only be used as a destination address and then must be sent back to the sender. It should never be relayed to an address other than the sender, and the loopback address must not appear as a source address in a PDU.

IPv6 Unicast Addresses

It is the aggregatable unicast address over which there has been the greatest amount of debate. This debate has evolved around the decision that the IP address will continue to label an interface. This was complicated by the politics surrounding IP and OSI. By the time IPv6 was proposed, some had realized that addresses had to be topological. But they thought topology meant the graph of the network. Mainly, they were concerned that the addresses had to be aggregatable. As discussed in this chapter, the problem with the IPv4 address space is not so much the lack of address space but the growth of the routing tables. To reduce the number of routes that must be stored requires the ability to aggregate them. For example, the post office aggregates routes based on the hierarchy of the address (i.e., country, state/province, city, street, street number, and so on). When a letter is mailed, the first post office has to look at only the first couple of levels of the hierarchy to know where to send it. It does not need to figure out precisely where the destination is; it merely has to send the letter in the right direction. Similarly, some sort of hierarchy was required for IPv6 addresses. As we saw, CLNP adopted such a hierarchy based on countries and organizations within them.

The Internet had the same problem that had faced OSI: a flawed architecture and a reactionary group of traditionalists who opposed any change to the concept that an address labels an interface. However, the Internet architecture was also weak in another area. The Internet architecture really only covered the network and transport layers (or in terms of the seven-layer model, the top third of the network, SNIC, and transport and only had an address for the bottom third). Above and below network and transport, there was not really any structure, so there was no convention for names or routes, as proposed by Saltzer. This led to a tendency to try to solve everything in the network and transport layers.

Names and Addresses

Giving them the benefit of the doubt, it might be closer to the truth that people had become so used to addresses being names that they used them as names and expected that IP addresses could act like both names and addresses. After all, they had never been taught anything different. There are no textbooks in networking that cover what should be named.

The IPv6 effort determined the PDU header format and the size of the address field years before they determined what an address was to look like ("arithmetic before the algebra"). Also, most of the people involved in IPv6 were initially working under the misconception that the number of addresses was the major problem to be solved. There were some initial proposals that were similar to the NSAP address. But because the IPv6 address had to name an interface, to be aggregatable the addresses had to be provider-based. This had the unacceptable consequence that if one changed providers all hosts on your network would have to be re-addressed. (It is significant that the term commonly used in Internet circles is *renumbering* rather than *re-addressing,* which indicates that they think of it as enumeration or naming rather than addressing or changing location.)

As noted previously, a network architecture must make a transition from logical to physical at least once. The Internet architecture has no such transition. OSI had been "fortunate" enough that its traditionalist faction was X.25. That forced (or created the opportunity) to separate the physical address or subnetwork PoA from the network address. The Internet architecture did not really address the layers below network, and there was no X.25 faction. (Its traditionalists hung on to the IP of the "good old days.") Furthermore, the political climate was such that if OSI had done something, the Internet would either not do it or do the opposite and convince themselves there was a good technical reason to codify the old ways.[11]

This meant the possible solutions were severely limited. Therefore, any solution had to have an appearance of not doing what was most reasonable (i.e., a separation of logical and physical in different layers). Even though the idea and the solution had originated during the early development of the Internet and had been used by the, at least politically correct, XNS, it had last been used by OSI and was therefore unacceptable. (And yes, there are many rationalizations why this was not the reason.)

The developers working on the Internet had for many years realized that something needed to be done. But in the Internet, the "host" had always been the focus of attention. There had been several proposals (Curran, 1992; Chiappa, 1995) to name "endpoints." Chiappa defined an endpoint to be "one participant of an end-to-end communication, i.e., the fundamental agent of

[11] This reaction has always been perplexing: Why react with "do anything but what the 'opposition' has done" and fall prey to "cutting off your nose to spite your face;" rather than "let us show you how to get it right"? Is this a characteristic of crowd behavior? Or is it something else? This is not the only example.

end-to-end communication. It is the entity which is performing a reliable communication on an end-to-end basis." Chiappa et al. saw this as mapping fairly directly to the concept of "host." However, the use of *one* and *an* in the definition would seem to imply more a single protocol machine than a collection of them. This was definitely on the right track. Replacing the traditional semantics of an IP address with the semantics of an endpoint in the protocol would have gone a long way to solving the problems confronting IP. However, this did not meet with much acceptance, probably because the implications of continuing to name an interface with an aggregatable address had not yet dawned on many of the members of the Internet community. To replace the semantics of an IP address with the semantics of an endpoint smacked too much of OSI. This situation existed for several years, and then Mike O'Dell (O'Dell, 1997) made a valiant effort to separate the IPv6 address into "routing goop," which would change when the host moved and an invariant globally unique "end system designator" that identified "a system invariant of its interfaces as *in the XNS architecture*" (emphasis added). This led to an addressing format (Figure 5-7) where the interface-id was the end-system identifier and the rest was the "routing-goop," as follows:

Where:

FP	The format prefix
TLA ID	Top-level aggregation identifier (13 bits)
Res	Reserved (8 bits)
NLA ID	Next-level aggregation identifier (24 bits)
SLA ID	Site-level aggregation identifier (16 bits)
Interface ID	Interface identifier (64 bits), probably an EUI-64 identifier

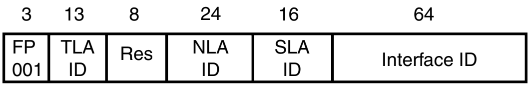

Figure 5-7 Format of an aggregatable IPv6 address.

The TLA, NLA, and SLA form the routing hierarchy of the address to the level of subnet, and the interface-id represents a completely independent globally unambiguous identifier. But, it does precisely what we found earlier that we didn't want to do: make it into a pathname.

This proposal came four years after the initial decision to develop IPv6 was made. By this time, memories had faded, there had been considerable turnover in the people involved, and the ramifications of the decision had finally become clearer to many. So with a little artful prose that did not open old wounds, O'Dell's proposal was able to thread the needle between the technical requirements and the political climate for a solution with only a moderate level of additional complexity. However, this was also unacceptable. The routing part of the IPv6 address is a path through a hierarchy of subnets, while the end-system designator has the same semantics as an IPv4 address. It names the interface (or to put it in other terms, the data link protocol machine). Here again, the IPv6 group found a way to take on the trappings of the solution without taking its substance to solve the problem. So although the form of O'Dell's proposal may be discernable in the IPv6 address format, the substance of it is not, and the problems remain.

At arm's length, an IPv6 address is similar to an NSAP in form. (…the longest, most torturous path….) It was common with NSAPs to use an IEEE 802 MAC address as the system-id, analogous to the use of an EUI-64 address as the interface-id. This was a case where the OSI architecture figured out something but the OSI Network Layer group, in a different committee, stayed with their intuitions. And as so often is the case in science, our intuitions were wrong. The NSAP format had four levels of hierarchy, whereas the IPv6 has three levels. OSI did not require "endpoints" or anything like them because it had application names. Because the IETF had no common application naming, it had, or thought it had, to solve everything in either the network or transport layer.

With IPv6, the routing part is not sufficient alone to distinguish a node. It can only distinguish the subnet but requires the interface-id to distinguish the node, whereas the interface-id alone can distinguish the interface. There are roughly 32 bits of redundancy in an IPv6 address (or enough for a couple of more levels in the routing hierarchy).

This approach will not support multihoming and mobility for the same reasons that IPv4 does not, and it greatly exacerbates the scaling problems in IP. The impact of these problems have been known about for a decade and a half, and now at this writing, with IPv6 barely deployed, they are already showing signs that are causing problems that are somewhere between severe and catastrophic. ("But she didn't do it and….")

Looking Back over IPv6

IPv6 has not instilled a lot of confidence among the cognoscenti. In fact, fear and trepidation is closer to the case. But deployment is beginning in fits and starts. There are still strong debates going on relating to the architecture of its addressing. For example, until very recently, some still argued that multihoming is being overly stressed. They contend that only a few hosts will need it and that a solution to multihoming is not really required; or because so few hosts need it, its cost should not be incurred by those who don't. This essentially ensures that any solution will be asymmetric and consequently will appear and be cumbersome and hence unacceptable.[12]

Superficially, it might appear that only a small percentage of all hosts require multihoming; that is, there are many more individuals connected to the Net than servers. However, even a small percentage of a large number can be a large number. But the real reason is that the ones that do need multihoming are very important to all the others. This is changing. As more companies come to rely on the Internet, the more they see multihoming as a necessity, and it is becoming more of a problem. Why is there an assumption that a solution must cost more, when in fact it actually costs less? It makes one wonder why people would argue that it is not very important. Why should there be so much debate over not doing multihoming? Redundant connections to the network would seem to be an "apple pie" issue. Of course, redundancy is a good thing, but not for the traditionalists. A simple solution to multihoming requires changing the semantics of the address. If multihoming is not important, there is no need for a change. So, the argument that multihoming is not important is actually more political than technical.

The concern over the addressing situation was sufficiently great that in 1999 that the IAB created an *Internet Research Task Force* (IRTF), the research side of the IETF) working group independent of the IPv6 work to consider namespace issues. This group met several times. There was a lot of discussion of endpoints as opposed to naming, but without a strong architectural model it was impossible to establish precisely what was required. Consequently, there was no consensus on the conclusions. But this effort seemed to focus the discussion on what has become known as the locator/identifier split. Many see the problem with the IP address is that its semantics have been overloaded with both locator meaning and identifier meaning, and if we simply separate them all the problems will be solved. Notice that they do not see that the IP address naming the interface is naming the same thing the MAC address does, but they also rely on the fact that the MAC address has greater scope than the IP address to make certain mobility-related capabilities work.

However, referring back to the Saltzer paper, this approach will give us an application name and a PoA address. Once again, it addresses the symptom but

12 This is a nice piece of electro-political engineering: Come up with very reasonable criteria that can only be met by an unacceptable proposal. This one is even better than the "lightweight transport protocol" red herring.

How Bad Could It Be?

The designers of IPv6 have blithely increased the size of the address without really considering the scaling implications of a full-blown IPv6 flat network. For several years, they ignored the router table expansion problem. They have continued to kludge the multihoming problem until the fall of 2006 when recognition of a looming crisis predicted dire consequences. After about ten days of considering that a more in-depth investigation was warranted, they fell back into the artisan response of looking for another band-aid.

In addition, some experts are concerned that router table calculations for the much larger v6 address will take much longer, greatly shortening the period between calculations. There is some question as to whether the effects of new forwarding tables once calculated would have time to take effect before it was time to recalculate. If the effects of the new forwarding table have not had time to "settle" before a new calculation begins, the input for the new calculation will be based on transient conditions, increasing the likelihood of unstable behavior.

Or more starkly, when a failure in the Net causes a router table computation, the Net will continue using the old tables while the calculation is made. The longer the calculation takes, the longer traffic is *not* responding to the failure, compounding the situation so that by the time the new forwarding tables are available, they have been computed for a situation that no longer exists and may make the response to the failure worse, not better.

The rationale for automatic routing has always been that events are happening too fast for a human to be in the decision loop. It may be that events are happening too fast to have v6 in the loop.

not the problem. The Internet's focus on the transport and network layer has led to attempts to solve these problems in one of those two places. But, there is no such thing as a transport address. This is creating a "beads-on-a-string in disguise" model, not an operating system or distributed systems model. Consequently, efforts such as *Host Identifier Protocol* (HIP) (RFC 4423) and SHIM6 (Nordmark and Bagnulo, 2006) are simply more stopgaps that fail to address the whole problem and apply yet another band-aid to one aspect of the problem. As many in the Internet rightly realize, all of these myopic band-aids are creating a system that is more and more unwieldy.

Many prominent members of the Internet technical community have not expected wide deployment of IPv6. The biggest problem is that IPv6 offers very little to those who have to pay for its adoption. The removal of link-local (private) addresses provides one more reason not to adopt IPv6 in the enterprise, but to only use it externally. All new facilities, such as security, multicast, QoS-related developments, and so on, are designed to work equally well with IPv4 or IPv6. Thus, all statements in the recent trade press that IPv6 is necessary and has better QoS, security, and such are simply spin. The only new capability provided by IPv6 is a longer address, and that in and of itself may create more problems than it solves. In early 2003, figures were published that around 50% of the IPv4 address space had been assigned and less than 29% was actually being used (Huston, 2003). A cursory inspection shows that between 25-30 Class A address blocks could and should be re-claimed. This would seem to indicate (and is supported by recent government reports) that there is no rush to move to IPv6.

The only advantages to IPv6 are the bigger address space, the loss of isolation with no equivalent to private addresses, and the knowledge that you are a good network citizen—hardly the basis for a large capital expense to make the transition. This is not going to impress corporate budget committees. However, the possibility of IPv6 failing to be adopted has so alarmed certain factions that an immense PR campaign has been initiated to drum up interest in IPv6. (The possibility that IPv6 may fail for technical reasons does not seem to bother them.) An IPv6 forum was created and many trade journal articles written advocating advantages to IPv6 for security, QoS, and so on, which, in fact, are unrelated to IPv6. Trade journals go out of their way to put a positive spin on even the bad news. The European Union and the U.S. government have endorsed IPv6 in much the same way they endorsed OSI two decades earlier. IPv6 advocates point to this as proof of IPv6's pending success, just as they ridiculed the same statements by OSI advocates. Others see this as the kiss of death as it was for OSI. India, Japan, and China have embraced IPv6 mostly because they cannot get large IPv4 address blocks from IANA to support their huge populations. However, as we have seen, more than enough v4 address space exists. IPv6 may happen as much because the IETF has not been able to come up with anything that solves real problems, rather than on its own merits. This does not bode well.

But what contribution can we say that IPv6 has brought to our problem of trying to gain a deeper understanding of the nature of addressing? Unfortunately, not much. There is really nothing new here that has not been done before. As we have seen, IPv6 is simply a more cumbersome form of IPv4.

However, it does provide further confirmation of the social behavior of standards committees. (OSI provides earlier confirmation.) Another example of how a vocal conservative (dare I say ill-informed) faction can slow progress, and the lengths that a minority with greater technical understanding must go to find a way to bend the position of conservatives to get some sort of solution that solves real problems,[13] not to mention that this direction benefits the vendors: Not only does the iterative increase in complexity keep a steady stream of new products to buy, but it also serves as a barrier to entry to new competitors and keeps customers tied to the vendor because their personnel can't understand the interactions of all the incremental improvements. CLNP had been only a slight improvement over IPv4. But it had been a bigger step than IPv6 represents and had been at least a move in the right direction. All of this contributes to the feeling that the concepts had run out of steam. After about 1975, there was very little new or innovative thinking going on. The only significant development one can point to is the development of link-state routing algorithms, which primarily was done in OSI, which stimulated similar efforts in the IETF.

If there is anything to learn from the IPv6 experience, it probably has more to do with the dynamics (or lack thereof) of consensus. It was James Madison (1787) who was the first to realize the inherently conservative nature of such groups. And human nature hasn't changed in 200 years. In his case, it led to the creation of mechanisms to stabilize an otherwise unstable system. In this environment, the lack of understanding of this dynamic has merely undermined innovation in a fast-moving technology. OSI started out as a "revolutionary"

[13] The similarity to controversies in other areas of science are striking.

group intending to promulgate the packet network connectionless model. But the European tendency toward centralism and fear of the PTTs expanded the participation in the effort to include the opposition that saw X.25 as the answer to all network layer issues. This irresolvable conflict so severely split the OSI attempt that it ultimately failed. We have already discussed how the minority had to contort that architecture to achieve a semblance of a reasonable addressing architecture for the network layer, only to have it botched by the implementers. The fundamental lesson here is that the old paradigm can never be invited to collaborate with the new paradigm.

In the IETF, the conservatives have been a similar drag on innovation and good engineering. But here the stakes are much higher. OSI basically never had wide deployment. Businesses the world over now depend on the Internet. The IETF is now more concerned that the Internet architecture should not deviate from the old ways—that the architecture of 1972 has been given to it on stone tablets handed down from on high. When in reality, it was done by a group of engineers who were struggling to understand a new field and just to get something that worked. The conservatives now read deep meaning into what were expedient hacks, the authors of which knew they were hacks and knew they would need to be replaced "when there was time." The keepers of the flame are protecting an unfinished demo, rather than finishing it in the spirit in which it was started.

So if we have learned anything from IPv6, it is that all committees behave pretty much the same and will try to avoid deviating from the status quo. The problem within the IETF is compounded by the "demokratic" organization, rather than a "representative" or republican organization. It has been well understood for 250 years that democracies don't work and are susceptible to just this kind of long-term behavior. But, mechanisms can be created in a republican form of organization that will work; this was Madison's innovative discovery in system design. Representative forms have the potential to adopt new results not yet fully understood by the larger group. However, it remains that the only time a committee will do something innovative is when the majority perceives it as unimportant. Not exactly a result that is terribly helpful or encouraging.

"Upper-Layer" or Application Addressing in OSI

From our previous discussion, we would expect addressing for upper layers to involve some unique problems. According to Shoch and Saltzer, applications are supposed to have names, whereas lower-layer protocols have addresses. We must consider the problem of naming applications and relating that to addressing. Let's consider how the Internet and OSI dealt with upper-layer addressing.

As noted earlier, the early ARPANET had its hands full demonstrating a resource-sharing network and created "well-known sockets" as a stopgap so that it could demonstrate the usefulness of the network. The need for a directory was well understood at the time, but there were other priorities. Because there were no new applications in the Internet for another 20 years, there was no reason to change. (And by this time, there was a new generation of engineers who now argued that well-known sockets were a gift from the gods, divine insight, not a kludge that should be fixed.)

The first impetus for change was not required by applications and all the resource sharing that had been expected, but by the proliferation of hosts. Since the beginning, each host had maintained its own table of hostnames and their corresponding network address (NCP or IP). Only a few hosts might be added per month, and not all hosts found it necessary to keep a complete table. However, as the rate of new hosts increased in the late 1970s, this fairly informal approach was no longer practical. The result was the development of DNS or the *Domain Name Server* (RFC 881, 882). DNS defined a database structure not only for mapping hostnames to addresses, but also for distributing the database to servers around the network. Later, DNS was used to also distribute URLs for HTTP.

URLs are not the same as well-known sockets. A well-known socket identifies a special transport layer port identifier that has a particular application protocol bound to it. There is an implicit assumption that there is only one instance of this protocol per host. A connection to a well-known socket will create a distinct connection or flow to the requestor. A URL identifies an application (i.e., a particular Web page that uses that protocol [HTTP]), and an arbitrary instance of that application is created. We must be careful when talking about URLs. What they were defined for and how they are used in combination with other conventions make them several things at once. This is fine and perhaps even advantageous for human use, but for architecture we need to understand the different objects being named and their relation.

As discussed in Chapter 4, OSI created problems for itself by getting the upper layers upside down. Applications sat on top of two layers (session and presentation) that had addressing (a general property of a layer). These layers were constrained to not allow mapping between connection and connectionless and to have no multiplexing. Consequently, mappings between two layers were required to be one-to-one. There was no need for addressing in these two layers. Another indication that these were not layers.

We saw that for the lower layers it was not a good idea to create addresses for a layer by concatenating it with the address of the layer below because it formed a pathname. For the upper layers of OSI, there was no multiplexing and, hence, no multiple paths. However, this would create very long addresses with considerable redundant information as one moved up from the network layer. For example, because a transport address would be NetAddr.suffixT, the session address to be carried in protocol would be TrptAddr.suffixS or NetAddr.suffixT.suffixS, and the presentation address would beNetAddr.suffixT.suffixS.suffixP. This creates a lot of unnecessary overhead in the PDUs. To avoid this, an (N)-address for the transport, session, and presentation was defined as a tuple consisting of a network address and the appropriate number of (N)-selectors. Thus, a presentation address was defined as follows:

(Network address, T-sel, S-sel, P-sel)

The PCI in each layer above the network layer only carried the selector. If an implementer was smart, the P-selector and S-selector were null. Consequently, the only addressing above the network layer was that transport protocol had to carry a T-sel of 16 bits.[14]

Because there was no addressing in the session and presentation layers, the interesting aspect of OSI addressing for the upper layers was the addressing architecture of the application layer. In Chapter 4, we saw how the distinction between the application process and application entity came about. Now we have to consider how the naming of them works.

Table 5-1 *Summary of OSI Application Naming*

Item (Identified by AE)	APT	APII	AEQ	AEII
Appl Process	+			
Appl Process Invocation	+	+		
Appl Entity	+		+	
Appl Entity Invocation	+	+	+	+

	Scope
APT = Application-Process-Title	Application layer
APII = Application-Process-Invocation-Identifier	Application process
AEQ = Application Entity Qualifier	Application process
AEII = Application Entity Invocation Identifier	(API, AE)

14 Somebody in a NIST workshop thought the maximum size of T-sel should be 40 octets. Now I believe in large addresses as much as anyone, but even I thought 2^{320} application connections in a single host at the same time was a little excessive! Another indication that separating designers and implementers is not a good idea.

To recap from Chapter 4, OSI distinguished the "application entity" (AE), which was within the OSI architecture and consisted of the application protocols. Databases, file systems, the rest of the application, and so on were outside of OSI. (This was somewhat political so that the OSI committee did not tread on the turf of other committees.) Thus, the protocols an application used were part of the network architecture but everything else was outside. This is exactly the distinction we noted in the Web page example earlier. The application that constitutes the Web page and everything it needs is outside the communication architecture, but the HTTP protocol (and any other application protocols it uses, such as FTP or a remote query protocol) is within the architecture.

Thus, the Web application is an AP, and HTTP is the AE; and in this case, the AP may have several AE instances, for the simultaneous HTTP connections. Each must be distinctly identifiable. An application could have multiple protocols associated with it. For example, a hotel reservation application might use HTTP to talk to the customer and a remote database protocol to make the reservation. Similarly, an application could have multiple instances of each protocol and different dialogs with different customers. So, there could be application entity instances. Of course, the designer might choose to instantiate a different process for each customer so that there are multiple instances of the application process but single instances of the AEs. Clearly, there could be applications where there were instances of both processes and entities. The AEs were the only part of the application process inside the OSI architecture.

We can see in hindsight that the early Internet applications were special cases and hence not good examples to generalize from. Not only were the protocol and the application essentially synonymous, but there was only one per system. This is where our operating system experience was not sufficiently rich and we needed insight from the users' world. Our first real-life example of this application structure was the Web.

Once this structure was recognized, the application naming architecture was straightforward. OSI defined naming that allowed AEs and their instances as well as APs and their instances to be addressed. Addressing in the lower layers had never bothered to address to the level of instances. There is no reason to connect to a specific transport or TCP connection. They are all the same. However, for applications this is not the case. Recovery and other mechanisms would need to be able to establish or reestablish communication to an existing invocation of a protocol (AE) or to the invocation of an application (AP) using it. This leads to the addressing structure shown in Table 5-1.

Before one balks too much at the apparent complexity of this naming structure, a couple of things need to be observed. First of all, most applications don't need most of this. But the ones that do, really need it. Second, the complex forms, when they are needed, are generally needed by processes, not humans. Third, it is not at all clear that any "naming" at this level should be intended for human use. In the days of command language–driven operating systems, application names and filenames were intended for human use. However, today this is much less clear. What we used to think of as "user-friendly" (e.g., www.cnn.com) is not considered so today.

In the early days of networking, it was believed that applications had names and hosts had addresses. But this was an artifact of the implementation (and sloppy thinking); it turns out that when one carefully analyzes the problem, the host never appears (another surprise). Processes on a host appear but not the host. As we saw, this concept was brought over from operating systems. As understanding improved, it became clear that the important property of addresses is that they are used to "locate" objects; that is, that they be topologically significant. But application "names" are not just labels. They are used to locate applications and are just as topological as addresses, although admittedly in a very different topology. The structure of application names is used just as much to *locate* the application in the space of applications as the structure of network addresses locates in the space of network nodes. (This might be close to what some call the "semantic Web.")

In most incarnations, this leads to proposals for a hierarchical name structure. However, more recently this has been challenged by a more brute-force approach relying on searching. The role in the 1980s and early 1990s that many saw a system like the X.500 Directory or URNs playing now seems to be supplanted by Google, Yahoo!, and so on. Even within our systems, we have relied on search rather than richer structures. It remains to be seen whether searching can scale or whether other mnemonic or more structured methods may be necessary. But the question remains, that some form of common name that humans can exchange among themselves for use with computers is needed. How do we make this user friendly when a *Macintosh* might be a red apple, a computer, a stereo amplifier, or a raincoat. Or do the humans have to learn how to be friendly with the names computers use? For our purposes, we are less concerned with how these interface to people and are more concerned with what needs to be named, the properties of the names, and their relation.

URI, URL, URN, and So On: Upper-Layer Addressing in the Internet

As noted in Chapter 4, there has been very little work in the Internet space on upper-layer architecture and consequently also on naming and addressing issues in the upper layers. Everything derives from the host-naming convention. Originally, the convention was simply <hostname>, as the number grew it became necessary to move to a multilevel structure:

<local domain-id>.†<host/site name>.<TL-domain>

This structure was generally used to name hosts within a site or subnet. In fact, if one looks closely at the URL syntax, one finds that it is a pathname through the stack. Precisely what we saw earlier we wanted to avoid.

The work on the Universal Resource Name moves to a more sophisticated level of directory functions but does not really give us any insight in to the architecture of application naming requirements. The URN work in essence defines a syntax for names of resources and its interaction with a database defining various mechanisms to search the database and return a record. What the record contains is left to the designer of the specific URN. The URN syntax defines the top level of a hierarchy and conventions of notation and then allows specific communities to define the specific syntax to fit their application.

This would lead us to look at the applications to perhaps find some insights into application architecture naming issues. Unfortunately, most applications have not reached a level of complexity that requires more structure than a simple pathname hierarchy.

Conclusions

As we have seen, addressing is a subtle problem, fraught with traps. Early in the development of networks, simple solutions that ignored the major issues were more than sufficient. But as networks grew, the addressing problems should have been investigated. With the exception of two seminal pieces of work, however, they were largely ignored. However, the very long incubation period as an R&D effort (more than 20 years) removed from the pressures of business and used primarily by experts allowed people's ideas to calcify. The effect of Moore's law, increasing power, and decreasing cost of equipment made it possible to ignore the problems until long past the point when they should have been resolved (making it very painful to fix them). Early on (in CYCLADES), it was understood that it was necessary to make a transition from physical to logical address at least once (and even better if more than once). This was supported by Shoch's and then Saltzer's view that applications, nodes, points of attachment, and routes were the fundamental elements of addressing that had to be distinguished. From this and early distributed computing experiments, we recognized that application names were location independent, whereas nodes were location dependent but not route dependent. Although nodes seemed to be synonymous to hosts most of the time, there were counter-examples that showed that this was another false intuition. Oddly enough, it turns out that the only requirement to name a host or a system occurs in network management. Naming hosts is irrelevant to communications.[15]

This was later refined as topologically dependent. It was still unclear how these properties should manifest themselves. Given how network topologies can change, it was often unclear how this could be accomplished without being too tightly coupled to the physical topology of the network. It even took some time to realize (and is still unlearned by many protocol designers) that the limited scope of some layers meant that not all addresses had to be globally unambiguous. It is a sorry state of affairs that there has been almost no progress in understanding addressing in the past 25 years.

It should also be pointed out that although one can point to these facts in the literature, they were generally not understood by 99% of the engineers involved in networking. Very few, if any, textbooks in the field teach general principles of networking; they generally only teach current practice.[16] By the 1990s, current practice was the only general theory most engineers knew. There had always been a tendency to concentrate on research directly applicable to the Internet, instead of understanding the field of networking as a whole. Such general research had always been a fraction of the total, as one would expect, but by the mid-1980s it had pretty much died out entirely. Has the field begun to more resemble an artisan guild than an engineering discipline? This was compounded by no new applications to drive new requirements. The three applications that existed were all special cases that did not expose the full structure. This was not helped by the fact that addressing is a hard problem. Saltzer gave us the basics of what needed to be named, but finding a meaningful interpretation to location dependence was a major stumbling block. Both IP and CLNP made attempts, but both were rooted in the past. Now with all of this background, we are ready to consider how to assemble larger architectural structures.

---

15 Yes, it is often the case that *node* and *host* are synonymous, and it may be convenient in informal conversation. But as Shoch's quote we referenced earlier indicates, professionally we must be precise in our use of terms, or we will get ourselves in trouble.

16 Every so often, on one of the IETF discussions lists, some young engineer or professor gets a glimmer of these general principles that often contradict what we currently do. Instead of being told that "yes, those are the principles but we did not know that at the time," he is quickly led back to the party line using varying degrees of coercion.

Index

Page numbers followed by *n* designate footnotes.

A

D

F

G

H

M

N

O

Q-R

T