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, 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 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 Protocol," 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 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, "Diving 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 they 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 with each pattern that emerged 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 more than 20 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 the fundamental insight, 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 to 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, "Naming and Addressing," 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 will we need and can consider the problem of assembling them into a system. 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. (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, Multihoming, Multicast, and Mobility," we look at how multihoming, mobility, and multicast/anycast 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.