Designs for an Internet

David D. Clark

Draft Version 3.0 of Jan 1, 2017

Status This version of the book is a pre-release intended to get feedback and comments from members of the network research community and other interested readers. Readers should assume that the book will receive substantial revision.

The chapters on economics, management and meeting the needs of society are preliminary, and comments are particularly solicited on these chapters.

Suggestions as to how to improve the descriptions of the various architectures I have discussed are particularly solicited, as are suggestions about additional citations to relevant material. For those with a technical background, note that the appendix contains a further review of relevant architectural work, beyond what is in Chapter 5.

I am particularly interesting in learning which parts of the book non-technical readers find hard to follow.

Revision history

Version 1.1 first pre-release May 9 2016.

Version 2.0 October 2016. Addition of appendix with further review of related work. Addition of a ”Chapter zero”, which provides an introduction to the Internet for non-technical readers. Substantial revision to several chapters.

Version 3.0 Jan 2017 Addition of discussion of Active Nets Still missing–discussion of SDN in management chapter.

A note on the cover

The picture I used on the cover is not strictly "architecture". It is a picture of the Memorial to the Murdered Jews of Europe, in Berlin, which I photographed in 2006. When I saw it, I was quite struck by the apparently endless variation on a common theme represented by the blocks. According to Wikipedia (see https://en.wikipedia.org/wiki/Memorial_to_the_Murdered_Jews_of_Europe), the artist said that the stelae are designed to produce an uneasy, confusing atmosphere. Perhaps, but I actually found wandering among all the small variations on an endless theme somewhat tranquil.

As I thought about all the alternative proposals I have seen for an Internet architecture, the picture grew on me.

It is possible that I should not entangle respect for the victims of the Holocaust with my musings on architectural diversity, but I found this installation wonderful, and all the little variations, all requesting our attention, seem quite symbolic of what we in the research community sometimes find ourselves doing.

Contents

Preface

This is a book about how to design an Internet. I say *an* Internet rather than *the* Internet because the book is not just about the Internet we have today, but as well about possible alternative conceptions of an Internet–what we might instead have designed back then, or might contemplate in the future. I take the word "Internet" to describe a general purpose, global interconnection of networks designed to facilitate communication among computers, and among people using those computers. The book concerns itself with the implications of globality, the implications of generality, and the other requirements that such a network would have to meet. But it does not take the current Internet as a given–it tries to learn from the Internet of today, and from alternative proposals for what an Internet might be, to draw some general conclusions and design principles about networks.

Those design principles I will call *architecture*. So this book is as well about architecture. There are lots of little design decisions that shape today's internet, but they could have been made differently and we would still have an Internet. It is the basic design decisions that define the skeleton of the design, on which subsequent, more specific decisions are made. I am concerned with the question of what the essence of the design is–what defines a successful skeleton, if you will.

This is a very personal book. It is opinionated, and I write without hesitation in the first person. It is a book-length position paper–a point of view about design. I have drawn on lots of insights from lots of people, but those people might well not agree with all of my conclusions. In this respect, the book reflects a reality of engineering–while engineers hope that they can base their work on sound, scientific principles, engineering is as well a design discipline, and design is in part a matter of taste. So what this book talks about is in part matters of taste, and if I can convince the reader about matters of taste, so much the better.

The inspiration for this book arose out of the NSF-sponsored Future Internet Architecture program, and its predecessors, the Future Internet Design program (FIND) and the Network Science and Engineering (NetSE) program. These programs challenged the network research community to envision what an Internet of 15 years from now might be, without being constrained by the Internet of today. I have been involved in this program for its duration, and I have had the chance to listen to several excellent groups of investigators discuss different approaches to designing an Internet. These conversations have been very helpful in bringing into focus what is really fundamental about an Internet. There have also been similar projects in other parts of the world, in particular Europe, which have contributed to my understanding. Just as one may perhaps come to understand one's language better by the study of a foreign language, one may come to understand *the* Internet better by the study of alternative approaches. Chapter 5 provides an introduction to these various projects.

An Internet is deeply embedded in the larger social, political and cultural context. Assuming that we aspire to build a future global internetwork, we must accept that different parts of the world will present very different contexts into which the technology must fit. So this is not a book just about technology. Indeed, technology is not center stage at all. Much of the book centers on the larger issues, the economic, social and political considerations that will determine the success or failure of a system like this that is so woven into the larger world. If this book provides some insights into how the technical community can reason about this larger set of design constraints, it will have been a success from my point of view.

Because the Compute Science community has co-opted the word "architecture" I begin the book with a

discussion of that concept. The book then…@@

Chapter 0

A primer on the Internet

If you come to this book with a technical background, you can probably skip this chapter. But in order to understand some of the material in this book, it is important that the reader have a basic understanding of how the Internet works. Most of the important concepts are introduced and explained when they are first introduced in the text, but this chapter provides a brief overview of the structure and function of the Internet, to provide a better foundation for the less technical reader.

0.1 Introduction

The Internet is a communications facility designed to connect computers together so that they can exchange digital information. Data carried across the Internet is organized into *packets*, which are independent units of data, complete with delivery instructions attached in the first part or *header* of the packet. The Internet provides a basic communication service that conveys these packets from a source computer to one or more destination computers . Additionally, the Internet provides supporting services such as the naming of the attached computers (the Domain Name Service or DNS). A number of high-level services or applications have been designed and implemented making use of this basic communication service, including the World Wide Web, Internet e-mail, the Internet "newsgroups", distribution of audio and video information, games, file transfer and "login" between distant computers. The design of the Internet is such that new high-level services can be designed and deployed in the future.

An application program on a computer that needs to deliver data to another computer invokes software that breaks that data into some number of packets and transmits these packets one at a time into the Internet. A number of application support services are defined that can assist applications by performing this function, most commonly the Transmission Control Protocol (TCP).

Internet users tend to use the term *Internet* to describe the totality of the experience, including the applications. But to a network engineer, the Internet is a packet transport service provided by one set of entities (Internet Service Providers or ISPs), and the applications run *on top of* that service, and are in general provided by a different set of entities (Application Providers). It is important to distinguish the Internet as a packet forwarding mechanism from the applications that run on top of that mechanism. It is also important to distinguish the Internet from the technology that supports it. The Internet is not a specific communication "technology", such as fiber optics or radio. It makes use of these and other technologies in order to get packets from place to place. The Internet was intentionally designed to allow as many technologies as possible to be exploited as part of the Internet, and to incorporate new technologies as they are invented.

The heart of the Internet itself is a very simple service model that allows a wide range of applications to exploit the basic packet carriage service of the Internet over a wide range of communication technologies. The designer of each application does not need to know the details of each technology, but only the specification of this basic communication service. The designer of each technology must support this service, but need not know about the individual applications. In this way, the details of the applications and the details of the technologies are

separated, so that each can evolve independently.

0.2 The basic communication model of the Internet

The basic service model for packet delivery is very simple. It contains two parts: the addresses that identify the computers attached to the the Internet and the delivery contract that describes what the network will do when it is given a packet to deliver. To implement addressing, the Internet has numbers that identify end points, somewhat similar to the telephone system, and the sender identifies the destination of a packet using these numbers. (But see the discussion of multicast below.) The delivery contract specifies what the sender can expect when it hands a packet over to the Internet for delivery. The original delivery contract of the Internet is that the Internet will do its best to deliver to the destination all the data given to it for carriage, but makes no commitment as to data rate, delivery delay, or loss rates. This service is called the best effort delivery model.

This very indefinite and non-committal delivery contract has both benefit and risk. The benefit is that almost any underlying technology can implement it. The risk of this vague contract is that some applications might not be successfully built on top of it. However, the demonstrated range of applications that have been deployed over the Internet suggests that the service is adequate in practice. As is discussed below, this simple service model does have limits, and research in the early 1990's had the goal of extending this service model to deal with new objectives such as real time delivery of audio and video.

Protocols

The word protocol is used to refer to the conventions and standards that define how each layer of the Internet operates.[1] Different bodies have created the protocols that specify the different parts of the Internet. The Internet layer discussed above is specified in documents that define the format of the packet headers, the control messages that can be sent, and so on. This set of definitions is called the Internet Protocol, or IP. These standards were initially specified in 1981 by the Internet Engineering Task Force (and its predecessor, the Network Working Group), an open working group that has grown up along with the Internet.[2] This group created the Internet Protocol and the other protocols that define the basic communication service of the Internet. This group also developed the protocols for early applications such as e-mail. Some protocols are defined by academic and industry consortia; for example the protocols that specify the World Wide Web are mostly developed by the World Wide Web Consortium (the W3C) hosted at the Computer Science and Artificial Intelligence Laboratory at MIT. These protocols, once developed, are then used as the basis of products that are sold to the various entities involved in the deployment and operation of the Internet.

0.3 The role of the router

The Internet is made up of a series of communication links connected by relay points called routers. There can be many sorts of communications links–the only requirement is that they can transport a packet from one router to another. Each router, as it receives a packet, examines the delivery information in the header of the packet and based on the destination address, determines where to send the packet next. This processing and forwarding of packets is the basic communication service of the Internet.

Typically, a router is a computer, either general purpose or specially designed for this role, with software and hardware that implements the forwarding functions. A high-performance router used in the interior of the

[1] *The word* protocol *may well have been chosen based on its use in diplomacy, where it describes formal and proscribed modes of interaction. However, the etymology of the word tells another story. The word comes from the Greek prōtokollon, and means "first page," from prōtos "first" + kolla "glue." The Greeks glued the table of contents of a scroll onto the beginning after the scroll had been written out; we put the header (not literally with glue) onto the front of each packet. I have not yet determined whether the researchers who first picked the term protocol for our standards had a good Classical education.*

[2] *The various standards of the IETF (and other related publications) are published in a series of documents called "Requests for Comment", which captures the idea that the IETF is open to suggestions and change. The RFCs can be found at https://tools.ietf.org/html/. The specification of IP is RFC 791.*

Internet may be a very expensive and sophisticated device, while a router used in a small business or at other points near the edge of the network may be a small unit costing less than a hundred dollars. Whatever the price and performance, all routers perform the same basic communication function of forwarding packets.

A reasonable analogy to this process is the handling of mail by the post office or a commercial package handler. Every piece of mail carries a destination address, and proceeds in a series of hops using different technologies (e.g. truck, plane, or letter carrier). The addressing information is on the outside of the envelope, and the contents (the data that the application wishes to send) is inside the envelope. The post office (with some limitations) is indifferent as to what is in the envelope. At each hop, the address is examined to determine the next hop to take. To emphasize this analogy, the delivery process in the Internet is sometimes called *datagram* delivery. [3]. Similarly, routers forward packets based on the header, not the application-level data that is inside the packet. Routers do not provide application-specific services. Because routers in the Internet do not take into account what application generated the packets they are forwarding, I will use the term *application-agnostic* to describe the router function. When some abnormal situation arises, a router along a path from a sender to a receiver may send a packet with a control message back to the original sender of the packet. Again, by analogy, if the address on a letter is flawed in some way, the post office may write: "Undeliverable: return to sender" on the envelope and send it back so the sender can attempt to remedy the situation.

Another important aspect of packet forwarding is that the routers do not keep track of the packets they forward. (Nor does the post office log the letters they forward, unless the sender pays a substantial fee for tracking.) Systems like this are sometimes called *memoryless*, or *stateless*. Computer Science uses the word *state* to capture the idea that a device with memory can be in different states based on stored information–information that can reflect what has happened in the past. Given a similar input (e.g., a packet to forward) a system that can be in different states might treat the same input differently. We talk about the stored information that defines the state of a system as its *state variables* and from time to time throughout this book I will talk about the state variables of a component (or say that a component is stateless or has no state variables), as part of explaining why it can or cannot implement a certain function.

0.4 Application support services in the end-node

The delivery contract of the Internet is very simple: the best effort service tries its best to deliver all the packets given it by the sender, but makes no guarantees–it may lose packets, duplicate them, deliver them out of order, and delay them unpredictably. Many applications find this service difficult to deal with, because there are so many kinds of errors to detect and correct. For this reason, the Internet protocols include a transport service that runs "on top of" the basic Internet service and tries to detect and correct all these errors, and give the application a much simpler model of network behavior. This transport service is called Transmission Control Protocol, or TCP. TCP offers a service to the application in which a series of bytes given to the TCP at the sending end-node emerge from the TCP software at the receiving end-node in order, exactly once. This service is called a virtual circuit service. The TCP software takes the responsibility of breaking the series of bytes into packets, numbering the packets to detect losses and reorderings, retransmitting lost packets until they eventually get through, and delivering the bytes in order to the application. This service is often much easier to utilize than the basic Internet communication service.

Division of responsibility

The router, which implements the relay point between two communication links, has a very different role than the computer or end-node attached to the Internet. In the Internet design, the router is only concerned with forwarding the packets along the next hop towards the destination. The end-node has a more complex set of responsibilities related to providing service to the application. In particular, the end-node provides the additional services such as TCP that make it easier for the application (such as the World Wide Web) to make use of the

[3] *Datagram sounds more like a telegram analogy than a postal service analogy. I think the postal analogy is better, but I did not pick the term.*

basic packet transfer service of the Internet.

TCP is implemented in the end-nodes, but not in the packet forwarding software of the routers. The routers look only at the Internet information, such as the destination address, when forwarding packets. Only the end-nodes look at the TCP information in the packets. This is consistent with the design goals of the Internet, and is a very important example of layered design. TCP provides a very simple service that most high-level applications find easy to use, but some applications such as real time streaming are not well-matched to the service model of TCP. If TCP were implemented in the routers, it would be much harder for the high-level service to bypass it and use some other sort of transport service. So the design principle of the Internet has been to push functions out of the network to the extent possible, and implement them only in the end-nodes and higher-level service nodes. By doing so, the high-level services can be modified by adding new software to the service nodes. This is another means by which the Internet can evolve rapidly. Installing new services can be done without any need to modify the routers.

0.5 Routing and forwarding

There are also functions that are implemented in the routers, but not the end-nodes. The router must make a decision as to how to forward each packet as it arrives. In order to do this, it must have forwarding tables that specify, for each destination address (or cluster of addresses) what the preferred path is onward towards that point. In addition to forwarding packets, routers compute the best routes to all the addresses in the network, in order to construct this table. This requires that the routers send messages to other routers describing what links in the Internet are currently in operation, and what routers these links connect. This results in a collective decision-making, the routing computation, to select the best overall routes. Routers perform this task in the background, at the same time that they forward packets. If a low-level communications link fails or a new one is installed, this routing computation will construct new routes as appropriate. This adaptation is part of making the Internet robust in the face of failure. There are a number of routing protocols that have been defined and deployed to implement the routing computation.[4]

End-nodes do not participate in the routing computation. They know only the identity of a router or routers closest to them; when they send a packet they just pass it to this first router, which then decides where to send it next, and so on. This division of responsibility makes it possible to replace the routing protocol (which has happened several times in the life of the Internet) without having to change the software in the end-nodes, an almost impossible task if it had to be done in a coordinated way for all the millions of end-nodes on the Internet.

Regions of the Internet

The Internet is made up of routers, but every router is part of some region, or *Autonomous System*, or AS. Each AS is operated by some entity, which might be a commercial Internet Service Provider (ISP), corporation, university, and so on. There are about 45,000 ASes across the globe as I write this book. The original Internet used one global routing protocol, but as the Internet got bigger, it was clear to the designers that the Internet needed at least two tiers of routing: schemes that operated inside each AS and provided the routes among those routers, and a routing protocol that connects all the ASes together. The protocol that currently defines this function is called the Border Gateway Protocol, or BGP. BGP is used to tell each AS where all the others are, and what destination addresses are inside each AS. To over-simplify (as I will often do in this book), BGP works as follows. Imagine an AS at the edge of the Internet, for example the AS that represents the portion of the Internet that is inside MIT. In order for MIT to get access to the rest of the Internet, it makes a business arrangement with some Internet Service Provider that offers this service, and once that business arrangement is in place, the border router at MIT (hence the name of the protocol) sends a BGP message to that provider ISP, saying "here is where MIT is connected". That ISP (call it A) now knows where MIT is connected, and it tells its neighbor networks (for example, ISP B) "If you want to

[4] *I mentioned above, when introducing the concept of* state *and* state variables*, that routers do not have state variables to keep track of the different packets they forward. However, the forwarding table is clearly an example of state in a router.*

get to MIT, just send the packets to A". ISP B now tells its neighbors (for example, ISP C): "If you want to get to MIT, send your packets to B, which will send them to A, which will send them to MIT". These messages (called *path vectors*) propagate across the Internet until in principle every AS knows where to send a packet to get it to any other AS on the Internet.

BGP will come up several times in this book.

0.6 The Domain Name System

The routers use the destination addresses in the header of packets to select the forwarding path, but these addresses are not very easy for users to remember or use, so the Internet has a system to provide names for end points that are more "user friendly". A machine that receives mail must have an address, but it also will have a name like "mail.mit.edu", where the suffix "edu" is reserved for educational institutions. The system that keeps track of these names and translates them into numbers on demand is called the Domain Name System or the DNS–"domain" because the names are typically associated with regions such as MIT. A region that has a name like "mit.edu" in the DNS might also be an Autonomous System (MIT is AS 3), but that relationship is not necessary. The DNS is built out of a large number of servers connected across the Internet that provide name to address translation for different regions. I will not be discussing the design of the DNS in any detail in this book, but it is important to understand that it exists and the role that it plays.

0.7 The design of applications

The Internet itself, as an entity built of links and routers, is concerned with the delivery of packets. Applications such as the World Wide Web exist at a higher level. Users may think of applications as a part of the Internet (users tend to say they are using the Internet when they use Facebook, explore the Web or send email), but technically applications run "on top of" the basic communication service of the Internet, most often on top of TCP.

The World Wide Web as an example

The World Wide Web was conceived as a set of protocols that allow a Web Client (often called a browser) to connect to a Web server.

A Web server (a particular kind of end-node attached to the Internet) stores Web pages, and makes them available for retrieval on request. The pages have names, called URLs (Universal Resource Locators). These names are advertised so that potential readers can discover them; they also form the basis of cross-references or " links" from one Web page to another; when a user positions the mouse over a link and "clicks" it, the matching URL is used to move to the associated page. The first part of a URL is actually a DNS name, and the DNS system I described above is used to translate that name into the address of the intended Web server. A message is then sent to that Web server asking for the page. [[[Say more; say less? Need this sort of stuff?]]]

There are actually a number of protocols that together define the basic function of the Web. There is a protocol called HyperText Transfer Protocol, or HTTP, that provides the rules and format for messages requesting a Web page. However, the HTTP standard does not define how pages are represented. The most common representation of a Web page is HTML, which stands for HyperText Markup Language

This description of the Web illustrates the layered nature of the Internet design. The transfer of a Web page involves actions at several different layers at the same time: [[[delete this?]]]

IP: At the Internet level, packets are received and transmitted when a Web page is requested. At this layer, the only relevant factors are the Internet addresses in the packets.

TCP: The TCP software takes a unit of data (a file, a request for a Web page or whatever) and moves it across the Internet as a series of packets. It does not examine these bytes to determine their "meaning"; in fact, the bytes might be encrypted.

HTTP: HTTP makes use of TCP to move requests and replies. In contrast to TCP, it looks at the contents of requests, understands the syntax and "meaning" of these requests. Based on the request, HTTP then transfers the

Web page in question. However, it does not know the format or ”meaning” of the page itself. The page could be a traditional Web page, an image, music, and so on.

HTML: All browsers include software that understands HTML, so that arriving Web pages can be interpreted and displayed on the screen. HTML is not the only page format. Images, for example, are encoded in a number of different ways, indicated by the name of the standard: GIF, JPEG etc. These are alternative formats that can be found inside a Web page. Any format is acceptable, so long as the browser includes software that knows how to interpret it.

E-mail

The description of the World Wide Web given above was of transfers between two computers, the browser and the server. Not all applications work this way. An important and early example of an alternative application design is electronic mail, or e-mail. Since many users are not connected full time, if mail was transferred in one step from origin to destination, the transfer could only be successful during those occasional periods when both parties just happened to be connected at the same time. To avoid this problem, almost all mail recipients make use of a server to receive their mail and hold it until they connect. They then collect all their mail from their server. The concept is that the mail server is always attached and available, so anyone can send mail to it at any time, and the receiver can retrieve mail from it at any time. This eliminates the necessity for the sender and the receiver to be attached at the same time. Most mail is actually transferred in three steps, from sending end-node to the mail server serving that sender, then to the mail server serving the recipient, and then to the final end-node.

Different applications are different

As these two examples illustrate, different high-level services have different designs. The pattern of mail distribution does not resemble the pattern of Web page retrieval. The delivery of email depends on servers distributed across the Internet, but these servers, in contrast to the application-agnostic routers, are application-aware. They are designed to function as part of a part of a specific application, and provide services in support of that application.

There are many applications on the Internet today: email, the Web, computer games, Voice over IP (internet telephone service or VoIP), video streaming and so on. The list is almost endless (and in fact there is no list–one does not need to “register” to invent a new application; one “just does it”). Part of the power of the Internet packet transport service is that it is an open platform that makes it possible for anyone with the right skills and initiative to design and program a new application.

0.8 Onward

With this background, it is time to proceed to the real focus of the book, which is a study of the basic design principles of the Internet (what I call its “architecture”), how those principles relate to the requirements that the Internet should meet (requirements both technical and social), and how alternative conceptions of an Internet might better address these requirements, or perhaps focus on different requirements that the designers consider more important.

Chapter **1**

Introduction

1.1 What is "architecture"

This is a book about architecture. So to get off on the right foot, it is important to understand what is meant by that word. It is perhaps overused, and used in a variety of contexts; without having a shared understanding between writer and reader there is a risk of failing to communicate. So what does the word mean?

Architecture is a process, an outcome and a discipline. As a process, it involves putting components and design elements together to make an entity that serves a purpose. As an outcome, it describes a set of entities that are defined by their form. The architectural form we know as "gothic cathedral" is characterized by a set of recognized design elements and approaches–the purpose may have been "place of worship"', but "gothic cathedral" implies a lot more. And finally, as a discipline, architecture is what architects are trained to do. The field of computer science borrowed the term from the discipline that designs physical things like building and cities, where there is a well-understood process of training and accreditation.

All three of these faces of architecture apply both to "real architecture" and to computer science.

As a process: There are two important parts to the definition: *putting components together* and *for a purpose.*

- Putting components together: this is what computer scientists are doing when they consider issues such as modularity, interfaces, dependency, layering, abstraction and component reuse. These are design patterns that we are trained to consider as we contemplate one or another design challenge.

- For a purpose: The process of design must be shaped by the intended purpose of the artifact: a hospital is not a prison, and a low-power processor is not a super-computer. As a part of architecture, the designers must address what the system cannot do (or do well) as well as what it is intended to do. In computer science, there is a peril in system design so well-known that it has a name: *second system syndrome*, the tendency, after having built a first system that perhaps does a few things well, to propose a replacement that tries to do everything.

As an outcome: In the practice of designing buildings, the design normally results in one copy of the result. There are exceptions, such as tract houses, where one design is constructed many times, but there is only one copy of most buildings. The term "architecture", when describing an outcome, normally implies a class of design, typified by its most salient features (e.g., flying buttresses). The term is applied to this abstraction, even though the architect has had to specify the building down to a very fine level of detail before the construction team takes over.

When computer science co-opted the term architecture, they slightly redefined it. With respect to the Internet, there have been many different networks built based on the same design: the public global network we call "the Internet", private networks belonging to enterprises, militaries, and the like, and special use networks such as

financial networks. In this context the word "architecture" only describes part of what is built, and much of the design process for a given instantiation occurs at a later point, perhaps specified by a different group.

As a discipline: "Real" architects–those who design building–go to school to learn their trade. Looking over the fence at what they do is instructive. Architecture (as opposed to structural engineering) is not a design discipline built on an underlying base of science and engineering principles. Architects do not normally concern themselves with issues such as strength of materials; they leave that to others. Of course, technical considerations may need to enter the design process early, as the architect deals with such issues as energy efficiency or earthquake resistance, but architects are primarily trained in the process of design. They do not study engineering, but buildings. They learn by case study–they look at lots of buildings and how (or not) they are fit for purpose. Do they meet the needs of the user? Are they considered visually attractive? How were the design trade-offs handled? And so on.

In computer science, we tend to hope that we can base our designs on strong engineering foundations, theories that give us limits and preferred design options, and so on, but (at least in the past) most of the business of system architecture has more resembled that of the building architect: learning from previous designs, asking what worked well and what did not, asking if the design was fit for purpose, and so on. We train computer scientists in both theory and practice, but we tend to deprecate the study of prior designs as "not science" or "not based on fundamentals". This book is unapologetically a study of design, not a book centered on a discipline with quantifiable foundations, like queueing theory or optimization. I am personally excited by attempts to make architecture more "rigorous", but we should not deprecate what we do today with phrases like "seat of the pants" design. [[[Confirm I did that.]]]Ours is a design discipline, just as is building architecture, and we should strive to excel at it, not dismiss it.

So if the "architecture" of the Internet is not the complete specification, but just a part of that specification, what is included in the architecture? We can say what is *not* included–we can look at all the different examples of networks based on Internet technology, or different regions of the global Internet, and note all the ways in which they differ. We see differences in performance, degree of resilience, tolerance of mobility, attention to security and so on. So design decisions at this level build on the core architecture, but are not *specified by* the core architecture. So what should we see as being in that core architecture?

Issues on which we must all agree for the system to function. For example, the Internet architecture is based on the use of packets, and the assumption that the packet header will the same everywhere. (A different design might allow different formats in different regions, in which case the architecture might choose to describe what sort of architectural support is provided for the necessary conversion.)

When we first designed the Internet, we thought that the design depended on having a single, global address space. It is now clear that this assumption was not necessary–there need not be global agreement on a uniform meaning of addresses (think about Network Address Translation). It is interesting to note that once we realized that we could build a network out of regions with different address spaces, there was no rush to extent the architecture to provide any support or guidance as to how disjoint address spaces are interconnected. How "NAT boxes" maintain enough state to map addresses is taken as a local matter. Of course, many view this state of affairs as deplorable, since it prevents certain sorts of applications from being deployed easily, but the "Internet architects", whoever they are, have not responded with a set of globally agreed conventions by which NAT state can be managed to facilitate the support of a broader suite of applications.

There are a few other points where global agreement is necessary. The Internet is made up of regions operated by different entities, called Autonomous Systems, or ASes. Information about how to route packets across multiple regions is exchanged among these ASes, using a protocol called the Boarder Gateway Protocol, or BGP. To some extent, all the regions must agree on the use of BGP, or at least the meaning of the packets exchanged across the inter-AS interface. Even if there were a region of the Internet that did not use BGP for interconnection, it is probably unavoidable to agree on the existence and meaning of Autonomous System numbers. And within the

global address space of the core of the Internet, it is necessary to agree on the meaning of certain address classes, such as multicast. It is worth noting that both multicast addresses and Autonomous System numbers were not conceptualized as part of the Internet's original design, but were designed later. In some sense, they have earned the right to be considered part of the core architecture exactly because a critical mass have agreed to depend on them. Things that the original designers thought were mandatory, such as a global address space, have turned out not to be mandatory, and other things that they were not contemplating have crept in and acquired the status of "that on which we must all agree".

Issues on which it is convenient to agree. There is no requirement that applications use the DNS, but since essentially all applications are designed based on the assumption that the DNS exists, it has become essentially mandatory as a part of the Internet.

The basic modularity of the system. For example, the specification of the Internet Protocol (IP) defines two sorts of module interfaces. It defines layer interfaces, for example the *service interface* on top of which higher level services are built, and it defines (implicitly and partially) *domain interfaces*: the interface among the different regions of the Internet. The service interface is the *best effort* packet-level delivery model of the Internet: a packet handed in to the Internet at one interface with a valid destination IP address in the packet will be forwarded to the interface defined by that IP address to the extent the network can do so at the moment. This service definition defers issues such as reliability onto higher layers. The other modularity interface in the Internet architecture is much less well-defined, and in fact is hardly visible in the early architectural specification–this is the modularity that corresponds to the different Internet Service Providers that make up the Internet, the Autonomous Systems I mentioned above. The emergence of Border Gateway Protocol (BGP) as a convention to hook Autonomous Systems together (which only occurred in the 1990's, as part of the transition of the Internet to a commercial undertaking) might seem to have the status today of "that on which we must all agree", but in fact that agreement is probably more a convenience than a necessity–a region of the Internet could deploy and use a different protocol, so long as it complied with a few more basic conventions–the role of AS numbers and routing on IP prefix blocks.

Functional dependencies One aspect of architecture is to make clear the *functional dependencies* of the design. I will use the Internet to illustrate what this means. The basic operation of the Internet is very simple. Routers compute routing tables in the background so that they know routes to all parts of the Internet. When they receive a packet, they look up the best route and send the packet on. While there is a lot of stuff going on inside the Internet, at its core that is really "all" that the Internet does. So for the Internet to provide service, the relevant routers must be up and providing service. The proper functioning of the Internet depends of necessity on the proper functioning of the routers. But what else is necessary for the Internet to provide service? In fact, the early designers of the Internet tried to be very careful to limit the number of services or elements that had to be operating in order for packets to flow. An early design goal was stated as follows: "If there are two computers hooked to a network, and each one knows the address of the other, they should be able to communicate. Nothing else should be needed".[1] This design preference could be expressed as the goal of minimal functional dependencies. Other proposals for the design of an Internet, which I will explore later in the book, have many more functional dependencies–they depend on more services to be up and running for basic communication to succeed. They are trading more functionality for (perhaps) less resilience when things go wrong. I will return to this later in the book.

[1] *In1987, Leslie Lamport, a well-known and very thoughtful computer scientist, sent an email in which he offered the following observation: "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable." That is a condition the designers of the Internet were trying to avoid.* http://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt.

Aspects of the system that are viewed as long-lasting. In a system like the Internet, we know that much will change. Indeed, the ability to change, and to upgrade and replace aspects of the system, are a key to successful longevity. (See chapter 6 for an extended discussion of these issues.) But to the extent that there are aspects that seem like durable invariants, specifying them as part of the design may provide stable points around which the rest of the system can evolve.

1.2 The role of interfaces

Interfaces are the specification of how modules are interconnected to make up the overall system. Interfaces become fixed points in the architecture–points that are hard to change precisely because many modules depend on them. Kirschner and Gerhart [Kirschner and Gerhart, 1998] develop the idea of interfaces as "constraints that deconstrain": points of fixed functionality that separate modules so that the modules can evolve independently, rather than being intertwined. Their work is in the context of evolutionary biology, but seems to apply to man-made systems, whether the designers are clever enough to get the interfaces right from the beginning, or whether the interfaces in this case also "evolve" to reflect points what stability is beneficial, and evolution elsewhere is also beneficial.[2] One could argue that the original Internet architecture posited that certain semantics of addresses were fixed points–the single global address space–and over time the system has evolved away from that constraint. But the syntax of the header–that addresses are 32 bits long–is proving very hard to change, since so many actors depend on it. IPv6 has been "trying to happen" for a painful number of years now.

Layering

Layering is a particular kind of modularity, in which there is an asymmetry of dependence. A system is layered, or more specifically two modules have a layered relationship, if the function of one module (the lower layer) does not depend on the correct function of the higher-layer module. Operating systems display a layered structure: the system itself should not be harmed or disrupted if an application running on the system crashes. Similarly, networks like the Internet are conceived as being layered–the basic packet forwarding service should not be affected by the applications running on top of it.

The idea of asymmetry of dependency may helps with the overall conception of a system, but is often not quite accurate in practice. One issue is performance–different applications can interact because they compete for resources, and in networking we see extreme examples of this in what are called Distributed Denial of Service attacks, in which a malicious actor tries to send enough traffic that a host on the network or a region of the network itself is precluded from making progress. One response to this would be to say that the design of a layer, if it is truly a "layer" with no dependence on what the modules above it do, must include mechanisms to protect itself from malicious applications and to isolate the different applications. The very simple service model of the Internet, of course, has no such protections in its architecture. In Chapter 5 I discuss a number of architectural approaches for mitigation of DDoS.

1.3 Summary–Thinking about architecture

I have sketched a basic conception of what I will mean by the word "architecture". In my view (and as I warned in the preface, this book is a personal point of view) a key principle is *architectural minimality*. In the computer science context, the architecture of a system should not try to specify every aspect of the system. This conception of architecture seems perhaps at variance with the architecture of buildings. When the architect of a building hands off the plans to the builder, the specification is complete down to the small details–not just the shape and structure but where the power outlets are. But it is not clear that all of these decisions should be classified as "architecture". As I said above, one of the distinctions between the architecture of a building and the architecture of an artifact like the Internet is that there are lots of networks built out using the same Internet technology, not

[2] *John Doyle and his co-authors [?] have developed and defended this conception of architecture. Dolye has described constraints as " hour-glasses" for interfaces that divide layers and "bow-ties" for interfaces that connect peer modules, like different ISPs.*

just one. There are obvious benefits if it is possible to use Internet technology in different contexts: commercial products are cheaper and likely to be more mature, the relevant software is found in almost all computer systems and so on. However, these networks may not have exactly the same requirements–they may have different requirements for security, resilience, and so on. So the power of architecture is not that it defines exactly what the network should do (as building plans specify exactly how the building is built) but that it allows these requirements to be met, but perhaps in different ways in different contexts.

I will argue, to paraphrase Einstein, that architecture should be as minimal as possible, but no less. One might argue that the most fundamental aspect of the architecture of the Internet as I characterize it is its preference for minimality. Given that point of view, the scope of what we take as the *architecture* of a network system should include only those aspect that fit within the framework I have laid out here, given the requirements that architecture sets out to address.

The next step in understanding how to define the architecture of an Internet is to return to the first point of the chapter, that architecture is the putting together of components for a purpose. We must ask: what is the purpose of an Internet. That is the topic of the next chapter.

Chapter 2

Requirements

2.1 Fitness for purpose–What is a network for?

In the previous chapter, I talked abstractly about "architecture", and about the architecture of an internet, assuming some common understanding of what it is that an internet actually does. But if we are to be both concrete and precise, we need to start with a specification of what such a system is expected to do. In this chapter I review a number of possible design requirements for the Internet (or for *an* Internet), which will set the stage for several of the following chapters.

The first requirement for an Internet is that it provide a useful service. The service model of the original Internet, while perhaps never carefully written down, is pretty simple. The Internet was expected to deliver a packet (of a certain maximum size) as best it could from any source to a destination specified by an IP address. This specification tolerated failure of the delivery, and indeed it was a rather explicit decision not to include in the specification any bound on the rate of failure. If the network is "doing its best", then so be it–the user can decide if this service is better than nothing. The "meaning" of the IP address is not a part of the specification–it is just a field used as an input to the forwarding algorithm in the routers. The limitations on our ability to design highly scalable forwarding algorithms imposes "soft" constraints on the use of IP addresses–they have to be allocated in ways that they can be aggregated into block that the routing protocols can process, as opposed to having the routing and forwarding mechanisms deal with each address separately.[1] But there is no outright prohibition against having the routing protocols deal with a single address as a routed entity.

As I will detail in Chapter 3, there were good reasons for this rather weak specification of what the Internet was to do. Had the initial designers challenged themselves with a much more constraining specification that set limits on such things as loss rates, throughput, etc., it is possible that the network would never have been built successfully in the beginning. However, as I will discuss in Chapter 7, this weak specification, which among other things is totally silent on what the network should *not* do, opens the door to a number of malicious behaviors we see on the Internet today. In that chapter, I will explore in more depth whether it would be practical and desirable to start with a more restrictive specification that precludes classes of bad behavior. [[[Confirm I did that.]]]

Should the network do more?

Part of the appeal of thinking about a "new" Internet is the challenge of devising new services that would make the network more useful–make it easier to design applications, or make it possible to serve a broader class of applications, or for the network to function in a wider range of circumstances.

Adding more complex functions to the network might make it easier to deploy new classes of applications, but obviously adds complexity to the network itself. There is thus a tradeoff between what "the network" should do, and what a service layer on top of the network could do for a class of applications. This tradeoff is a recurring one in system design–the early history of operating systems was marked by functions initially being implemented

[1] *[Caesar et al., 2006] provides an assessment of the practicality of relaxing this constraint. The conclusions are not optimistic.*

by applications and then migrating into the kernel as their value was proven.[2] So several threads of network research today are exploring the addition of new functionality to the network.

Over time, new services have been added to the specification of the Internet. An IP address originally referred to a single destination, associated with a network interface on a specific machine. However, IP addresses can now be used in different ways. The concept of *anycast* is that multiple destinations can have the same IP address, and the routing protocols will direct the packet to the "closest" one. The concept of *multicast* is that multiple destinations can have the same IP address and the routing protocols will direct copies of the packet to *all* of them. Multicast is distinctive in that it requires a different set of routing and forwarding algorithms to be implemented in the system–whether to use the multicast or the unicast forwarding algorithm is determined by the prefix of the addresses. Another possible service objective would be that the network could tailor the parameters of delivery to the requirements of the application. This concept, which today is commonly called Quality of Service (QoS), requires more complex scheduling in the forwarding mechanisms and/or more complex routing mechanisms. Without debating here the merits of either multicast or QoS forwarding, we can note their implications on overall network design–if there are alternative treatments that different packets receive, there has to be some signal, either in the packet or stored as state in the router, that indicates which treatment each packet gets. With respect to QoS, the original design of the Internet contemplated such a scheme and used the Type of Service field in the header to trigger different services. With respect to multicast, which was not initially contemplated, a set of distinct addresses had to be set aside to trigger the desired behavior.

Implicit in the specification of the original Internet was that a router could only forward a packet or drop it. The idea that it might store the packet was hardly even discussed, since memory was scarce in the 1970's, and the unstated assumption was that the goal of the Internet was rapid delivery–an important early application was remote login. Storing packets in the network if they cannot be forwarded both adds complexity to the network (should the specification define how long packets should be stored, and under what circumstances) and as well complexity to the behavior that the application sees. However, allowing storage as a part of the network behavior might make it possible to design a new class of applications directly on top of the network, as opposed to requiring the deployment of storage servers on the network as a part of the application.[3]

One of the more innovative ideas now being explored with respect to a future Internet is that the basic service objective should be rethought–rejecting the idea that the correct service is delivering a packet to a destination specified by an address. One alternative is that the packet should be delivered to a more abstract conception of a destination, a *service*. In some respects, this proposal is a generalization of the anycast concept I mentioned above; for this to be practical the routing and forwarding schemes must be prepared to deal with a very large number of such addresses (with the current current *anycast* mechanism, such addresses are exceptions and are few in number). Another alternative idea is that the goal of the network is to deliver to the requester a packet of contents, without the requestor knowing anything about the location of the contents. The equivalent of the "network address" in this conception is the name of the content that is to be returned. This concept, called Information Centric Networking (ICN), has profound implications both for the network and the application. The network must be able to forward packets based on the name of the desired content, rather then the address of the destination. Applications may or may not find this a natural model of network behavior, but since it is a very different model, application designers must learn to work with it.

I return to this design question in Chapter 4: how can we reason about the range of services that the network might usefully offer to the higher layers that exploit the network. I will discuss providing generality in the packet header (the *syntax* of the network, perhaps) to trigger a range of behaviors (the *semantics* of the network. In chapter 5 I return to the design of ICNs.

[2] *The operating system on the IBM 1620, in the mid-1960's, did not include support for a file system, but left disk management to the application. The system would continue to run if the disk was powered down during operation.*

[3] *The Delay/Disruption Tolerant Network community represents one example of this approach, as does the Mobility First FIA project. See Chapter 5.*

2.2 Generality

One of the reasons that the Internet has been successful is that it was designed with the goal of generality. In fact, there are two important aspects of generality that are represented in the Internet: generality with respect to the applications that run over it, and generality with respect to the sorts of network technology out of which it can be built.

Generality of purpose

The Internet was conceived to be a"general purpose" network. It is suited to email, watching a video, playing a computer game, looking at Web pages, and a wide variety of other applications. This generality seems a natural way to structure a network that hooks computers together: computers are general-purpose devices and since the Internet hooks computers together, it too was intended to be general. When the Internet was initially being designed, however, this preference for generality was not uniformly accepted. Indeed, this idea was quite alien to the communications engineers of the time, who worked for the telephone companies. They asked what to them was an obvious question: how can you design something if you don't know what it is for? The telephone system was designed for a known purpose: to carry telephone calls. The requirements implied by that purpose drove all the design decisions of the telephone system, and the engineers from the world of telephone systems were confounded by the idea of designing a system without knowing what its application would be. One can understand the early history of the Internet by noting that it was designed by people who came from a computing background, not a classical networking (telephony) background. Most computers are designed without knowing what they are for, and this mind-set defined the Internet's design.

But this generality has its price. The service it delivers is almost certainly not optimal for any particular application. Design for optimal performance does not end up in the same place as design for generality. (There is thus perhaps a tension between design preferences such as generality, optimality, minimality and the like, to which I will return from time to time.) And it may take more effort to design each application than if the network were tailored to that application. Over the decades of the Internet's evolution, there have been a succession of dominant applications. In the early years of the Internet, it was equated to email, and to ask someone if they were "on the Internet" was to ask if they had an email address. Email is a very undemanding application to support, and if the Internet had drifted too far toward supporting just that application (as was happening to some degree), the Web might not have been able to emerge. But the Web succeeded, and the emergence of this new application reminded people of the value of generality. Now this cycle repeats, and the emergence of streaming audio and video tested the generality of an Internet that had drifted toward a presumption that now the Web, and not email, was "the application". Now "the application" that drives the constant re-engineering of the Internet is streaming, high quality video. And it is easy once again to assume that "now we know what the Internet is for", and optimize it for streaming video. In my view, the community that designs the Internet should always be alert to protect the generality of the Internet, and allow for the future in the face of the present.

Generality of technology

The other dimension of generality that was critical to the Internet's success is that it was structured so that it could work over a wide range of communications technologies. The early Internet interconnected three communications technologies: the original ARPAnet, SATnet (the Wideband experimental multi-point Atlantic satellite network) and a spread spectrum packet radio network (PRnet). Because the goal was to operate over as broad as possible a selection of technologies, the architecture made minimal assumptions about what these technologies could do. Had the design targeted a known communications technology, it might have been possible to exploit the particular features of that technology (for example, some wireless systems are inherently broadcast), which might have led to a more efficient outcome. But the decision to architect an Internet that could operate over "anything" allowed new sorts of technology to be added as they emerged, for example local area networks (LANs). We see this tension between generality and optimization repeating today: a network of limited scope, for example a network internal to a car, may be based on a known network technology, which will allow more sorts of cross-layer optimization.

2.3 Longevity

One measure of the Internet's success is how long its design has remained viable. Presumably, any proposal for a system architecture has the aspiration of proving durable over time. One view is that a long-lived network must be evolvable; it must have the adaptability and flexibility to deal with changing requirements, while remaining architecturally coherent. The goal of evolution over time is closely linked to the goal of operating in different ways in different regions, in response to regional requirements such as security. On the other hand, a factor that can contribute to longevity is the stability of the system: the ability of the system to provide a platform that does not change in disruptive ways. I explore different theories of how to design a long-lived system in Chapter 6.

For an architecture like the Internet to survive over time, there are several subsidiary requirements:

Support for tomorrow's computing: The Internet arose as a technology to hook computers together, so as the shape of computing evolves, so should the Internet. In 10 years, the dominant form of computing will not be the PC, nor even the smart phone or tablet, but most probably the small, embedded processor acting as a sensor or actuator.[4] At the same time, high-end processing will continue to grow, with huge server farms, cloud computing and the like. Any future Internet must somehow take this wide spectrum of computation into account. One point of view is that this wide range of requirements for performance and for low-cost ubiquitous connectivity cannot be met by one approach to transport and interconnection, in which case we will see the emergence of more than one network architecture. We will see the single Internet architecture of today replaced by a range of alternatives at this level of the design, each targeted toward each of these domains and only interconnected at higher levels. On the contrary, it is possible that one set of standards will span this range of requirements just fine.

Utilize tomorrow's networking: At least two communication technologies will be basic to tomorrow's networks, wireless and optical. Wireless (and mobility) implies new sorts of routing (e.g., broadcast), the tolerance of intermittent connectivity, and dealing with losses. Advanced optical networks not only bring huge transmission capacity, they can offer rapid reconfiguration of the network connectivity graph, which again has large implications for routing and traffic engineering. One point of view about the Internet is that the emergence of wireless networks requires more cross-layer optimization to make effective use of wireless technology, and the architecture of a future Internet should not imply a single way of doing things. The challenge this raises is how these different ways should hook together, but the requirement for interoperation does not mean that an Internet has to be based on the same design everywhere. Interoperation can be achieved at different layers. Part of what an architecture must do is frame the proposed solution to this problem.

There is an interesting interplay between architecture and technology. In the early days of the Internet, the networks were assembled using communications technology that had been designed for different purposes (e.g., telephone circuits). One of the early goals of the Internet was to work on top of "anything", because that was seen as the only path to rapid, wide deployment. But as the Internet has matured and proven its success, network technology has evolved to provide efficient support for the Internet as defined. Over the long run, technology can be expected to follow the architecture, rather than the architecture having to bend itself to accept technology designed for other purposes. The tension between short-term deployment and long-term effectiveness is a design challenge for any architecture. As well, careful design of the architecture can either facilitate or hinder the emergence of useful sorts of technological heterogeneity.

Support tomorrow's applications: Today's Internet has proved versatile and flexible in supporting a range of applications. There is not some important application that is blocked from emerging because of the current Internet. None the less, applications of today and tomorrow present requirements that a future Internet should

[4] *As I write this book in 2016, the current buzzword for this future is the Internet of Things, or IoT. We will see if this term sticks.*

take into account. These include a range of security requirements, support for highly available applications, real-time services, new sorts of naming, and the like.

2.4 Security

The Internet of today is marked by a number of serious security issues, including weak defenses against attacks on hosts, attacks that attempt to disrupt communications, attacks on availability (Denial of Service or DoS attacks), and attacks on the proper operation of applications. Ideally, an Internet architecture would have a coherent security framework, which makes clear what role the network, the application, the end node, etc. each has in improving security. I explore the issue of Internet security, and the relationship between architecture and the resulting security properties, in Chapter 7.

2.5 Availability and resilience

These two goals are sometimes lumped into security, but I have listed them separately because of their importance, and because availability issues arise in the Internet of today independent of security attacks. Improving availability requires attention to security, to good network management and preventing errors by operators, and to good fault detection and recovery. Again, what is needed is a *theory* for availability. While the Internet of today deals with specific sorts of faults and component failures (lost packets, links and routers that fail), it does not have an architectural view of availability. I return to this topic in Chapter 8.

2.6 Management

Management has been a weak aspect of the current Internet from the beginning, to a considerable extent because the shape and nature of the management problem was not clear in the early days of the design. Among other things, it was not clear what aspects of network operation would (or should) involve human operators, and which would preferably be automated if possible. As I will argue in chapter 10, there may not be a single coherent issue that is "management", just as there is no single issue that defines "security". The key, both to security and management, is to break the problem into its more fundamental parts, and address them without necessary reference to "basket words" like security and management.

2.7 Economic viability

A fundamental fact of the current Internet is that the physical assets out of which it built, the links, routers, wireless towers, etc., are expensive. These assets, often collectively called facilities, come into existence only if some actor chooses to invest in them. Chapter 9 explores the relationship between system design (and core design methods such as system modularity) and industry structure. To argue that a system is viable as a real-world offering, a designer must describe the set of entities (e.g., commercial firms) that are implied by the architecture, and make an argument that each will have the incentives to play the role defined for them by the architecture. Using the current Internet as an example, there is a tension between a core value of the current Internet–its open platform quality, and the desire of investors to capture the benefits of their investment. In Section 4.3 I introduce the term *tussle* to describe the situation where the different actors in an Internet ecosystem do not have aligned incentives or motivations, and I call the working out of this tension between an open architecture and the desire to monetize infrastructure the *fundamental tussle*. Any proposal for a network design must of necessity take a stance in this space. For example, one tilts the fundamental tussle toward vertical integration and a more closed architecture if additional functions are bundled with (or to any extent replace) the basic forwarding function.

2.8 Meeting needs of society

A network design will not succeed in the real world unless there is a purpose for which users find it useful. The Internet is not just a technical artifact connecting computers, but a social artifact connecting people, deeply

embedded in society. To a large extent, users do not directly observe the core architecture of the system–they partake of the system using the applications that are designed on top of it. So as I noted above, one measure of a successful network is that is it suited to support a wide range of applications, both today's and tomorrow's. On the other hand, the core design may impose conventions and provide features that cut across applications, and as the system in question supports more functions, the core design will become more visible to the users–consider the differences visible to the user between using an Android or IOS smart phone. The Internet of today provides a very simple service, and one could argue that many variants of an Internet would be equally successful. But the core design will influence the outcome of some very important social considerations, such as the balance between surveillance and accountability on one hand and anonymous action and privacy on the other. Users want a network where they can do what they please–they have choice in their applications and activities–but criminals have no ability to undertake their activities effectively. They want a network that is reliable and trustworthy, but they do not want either the private sector or governments watching what they (and thus the criminals as well) are doing. Chapter 11 explores some of these socially important tradeoffs, and considers whether, and to what extent, the core architecture defines the balance, or whether the balance is determined by the applications built on top of the network itself.

2.9 Moving beyond requirements

The topics listed in the previous sections are posed at a very high level. They are not actionable as posed; they are desiderata, an *aide-memoire*, as we contemplate design. It is a big jump from any of these to the design of specific mechanism, and that is a big issue. We would like the design process to be based on principles and theory, but there are no well-honed design methods to aid in the process of moving from these requirements to mechanism and architecture.

Several things can happen as we move from high-level requirements to specific architecture and mechanism. One is that in the attempt to reduce an idea such as "security" to practice we discover that lurking inside that requirement are sub-goals that are actually in tension with each other, or with other requirements. Design is not optimization along a single dimension, but a balancing of different priorities. Some of these may be quantifiable (e.g., a performance requirement), but most will end up as qualitative objectives, which makes the balancing harder. There is a tendency in the Computer Science community to prefer to optimize factors that can be quantified, such as performance, but if an Internet is going to be relevant in the real world, we must face the messy challenge of evaluating alternative approaches to security or economic viability.

A further problem is that as we move beyond requirements for a system like the Internet, the resulting design problem may grow too large for one team to contemplate holistically, so the design process may itself need to be modularized. The choice of that design modularity may end up be reflected in the modularity of the system itself. Another way of understanding this reality is that the fundamental modularity of the system had better be specified before the design process is modularized, so that the modularity dictates the design process, and not the other way around.

Requirements and architecture

Several of the subsequent chapters are dedicated to exploring in more depth the requirements I have discussed here and refining them so that they become actionable. But there is a high-level question which cuts across all of these requirements, which is how they relate to architecture. Should we look to the architecture of a network to see how these requirements are fulfilled? The definition that I offered of architecture in chapter 1 defined architecture in a minimal way: it was those things on which we have to agree, things on which it is highly convenient to agree, the basic modularity of the system, or aspects of the system that are expected to be long-lasting. Given this preference for architectural minimality, it will turn out that the architecture itself, as I have defined it, does not directly specify a system that meets these requirements. Rather, what it does is provide a framework within which it is possible to design a system that meets these requirements. In order to make this way of thinking more concrete, in Chapter 3 I use the existing Internet as an example, and go back to an earlier attempt to list the

requirements that the Internet was intended to meet, and how its architecture addressed these requirements.

Chapter 3

The architecture of the Internet–A historical perspective

The introduction to architecture in Chapter 1 was a bit abstract. I am going to look at what I consider the architecture of the current Internet as a more concrete example. In 1988 I wrote a paper titled "The Design Philosophy of the DARPA Internet Protocols", which tried to capture the requirements the Internet was being designed to meet, and the basic design decisions that had been taken in meeting these requirements–what I might now call architecture, but then called "design philosophy". It is now over 25 years since that paper was published, and looking back at that paper is a way to get started with a less abstract, more concrete example of "network architecture".

What follows is that original paper, as first published in 1988, with extensive commentary from the perspective of 2015.

THE DESIGN PHILOSOPHY OF THE DARPA INTERNET PROTOCOLS

David D. Clark
Massachusetts Institute of Technology
Laboratory for Computer Science (now CSAIL)
Cambridge, Ma. 02139

This paper was originally published in 1988 in ACM SigComm. Original work was supported in part by the Defense Advanced ResearchProjects Agency (DARPA) under Contract No. NOOOIJ-83-K-0125. Revised, with extensive commentary, 2015 . The original text has been reformatted, but is otherwise unchanged from the original except for a few spelling corrections.

Abstract

The Internet protocol suite, TCP/IP, was first proposed fifteen years ago. It was developed by the Defense Advanced Research Projects Agency (DARPA), and has been used widely in military and commercial systems. While there have been papers and specifications that describe how the protocols work, it is sometimes difficult to deduce from these why the protocol is as it is. For example, the Internet protocol is based on a connectionless or datagram mode of service. The motivation for this has been greatly misunderstood. This paper attempts to capture some of the early reasoning which shaped the Internet protocols.

Introduction

For the last 15 years [1], the Advanced Research Projects Agency of the U.S. Department of Defense has been developing a suite of protocols for packet switched networking. These protocols, which include the Internet Protocol (IP), and the Transmission Control Protocol (TCP), are now U.S. Department of Defense standards for internetworking, and are in wide use in the commercial networking environment. The ideas developed in this effort have also influenced other protocol suites, most importantly the connectionless configuration of the ISo protocols [2,3,4].

While specific information on the DOD protocols is fairly generally available [5,6,7], it is sometimes difficult to determine the motivation and reasoning which led to the design.

In fact, the design philosophy has evolved considerably from the first proposal to the current standards. For example, the idea of the datagram, or connectionless service, does not receive particular emphasis in the first paper, but has come to be the defining characteristic of the protocol. Another example is the layering of the architecture into the IP and TCP layers. This seems basic to the design, but was also not a part of the original proposal. These changes in the Internet design arose through the repeated pattern of implementation and testing that occurred before the standards were set.

The Internet architecture is still evolving. Sometimes a new extension challenges one of the design principles, but in any case an understanding of the history of the design provides a necessary context for current design extensions. The connectionless configuration of ISO protocols has also been colored by the history of the Internet suite, so an understanding of the Internet design philosophy may be helpful to those working with ISO.

This paper catalogs one view of the original objectives of the Internet architecture, and discusses the relation between these goals and the important features of the protocols.

This paper makes a distinction between the architecture of the Internet and a specific realization of a running network. Today, as discussed below, I would distinguish three ideas: [1]

1. *The core principles and basic design decisions of the architecture.*
2. *The second level of mechanism design that fleshes out the architecture and makes it into a complete implementation.*

3. *The set of decisions related to deployment (e.g. the degree of diversity in paths) that lead to an operational network.*

Fundamental Goal

The top level goal for the DARPA Internet Architecture was to develop an effective technique for multiplexed utilization of existing interconnected networks. Some elaboration is appropriate to make clear the meaning of that goal. The components of the Internet were networks, which were to be interconnected to provide some larger service. The original goal was to connect together the original ARPANET[8] with the ARPA packet radio network[9,10], in order to give users on the packet radio network access to the large service machines on the ARPANET. At the time it was assumed that there would be other sorts of networks to interconnect, although the local area network had not yet emerged.

This paragraph hints at but does not state clearly that the Internet builds on and extends the fundamental goal of the ARPANET, which was to provide useful interconnection among heterogeneous machines. Perhaps even by 1988 this point was so well-understood that it did not seem to require stating.

There is also an implicit assumption that the end-points of network connections were machines. This assumption seemed obvious at the time, but is now being questioned, with architectural proposals that "addresses" refer to services or information objects.

An alternative to interconnecting existing networks would have been to design a unified system which incorporated a variety of different transmission media, a multi-media network.

Perhaps the term "multi-media" was not well-defined in 1988. It now has a different meaning, of course.

While this might have permitted a higher degree of integration, and thus better performance, it was felt that it was necessary to incorporate the then existing network architectures if Internet was to be useful in a practical sense. Further, networks represent administrative boundaries of control, and it was an ambition of this project to come to grips with the problem of integrating a number of separately administrated entities into a common utility.

This last is actually a goal, and probably should have been listed as such, although it could be seen as an aspect of goal 4, below.

The technique selected for multiplexing was packet switching.

Effective multiplexing of expensive resources (e.g. links) is another high-level goal that is not in the explicit list but very important and well-understood at the time.

An alternative such as circuit switching could have been considered, but the applications being supported, such as remote login, were naturally served by the packet switching paradigm, and the networks which were to

[1] *I am indebted to John Wroclawski, both for the suggestion that led to this revision, and for the insight that there are three concepts to be distinguished, not two.*

be integrated together in this project were packet switching networks. So packet switching was accepted as a fundamental component of the Internet architecture. The final aspect of this fundamental goal was the assumption of the particular technique for interconnecting these networks. Since the technique of store and forward packet switching, as demonstrated in the previous DARPA project, the ARPANET, was well understood, the top level assumption was that networks would be interconnected by a layer of Internet packet switches, which were called gateways.

From these assumptions comes the fundamental structure of the Internet: a packet switched communications facility in which a number of distinguishable networks are connected together using packet communications processors called gateways which implement a store and forward packet forwarding algorithm.

In retrospect, this previous section could have been clearer. It discussed both goals and basic architectural responses to these goals without teasing these ideas apart. Gateways are not a goal, but a design response to a goal.

We could have taken a different approach to internetworking, for example providing interoperation at a higher level–perhaps at the transport protocol layer, or a higher service/naming layer. It would be an interesting exercise to look at such a proposal and evaluate it relative to these criteria.

Second Level Goals

The top level goal stated in the previous section contains the word ”effective,” without offering any definition of what an effective interconnection must achieve. The following list summarizes a more detailed set of goals which were established for the Internet architecture.

1. Internet communication must continue despite loss of networks or gateways.
2. The Internet must support multiple types of communications service.
3. The Internet architecture must accommodate a variety of networks.
4. The Internet architecture must permit distributed management of its resources.
5. The Internet architecture must be cost effective.
6. The Internet architecture must permit host attachment with a low level of effort.
7. The resources used in the Internet architecture must be accountable.

This set of goals might seem to be nothing more than a checklist of all the desirable network features. It is important to understand that these goals are in order of importance, and an entirely different network architecture would result if the order were changed. For example, since this network was designed to operate in a military context, which implied the possibility of a hostile environment, survivability was put as a first goal, and accountability as a last goal. During wartime, one is less concerned with detailed accounting of resources used than with mustering whatever resources are available and rapidly deploying them in an operational manner. While the architects of the Internet were mindful of accountability, the problem received very little attention during the early stages of the design, and is only now being considered. An architecture primarily for commercial deployment would clearly place these goals at the opposite end of the list.

Similarly, the goal that the architecture be cost effective is clearly on the list, but below certain other goals, such as distributed management, or support of a wide variety of networks. Other protocol suites, including some of the more popular commercial architectures, have been optimized to a particular kind of network, for example a long haul store and forward network built of medium speed telephone lines, and deliver a very cost effective solution in this context, in exchange for dealing somewhat poorly with other kinds of nets, such as local area nets.

The reader should consider carefully the above list of goals, and recognize that this is not a "motherhood" list, but a set of priorities which strongly colored the design decisions within the Internet architecture. The following sections discuss the relationship between this list and the features of the Internet.

At the beginning of the NSF Future Internet Design (FIND) project, around 2008, I proposed a list of requirements that a new architecture might take into account. Here, for comparison with the early list from the 1988 paper, is the one I posed in 2008:

2008

1. *Security*
2. *Availability and resilience*
3. *Economic viability*
4. *Better management*
5. *Meet society's needs*
6. *Longevity*
7. *Support for tomorrow's computing*
8. *Exploit tomorrow's networking*
9. *Support tomorrow's applications*
10. *Fit for purpose (it works?)*

The list from 1988 does not mention the word "security". The first 1988 requirement, that the network continue operation despite loss of networks or gateways, could be seen as a specific sub-case of security, but the text in the next section of the original paper (see below) does not even hint that the failures might be due to malicious actions. In retrospect, it is difficult to reconstruct what our mind-set was when this paper was written (which is in the years immediately prior to 1988). By the early 1990s, security was an important if unresolved objective. It seems somewhat odd that the word did not even come up in this paper. The modern list calls out availability and resilience as distinct from the general category of security, a distinction that was motivated by my sense that this set of goals in particular were important enough that they should not be buried inside the broader category. So there is some correspondence between goal 1 in the 1988 list and 2 in the 2008 list.

The 2008 list has economic viability as its third objective. As I noted above, the 1988 paper discussed "the problem of integrating a number of separately administrated entities into a common utility", which seems like a specific manifestation of the recognition that the net is built out of parts. But the focus on economic viability seems to have been poorly understood, if at all.

Survivability in the Face of Failure

The most important goal on the list is that the Internet should continue to supply communications service, even though networks and gateways are failing. In particular, this goal was interpreted to mean that if two entities are communicating over the Internet and some failure causes the Internet to be temporarily disrupted and reconfigured to reconstitute the service, then the entities communicating should be able to continue without having to reestablish or reset the high level state of their conversation. More concretely, at the service interface of the transport layer, this architecture provides no facility to communicate to the client of the transport service that the synchronization between the sender and the receiver may have been lost. It was an assumption in this architecture that synchronization would never be lost unless there was no physical path over which any sort of communication could be achieved. In other words, at the top of transport, there is only one failure, and it is total partition. The architecture was to mask completely any transient failure.

This last sentence seems, in retrospect, a bit unrealistic, or perhaps poorly put. The architecture does not mask transient failures at all. That is not the goal, and it seems like an unrealizable one. The rest of the paragraph makes the actual point–if transient failures do occur, the application may be disrupted for the duration of the failure, but once the network has been reconstituted, the application (or, specifically, TCP) can take up where it left off. The rest of the section discusses the architectural approach to make this possible.

Again in retrospect, it would seem that an important sub-goal would be that transients are healed as quickly as possible, but I don't think there was any understanding then, and perhaps not now, of an architectural element that could facilitate that sub-goal. So it is just left to the second-level mechanisms.

To achieve this goal, the state information which describes the on-going conversation must be protected. Specific examples of state information would be the number of packets transmitted, the number of packets acknowledged, or the number of outstanding flow control permissions. If the lower layers of the architecture lose this information, they will not be able to tell if data has been lost, and the application layer will have to cope with the loss of synchrony. This architecture insisted that this disruption not occur, which meant that the state information must be protected from loss.

In some network architectures, this state is stored in the intermediate packet switching nodes of the network. In this case, to protect the information from loss, it must replicated. Because of the distributed nature of the replication, algorithms to ensure robust replication are themselves difficult to build, and few networks with distributed state information provide any sort of protection against failure. The alternative, which this architecture chose, is to take this information and gather it at the endpoint of the net, at the entity which is utilizing the service of the network. I call this approach to reliability "fate-sharing." The fate-sharing model suggests that it is acceptable to lose the state information associated with an entity if, at the same time, the entity itself is lost. Specifically, information about transport level synchronization is stored in the host which is attached to the net and using its communication service.

There are two important advantages to fate-sharing over replication. First, fate-sharing protects against any number of intermediate failures, whereas replication can only protect against a certain number (less than the number of replicated copies). Second, fate-sharing is much easier to engineer than replication.

There are two consequences to the fate-sharing approach to survivability. First, the intermediate packet switching nodes, or gateways, must not have any essential state information about on-going connections. Instead, they are stateless packet switches, a class of network design sometimes called a "datagram" network. Secondly, rather more trust is placed in the host machine than in an architecture where the network ensures the reliable delivery of data. If the host resident algorithms that ensure the sequencing and acknowledgment of data fail, applications on that machine are prevented from operation.

See the later discussion about where failures should be detected, and the role of trust.

Despite the fact that survivability is the first goal in the list, it is still second to the top level goal of interconnection of existing networks. A more survivable technology might have resulted from a single multimedia network design. For example, the Internet makes very weak assumptions about the ability of a network to report that it has failed. Internet is thus forced to detect network failures using Internet level mechanisms, with the potential for a slower and less specific error detection.

Types of Service

The second goal of the Internet architecture is that it should support, at the transport service level, a variety of types of service. Different types of service are distinguished by differing requirements for such things as speed, latency and reliability. The traditional type of service is the bidirectional reliable delivery of data. This service,

which is sometimes called a ”virtual circuit” service, is appropriate for such applications as remote login or file transfer. It was the first service provided in the Internet architecture, using the Transmission Control Protocol (TCP)[11]. It was early recognized that even this service had multiple variants, because remote login required a service with low delay in delivery, but low requirements for bandwidth, while file transfer was less concerned with delay, but very concerned with high throughput. TCP attempted to provide both these types of service.

The initial concept of TCP was that it could be general enough to support any needed type of service. However, as the full range of needed services became clear, it seemed too difficult to build support for all of them into one protocol.

The first example of a service outside the range of TCP was support for XNET[12], the cross-Internet debugger. TCP did not seem a suitable transport for XNET for several reasons. First, a debugger protocol should not be reliable. This conclusion may seem odd, but under conditions of stress or failure (which may be exactly when a debugger is needed) asking for reliable communications may prevent any communications at all. It is much better to build a service which can deal with whatever gets through, rather than insisting that every byte sent be delivered in order. Second, if TCP is general enough to deal with a broad range of clients, it is presumably somewhat complex. Again, it seemed wrong to expect support for this complexity in a debugging environment, which may lack even basic services expected in an operating system (e.g. support for timers.) So XNET was designed to run directly on top of the datagram service provided by Internet.

Another service which did not fit TCP was real time delivery of digitized speech, which was needed to support the teleconferencing aspect of command and control applications. In real time digital speech, the primary requirement is not a reliable service, but a service which minimizes and smooths the delay in the delivery of packets. The application layer is digitizing the analog speech, packetizing the resulting bits, and sending them out across the network on a regular basis. They must arrive at the receiver at a regular basis in order to be converted back to the analog signal. If packets do not arrive when expected, it is impossible to reassemble the signal in real time. A surprising observation about the control of variation in delay is that the most serious source of delay in networks is the mechanism to provide reliable delivery. A typical reliable transport protocol responds to a missing packet by requesting a retransmission and delaying the delivery of any subsequent packets until the lost packet has been retransmitted. It then delivers that packet and all remaining ones in sequence. The delay while this occurs can be many times the round trip delivery time of the net, and may completely disrupt the speech reassembly algorithm. In contrast, it is very easy to cope with an occasional missing packet. The missing speech can simply be replaced by a short period of silence, which in most cases does not impair the intelligibility of the speech to the listening human. If it does, high level error correction can occur, and the listener can ask the speaker to repeat the damaged phrase.

It was thus decided, fairly early in the development of the Internet architecture, that more than one transport service would be required, and the architecture must be prepared to tolerate simultaneously transports which wish to constrain reliability, delay, or bandwidth, at a minimum.

This goal caused TCP and IP, which originally had been a single protocol in the architecture, to be separated into two layers. TCP provided one particular type of service, the reliable sequenced data stream, while IP attempted to provide a basic building block out of which a variety of types of service could be built. This building block was the datagram, which had also been adopted to support survivability. Since the reliability associated with the delivery of a datagram was not guaranteed, but ”best effort,” it was possible to build out of the datagram a service that was reliable (by acknowledging and retransmitting at a higher level), or a service which traded reliability for the primitive delay characteristics of the underlying network substrate. The User Datagram Protocol (UDP)[13] was created to provide a application-level interface to the basic datagram service of Internet.

The architecture did not wish to assume that the underlying networks themselves support multiple types of services, because this would violate the goal of using existing networks. Instead, the hope was that multiple types of service could be constructed out of the basic datagram building block using algorithms within the host and the gateway.

I am quite surprised that I wrote those last two sentences. They are seriously and embarrassingly incorrect. RFC 791 [Postel, 1981] states:

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load).

...

Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET is given in "Service Mappings" [Jon Postel, 1981].

At the time this RFC was specified (around 1981) the group clearly had in mind that different sorts of network might have different tools for managing different service qualities, and the abstract ToS field was to be mapped to the network-specific service indicators by the gateway (what we now call the router).

For example, (although this is not done in most current implementations) it is possible to take datagrams which are associated with a controlled delay but unreliable service and place them at the head of the transmission queues unless their lifetime has expired, in which case they would be discarded; while packets associated with reliable streams would be placed at the back of the queues, but never discarded, no matter how long they had been in the net.

This section of the paper may reflect my own, long-standing preference for QoS in the network. However, the discussion is about a much more basic set of service types, and an architectural decision (splitting IP and TCP), which gives the end-node and application some control over the type of service. There is no mention in this paper of the ToS bits in the IP header, which were the first attempt to add a core feature that would facilitate any sort of QoS in the network. Discussions about QoS at the IETF did not start for another several years. But this section does suggest that the idea of queue management as a means to improve application behavior was understood even in the 1980s, and the ToS bits (or something like them) would be needed to drive that sort of scheduling. I think, looking back, that we really did not understand this set of issues, even in 1988.

It proved more difficult than first hoped to provide multiple types of service without explicit support from the underlying networks. The most serious problem was that networks designed with one particular type of service in mind were not flexible enough to support other services. Most commonly, a network will have been designed under the assumption that it should deliver reliable service, and will inject delays as a part of producing reliable service, whether or not this reliability is desired. The interface behavior defined by X.25, for example, implies reliable delivery, and there is no way to turn this feature off. Therefore, although Internet operates successfully over X.25 networks it cannot deliver the desired variability of type service in that context. Other networks which have an intrinsic datagram service are much more flexible in the type of service they will permit. but these networks are much less common, especially in the long-haul context.

Even though this paper comes about five years after the articulation of the end-to-end arguments, there is no mention of that paper or its concepts here. Perhaps this was due to the fact that this paper was a retrospective of the early thinking, which predated the emergence of end-to-end as a named concept. The concept is lurking in

much of what I wrote in this section, but perhaps in 1988 it was not yet clear that the end-to-end description as presented in the 1984 paper would survive as the accepted framing.

Varieties of Networks

It was very important for the success of the Internet architecture that it be able to incorporate and utilize a wide variety of network technologies, including military and commercial facilities. The Internet architecture has been very successful in meeting this goal: it is operated over a wide variety of networks, including long haul nets (the ARPANET itself and various X.25 networks), local area nets (Ethernet, ringnet, etc.), broadcast satellite nets (the DARPA Atlantic Satellite Network[14,15] operating at 64 kilobits per second and the DARPA Experimental Wideband Satellite Net[16] operating within the United States at 3 megabits per second), packet radio networks (the DARPA packet radio network, as well as an experimental British packet radio net and a network developed by amateur radio operators), a variety of serial links, ranging from 1200 bit per second asynchronous connections to TI links, and a variety of other ad hoc facilities, including intercomputer busses and the transport service provided by the higher layers of other network suites, such as IBM's HASP.

The Internet architecture achieves this flexibility by making a minimum set of assumptions about the function which the net will provide. The basic assumption is that network can transport a packet or datagram. The packet must be of reasonable size, perhaps 100 bytes minimum, and should be delivered with reasonable but not perfect reliability. The network must have some suitable form of addressing if it is more than a point to point link.

There are a number of services which are explicitly not assumed from the network. These include reliable or sequenced delivery, network level broadcast or multicast, priority ranking of transmitted packet, multiple types of service, and internal knowledge of failures, speeds, or delays. If these services had been required, then in order to accommodate a network within the Internet, it would be necessary either that the network support these services directly, or that the network interface software provide enhancements to simulate these services at the endpoint of the network. It was felt that this was an undesirable approach, because these services would have to be re-engineered and reimplemented for every single network and every single host interface to every network. By engineering these services at the transport, for example reliable delivery via TCP, the engineering must be done only once, and the implementation must be done only once for each host. After that, the implementation of interface software for a new network is usually very simple.

Other Goals

The three goals discussed so far were those which had the most profound impact on the design on the architecture. The remaining goals, because they were lower in importance, were perhaps less effectively met, or not so completely engineered. The goal of permitting distributed management of the Internet has certainly been met in certain respects. For example, not all of the gateways in the Internet are implemented and managed by the same agency. There are several different management centers within the deployed Internet, each operating a subset of the gateways, and there is a two-tiered routing algorithm which permits gateways from different administrations to exchange routing tables, even though they do not completely trust each other, and a variety of private routing algorithms used among the gateways in a single administration. Similarly, the various organizations which manage the gateways are not necessarily the same organizations that manage the networks to which the gateways are attached.

Even in 1988 we understood that the issue of trust (e.g. trust among gateways) as an important consideration.

On the other hand, some of the most significant problems with the Internet today relate to lack of sufficient tools for distributed management, especially in the area of routing. In the large Internet being currently operated, routing decisions need to be constrained by policies for resource usage. Today this can be done only in a very

limited way, which requires manual setting of tables. This is error-prone and at the same time not sufficiently powerful. The most important change in the Internet architecture over the next few years will probably be the development of a new generation of tools for management of resources in the context of multiple administrations.

It is interesting that the limitations of manual route configuration were understood in 1988, and we are not yet really beyond that stage. It is not clear even now whether our persistent lack of progress in this area is due to poor architectural choices, or just the intrinsic difficulty of the tasks. Certainly, in the 1970s and 1980s we did not know how to think about network management. We understood how to "manage a box", but we had no accepted view on systems-level management.

It is clear that in certain circumstances, the Internet architecture does not produce as cost effective a utilization of expensive communication resources as a more tailored architecture would. The headers of Internet packets are fairly long (a typical header is 40 bytes), and if short packets are sent, this overhead is apparent. The worse case, of course, is the single character remote login packets, which carry 40 bytes of header and one byte of data. Actually, it is very difficult for any protocol suite to claim that these sorts of interchanges are carried out with reasonable efficiency. At the other extreme, large packets for file transfer, with perhaps 1,000 bytes of data, have an overhead for the header of only four percent.

Another possible source of inefficiency is retransmission of lost packets. Since Internet does not insist that lost packets be recovered at the network level, it may be necessary to retransmit a lost packet from one end of the Internet to the other. This means that the retransmitted packet may cross several intervening nets a second time, whereas recovery at the network level would not generate this repeat traffic. This is an example of the tradeoff resulting from the decision, discussed above, of providing services from the end-points. The network interface code is much simpler, but the overall efficiency is potentially less. However, if the retransmission rate is low enough (for example, 1%) then the incremental cost is tolerable. As a rough rule of thumb for networks incorporated into the architecture, a loss of one packet in a hundred is quite reasonable, but a loss of one packet in ten suggests that reliability enhancements be added to the network if that type of service is required.

Again, this 1988 paper provides a nice "time capsule" as to what we were worrying about 25 years ago. Now we seem to have accepted the cost of packet headers, and we seem to have accepted the cost of end-to-end retransmission. The paper does not mention efficient link loading as an issue, nor the question of achieving good end-to-end performance.

The cost of attaching a host to the Internet is perhaps somewhat higher than in other architectures, because all of the mechanisms to provide the desired types of service, such as acknowledgments and retransmission strategies, must be implemented in the host rather than in the network. Initially, to programmers who were not familiar with protocol implementation, the effort of doing this seemed somewhat daunting. Implementers tried such things as moving the transport protocols to a front end processor, with the idea that the protocols would be implemented only once, rather than again for every type of host. However, this required the invention of a host to front end protocol which some thought almost as complicated to implement as the original transport protocol. As experience with protocols increases, the anxieties associated with implementing a protocol suite within the host seem to be decreasing, and implementations are now available for a wide variety of machines, including personal computers and other machines with very limited computing resources.

A related problem arising from the use of host-resident mechanisms is that poor implementation of the mechanism may hurt the network as well as the host. This problem was tolerated, because the initial experiments involved a limited number of host implementations which could be controlled. However, as the use of Internet has grown, this problem has occasionally surfaced in a serious way. In this respect, the goal of robustness, which

led to the method of fate-sharing, which led to host-resident algorithms, contributes to a loss of robustness if the host misbehaves.

This paragraph brings out a contradiction in the architectural principles that might have been made more clearly. The principle of minimal state in routers and movement of function to the end-points implies a need to trust those nodes to operate correctly, but the architecture does not have any approach to dealing with hosts that mis-behave. Without state in the network to validate what the hosts are doing, it seems that there are few ways to discipline a host. In 1988, the problem was anticipated but we clearly had no view as to how to think about it.

The last goal was accountability. In fact, accounting was discussed in the first paper by Cerf and Kahn as an important function of the protocols and gateways. However, at the present time, the Internet architecture contains few tools for accounting for packet flows. This problem is only now being studied, as the scope of the architecture is being expanded to include non-military consumers who are seriously concerned with understanding and monitoring the usage of the resources within the Internet.

Again, a deeper discussion here might have brought out some contradictions among goals: without any flow state in the network (or knowledge of what constitutes an "accountable entity") it seems hard to do accounting. The architecture does not preclude what we now call "middle-boxes", but the architecture also does not discuss the idea that there might be information in the packets to aid in accounting. I think in 1988 we just did not know how to think about this.

Architecture and Implementation

The previous discussion clearly suggests that one of the goals of the Internet architecture was to provide wide flexibility in the service offered. Different transport protocols could be used to provide different types of service, and different networks could be incorporated. Put another way, the architecture tried very hard not to constrain the range of service which the Internet could be engineered to provide. This, in turn, means that to understand the service which can be offered by a particular implementation of an Internet, one must look not to the architecture, but to the actual engineering of the software within the particular hosts and gateways, and to the particular networks which have been incorporated. I will use the term "realization" to describe a particular set of networks, gateways and hosts which have been connected together in the context of the Internet architecture. Realizations can differ by orders of magnitude in the service which they offer. Realizations have been built out of 1200 bit per second phone lines, and out of networks only with speeds greater than 1 megabit per second. Clearly, the throughput expectations which one can have of these realizations differ by orders of magnitude. Similarly, some Internet realizations have delays measured in tens of milliseconds, where others have delays measured in seconds. Certain applications such as real time speech work fundamentally differently across these two realizations. Some Internets have been engineered so that there is great redundancy in the gateways and paths. These Internets are survivable, because resources exist which can be reconfigured after failure. Other Internet realizations, to reduce cost, have single points of connectivity through the realization, so that a failure may partition the Internet into two halves.

As I said earlier, today I believe that there should be three distinctions:

1. *The core principles and basic design decisions of the architecture.*
2. *The second level of mechanism design that flesh out the architecture and make it into a complete implementation.*

3. *The set of decisions related to deployment (e.g. degree of redundancy in paths) that lead to an operational network.*

The word "realization" seems to map to the third set of decisions, and the second set is somewhat missing from this paper. One could argue that that omission was intentional: the paper was about the architecture, and what this text is saying is that one of the goals of the architecture was to permit many realizations, a point that might have been listed as another goal. But it is equally important to say that a goal of the architecture was to allow for many different alternatives for mechanism design as well–the design decisions of the architecture should permit a range of mechanism choices, not embed those decisions into the architecture itself. I believe that in 1988 the Internet designers saw, but perhaps did not articulate clearly, that there is a benefit to architectural minimality–that is, to specify as little as possible consistent with making it possible for subsequent mechanisms to meet the goals. Were I writing the paper now, I would add a new section, which draws from the previous sections the set of core principles of the architecture, linking them back to the goals they enable.

Core architectural principles:

Packet switching.
Gateways (what we call routers today)
- Minimal assumptions about what the networks would do.
- No flow state in routers, which implies no flow setup, and thus the "pure" datagram model.
- Implies strict separation of IP from TCP, with no knowledge of TCP in routers.
Co-location of flow state with end-points of flows (fate-sharing).
No mechanisms to report network failures to end-points.
Trust in the end-node.
Minimal assumptions about service functions and performance.

Totally missing from this paper is any discussion of packet headers, addressing, and so on. In fact, much earlier than 1988 we understood that we had to agree on some format for addresses, but that the specific decision did not influence our ability to address the goals in the list. Early on in the design process (in the mid-1970s), variable-length addresses were proposed, which would have served us much better with respect to the goal of longevity. It was rejected because at the time, the difficulty of building routers that could operate at line speeds (e.g. 1.5 mb/s) made parsing of variable-length fields in the header a challenge. In my 1988 list "longevity" is missing–probably a significant oversight. But in the 1970s we made a design choice that favored the pragmatics of implementation over flexibility.

The packet header also embodied other design choices, which we thought we had to make in order to facilitate or enable the design of the second-level mechanisms that flesh out the architecture into a complete implementation.

- *The idea of packet fragmentation supported the goal that we be able to exploit pre-existing networks. Today, Internet is the dominant architecture, and we can assume that issues like network technology with small packet sizes will not arise.*
- *The use of a TTL or hop count was an architectural decision that tried to allow more generality in how routing was done–we wanted to tolerate transient routing inconsistency. The architecture did not specify how routing was to be done (the paper notes the emergence of the two-level routing hierarchy), and indeed it was a goal that different routing schemes could be deployed in different parts of the network.*

The Internet architecture tolerates this variety of realization by design. However, it leaves the designer of a particular realization with a great deal of engineering to do. One of the major struggles of this architectural development was to understand how to give guidance to the designer of a realization, guidance which would relate the engineering of the realization to the types of service which would result. For example, the designer must answer the following sort of question. What sort of bandwidths must he in the underlying networks, if the overall service is to deliver a throughput of a certain rate? Given a certain model of possible failures within this realization, what sorts of redundancy ought to be engineered into the realization?

Most of the known network design aids did not seem helpful in answering these sorts of questions. Protocol verifiers, for example, assist in confirming that protocols meet specifications. However, these tools almost never deal with performance issues, which are essential to the idea of the type of service. Instead, they deal with the much more restricted idea of logical correctness of the protocol with respect to specification. While tools to verify logical correctness are useful, both at the specification and implementation stage, they do not help with the severe problems that often arise related to performance. A typical implementation experience is that even after logical correctness has been demonstrated, design faults are discovered that may cause a performance degradation of an order of magnitude. Exploration of this problem has led to the conclusion that the difficulty usually arises, not in the protocol itself, but in the operating system on which the protocol runs. This being the case, it is difficult to address the problem within the context of the architectural specification. However, we still strongly feel the need to give the implementer guidance. We continue to struggle with this problem today.

This paragraph reflects an issue that could have been explored more clearly. The goal of continued operation in the face of failures (resilience) motivated us to design very good mechanisms to recover from problems. These mechanisms were in fact good enough that they would also "recover" from implementation errors. They papered over the errors, and the only signal of the problem was poor performance. What is missing from the Internet, whether in the architecture or as an expectation of the second-level mechanisms, is some requirement to report when the error detection and recovery mechanisms are being triggered. But without a good architecture for network management, it is not surprising that these reporting mechanisms are missing, because it is not clear to what entity the report would go. Telling the user at the end-node is not useful, and there is no other management entity defined as part of the architecture.

The other class of design aid is the simulator, which takes a particular realization and explores the service which it can deliver under a variety of loadings. No one has yet attempted to construct a simulator which take into account the wide variability of the gateway implementation, the host implementation, and the network performance which one sees within possible Internet realizations. It is thus the case that the analysis of most Internet realizations is done on the back of an envelope. It is a comment on the goal structure of the Internet architecture that a back of the envelope analysis, if done by a sufficiently knowledgeable person, is usually sufficient. The designer of a particular Internet realization is usually less concerned with obtaining the last five percent possible in line utilization than knowing whether the desired type of service can be achieved at all given the resources at hand at the moment.

The relationship between architecture and performance is an extremely challenging one. The designers of the Internet architecture felt very strongly that it was a serious mistake to attend only to logical correctness and ignore the issue of performance. However, they experienced great difficulty in formalizing any aspect of performance constraint within the architecture. These difficulties arose both because the goal of the architecture was not to constrain performance, but to permit variability, and secondly (and perhaps more fundamentally), because there seemed to be no useful formal tools for describing performance.

From the perspective of 2015, this paragraph is very telling. For some goals such as routing, we had mechanisms (e.g. the TTL field) that we could incorporate in the architecture to support the objective. For performance, we simply did not know. (We proposed an ICMP message called Source Quench, which never proved useful and may have just been a bad idea. It is totally deprecated.) At the time this paper was written, our problems with congestion were so bad that we were at peril of failing 2015 goal 10: "It works". Yet there is no mention of congestion and its control in this paper. Arguably, we still do not know what the architecture should specify about congestion and other aspects of performance. We seem to have some agreement on the ECN bit, but not enough enthusiasm to get the mechanism actually deployed. And there are many alternative proposals: XCP [Katabi et al., 2002] or RCP [Dukkipati, 2008], etc. that would imply a different packet header. The debate seems to continue as to what to put in the packet (e.g. specify as part of the architectural interfaces) in order to allow a useful range of mechanisms to be designed to deal with congestion and other aspects of performance.

This problem was particularly aggravating because the goal of the Internet project was to produce specification documents which were to become military standards. It is a well known problem with government contracting that one cannot expect a contractor to meet any criteria which is not a part of the procurement standard. If the Internet is concerned about performance, therefore, it was mandatory that performance requirements be put into the procurement specification. It was trivial to invent specifications which constrained the performance, for example to specify that the implementation must be capable of passing 1,000 packets a second. However, this sort of constraint could not be part of the architecture, and it was therefore up to the individual performing the procurement to recognize that these performance constraints must be added to the specification, and to specify them properly to achieve a realization which provides the required types of service. We do not have a good idea how to offer guidance in the architecture for the person performing this task.

Datagrams

The fundamental architectural feature of the Internet is the use of datagrams as the entity which is transported across the underlying networks. As this paper has suggested, there are several reasons why datagrams are important within the architecture. First, they eliminate the need for connection state within the intermediate switching nodes, which means that the Internet can be reconstituted after a failure without concern about state. Secondly, the datagram provides a basic building block out of which a variety of types of service can be implemented. In contrast to the virtual circuit, which usually implies a fixed type of service, the datagram provides a more elemental service which the endpoints can combine as appropriate to build the type of service needed. Third, the datagram represents the minimum network service assumption, which has permitted a wide variety of networks to be incorporated into various Internet realizations. The decision to use the datagram was an extremely successful one, which allowed the Internet to meet its most important goals very successfully. There is a mistaken assumption often associated with datagrams, which is that the motivation for datagrams is the support of a higher level service which is essentially equivalent to the datagram. In other words, it has sometimes been suggested that the datagram is provided because the transport service which the application requires is a datagram service. In fact, this is seldom the case. While some applications in the Internet, such as simple queries of date servers or name servers, use an access method based on an unreliable datagram, most services within the Internet would like a more sophisticated transport model than simple datagram. Some services would like the reliability enhanced, some would like the delay smoothed and buffered, but almost all have some expectation more complex than a datagram. It is important to understand that the role of the datagram in this respect is as a building block, and not as a service in itself.

This discussion of the datagram seems reasonable from the perspective of 2015, but as I said above, were I to write the paper now I would give similar treatment to some of the other design decisions we made.

TCP

There were several interesting and controversial design decisions in the development of TCP, and TCP itself went through several major versions before it became a reasonably stable standard. Some of these design decisions, such as window management and the nature of the port address structure, are discussed in a series of implementation notes published as part of the TCP protocol handbook [17,18]. But again the motivation for the decision is sometimes lacking. ln this section, I attempt to capture some of the early reasoning that went into parts of TCP. This section is of necessity incomplete; a complete review of the history of TCP itself would require another paper of this length.

The original ARPANET host-to host protocol provided flow control based on both bytes and packets. This seemed overly complex, and the designers of TCP felt that only one form of regulation would he sufficient. The choice was to regulate the delivery of bytes, rather than packets. Flow control and acknowledgment in TCP is thus based on byte number rather than packet number. Indeed, in TCP there is no significance to the packetization of the data.

This decision was motivated by several considerations, some of which became irrelevant and others of which were more important than anticipated. One reason to acknowledge bytes was to permit the insertion of control information into the sequence space of the bytes, so that control as well as data could be acknowledged. That use of the sequence space was dropped, in favor of ad hoc techniques for dealing with each control message. While the original idea has appealing generality, it caused complexity in practice.

A second reason for the byte stream was to permit the TCP packet to be broken up into smaller packets if necessary in order to fit through a net with a small packet size. But this function was moved to the IP layer when IP was split from TCP, and IP was forced to invent a different method of fragmentation.

A third reason for acknowledging bytes rather than packets was to permit a number of small packets to be gathered together into one larger packet in the sending host if retransmission of the data was necessary. It was not clear if this advantage would be important; it turned out to be critical. Systems such as UNIX which have a internal communication model based on single character interactions often send many packets with one byte of data in them. (One might argue from a network perspective that this behavior is silly, but it was a reality, and a necessity for interactive remote login.) It was often observed that such a host could produce a flood of packets with one byte of data, which would arrive much faster than a slow host could process them. The result is lost packets and retransmission.

If the retransmission was of the original packets, the same problem would repeat on every retransmission, with a performance impact so intolerable as to prevent operation. But since the bytes were gathered into one packet for retransmission, the retransmission occurred in a much more effective way which permitted practical operation.

On the other hand, the acknowledgment of bytes could be seen as creating this problem in the first place. If the basis of flow control had been packets rather than bytes, then this flood might never have occurred. Control at the packet level has the effect, however, of providing a severe limit on the throughput if small packets are sent. If the receiving host specifies a number of packets to receive, without any knowledge of the number of bytes in each, the actual amount of data received could vary by a factor of 1000, depending on whether the sending host puts one or one thousand bytes in each packet.

In retrospect, the correct design decision may have been that if TCP is to provide effective support of a variety of services, both packets and bytes must be regulated, as was done in the original ARPANET protocols.

Another design decision related to the byte stream was the End-Of-Letter flag, or EOL. This has now vanished from the protocol, replaced by the push flag, or PSH. The original idea of EOL was to break the byte stream into records. It was implemented by putting data from separate records into separate packets, which was not compatible with the idea of combining packets on retransmission. So the semantics of EOL was changed to a weaker form, meaning only that the data up to this point in the stream was one or more complete application-level elements, which should occasion a flush of any internal buffering in TCP or the network. By saying ”one or more” rather than ”exactly one”, it became possible to combine several together and preserve the goal of compacting

data in reassembly. But the weaker semantics meant that various applications had to invent an ad hoc mechanism for delimiting records on top of the data stream.

Several features of TCP, including EOL and the reliable close, have turned out to be of almost no use to applications today. While TCP is not properly part of the architecture of the Internet, the story of its design and evolution provides another view into the process of trying to figure out in advance what should be in, and what should be out, of a general mechanism that is intended to last for a long time. (The goal of longevity).

In this evolution of EOL semantics, there was a little known intermediate form, which generated great debate. Depending on the buffering strategy of the host, the byte stream model of TCP can cause great problems in one improbable case. Consider a host in which the incoming data is put in a sequence of fixed size buffers. A buffer is returned to the user either when it is full, or an EOL is received. Now consider the case of the arrival of an out-of-order packet which is so far out of order to be beyond the current buffer. Now further consider that after receiving this out-of-order packet, a packet with an EOL causes the current buffer to be returned to the user only partially full. This particular sequence of actions has the effect of causing the out of order data in the next buffer to be in the wrong place, because of the empty bytes in the buffer returned to the user. Coping with this generated book-keeping problems in the host which seemed unnecessary.

To cope with this it was proposed that the EOL should "use up" all the sequence space up to the next value which was zero mod the buffer size. In other words, it was proposed that EOL should be a tool for mapping the byte stream to the buffer management of the host. This idea was not well received at the time, as it seemed much too ad hoc, and only one host seemed to have this problem.[2] In retrospect, it may have been the correct idea to incorporate into TCP some means of relating the sequence space and the buffer management algorithm of the host. At the time, the designers simply lacked the insight to see how that might be done in a sufficiently general manner.

Conclusion

In the context of its priorities, the Internet architecture has been very successful. The protocols are widely used in the commercial and military environment, and have spawned a number of similar architectures. At the same time, its success has made clear that in certain situations, the priorities of the designers do not match the needs of the actual users. More attention to such things as accounting, resource management and operation of regions with separate administrations are needed.

While the datagram has served very well in solving the most important goals of the Internet, it has not served so well when we attempt to address some of the goals which were further down the priority list. For example, the goals of resource management and accountability have proved difficult to achieve in the context of datagrams. As the previous section discussed, most datagrams are a part of some sequence of packets from source to destination, rather than isolated units at the application level. However, the gateway cannot directly see the existence of this sequence, because it is forced to deal with each packet in isolation. Therefore, resource management decisions or accounting must be done on each packet separately. Imposing the datagram model on the Internet layer has deprived that layer of an important source of information which it could use in achieving these goals.

This suggests that there may be a better building block than the datagram for the next generation of architecture. The general characteristic of this building block is that it would identify a sequence of packets traveling from the source to the destination, without assuming any particular type of service with that service. I have used the word "flow" to characterize this building block. It would be necessary for the gateways to have flow state in order to remember the nature of the flows which are passing through them, but the state information would not be

[2] *This use of EOL was properly called "Rubber EOL" but its detractors quickly called it "rubber baby buffer bumpers" in an attempt to ridicule the idea. Credit must go to the creator of the idea, Bill Plummer, for sticking to his guns in the face of detractors saying the above to him ten times fast.*

critical in maintaining the desired type of service associated with the flow. Instead, that type of service would be enforced by the end points, which would periodically send messages to ensure that the proper type of service was being associated with the flow. In this way, the state information associated with the flow could be lost in a crash without permanent disruption of the service features being used. I call this concept "soft state," and it may very well permit us to achieve our primary goals of survivability and flexibility, while at the same time doing a better job of dealing with the issue of resource management and accountability. Exploration of alternative building blocks constitute one of the current directions for research within the DARPA Internet program.

Acknowledgments – A Historical Perspective

It would be impossible to acknowledge all the contributors to the Internet project; there have literally been hundreds over the 15 years of development: designers, implementers, writers and critics. Indeed, an important topic, which probably deserves a paper in itself, is the process by which this project was managed. The participants came from universities, research laboratories and corporations, and they united (to some extent) to achieve this common goal.

The original vision for TCP came from Robert Kahn and Vinton Cerf, who saw very clearly, back in 1973, how a protocol with suitable features might be the glue that would pull together the various emerging network technologies. From their position at DARPA, they guided the project in its early days to the point where TCP and IP became standards for the DOD.

The author of this paper joined the project in the mid-70s, and took over architectural responsibility for TCP/IP in 1981. He would like to thank all those who have worked with him, and particularly those who took the time to reconstruct some of the lost history in this paper.

References

1. V. Cerf, and R. Kahn, "A Protocol for Packet Network intercommunication", IEEE Transactions Communications, Vol. Corn-22, No. 5, May1974 pp. 637-648.

2. ISO, "Transport Protocol Specification", Tech. report IS-8073, International Organization for Standardization, September 1984.

3. ISO, "Protocol for Providing the Connectionless- Mode Network Service", Tech. report DIS8473, International Organization for Standardization, 1986.

4. R. Callon, "Internetwork Protocol", Proceedings of the IEEE, Vol. 71, No. 12, December 1983, pp. 1388-1392.

5. Jonathan B. Postel, "Internetwork Protocol Approaches", IEEE Transactions Communications, Vol. Corn-28, N"d: 4, April 1980, pp. 605-611.

6. Jonathan B. Postel, Carl A. Sunshine, Danny Cohen, "The ARPA Internet Protocol", Computer Networks 5, Vol. 5, No. 4, July 1981, pp. 261-271

7. Alan Shehzer, Robert Hinden, and Mike Brescia, "Connecting Different Types of Networks with Gateways", Data Communications, August 1982.

8. J. McQuillan and D. Walden, "The ARPA Network Design Decisions ' ' , Computer Networks, Vol. 1, No. 5, August 1977, pp. 243-289.

9. R.E. Kahn, S.A. Gronemeyer, J. Burdifiel, E.V. Hoversten, "Advances in Packet Radio Technology", Proceedings of the IEEE, Vol. 66, No. 11, November 1978, pp. 1408-1496.

10. B.M. Leiner, D.L. Nelson, F.A. Tobagi, "Issues in Packet Radio Design", Proceedings of the IEEE, Vol. 75, No. 1, January 1987, pp. 6-20.

11. "Transmission Control Protocol RFC-793", &DN Protocol Handbook, Vol. 2, September 1981, pp, 2.179-2.198.

12. Jack Haverty, "XNET Formats for Internet Protocol Version 4 IEN 158", DDN Protocol Handbook, Vol. 2, October 1980, pp. 2-345 to 2-348.

13. Jonathan Postel, "User Datagram Protocol NICRFC- 768", DDN Protocol Handbook, Vol. 2. August 1980, pp. 2.175-2.177.

14. I. Jacobs. R. Binder, and E. Hoversten, "General Purpose Packet Satellite Networks", Proceedings of the IEEE, Vol. 66, No. 11, November 1978, pp' 1448-1467.

15. C. Topolcic and J. Kaiser, ”The SATNET Monitoring System”, Proceedings of the IEEEMILCOM Boston, MA, October 1985, PP. 26.1.1-26.1.9.

16. W.Edmond, S.Blumenthal, A.Echenique, S.Storch, T.Calderwood, and T.Rees, ”The Butterfly Satellite IMP for the Wideband Packet Satellite Network’ ’ , Proceedings of the ACM SIGCOMM ’86, ACM, Stowe, Vt., August 1986, pp. 194-203.

17. David D. Clark, ”Window and Acknowledgment Strategy in TCP NlC-RFC-813”, DDN Protocol Handbook, Vol. 3, July 1982, pp. 3-5 to 3-26.

18. David D. Clark, ”Name, Addresses, Ports, and Routes NIC-RFC-814”, DDN Protocol Handbook, Vol. 3, July 1982, pp. 3-27 to 3-40. 114

3.1 The relation of architecture to function

The architecture of the Internet as I have defined it here (and as I did in 1988) clearly illustrates the point that architecture does not directly specify how the network will meet its functional requirements.

It is worth looking at the various requirements I laid out in Chapter 2 and considering how the architecture of the Internet relates to meeting those requirements.

Fit for purpose (it works?) Arguably, the Internet is a success. Its design led to a network that has passed the tests of utility and longevity. The basic ideas of packet switching, datagrams (no per-flow state in the routers) and the like were well-crafted. Those of us who designed the original Internet are so pleased (and perhaps surprised) that it works as well as it does that we feel justified in turning a blind eye to the aspects that don't work so well. If the Internet of today is not quite as reliable as the phone system, and routing takes rather long to converge after a transient, we say that after all routing is just one of those "second-level" mechanisms, and not a part of the architecture, and who said that "5 nines" is the right idea for the Internet? But overall, I think it is fair to argue that the architecture of the Internet produced a system that is fit for purpose.

Security In Chapter 7 I will argue that the Internet itself (the packet carriage layer as opposed to the larger definition that includes the applications and technology) can only solve a part of the security problem. Securing the network itself, which seems to call for secure versions of routing protocols, etc., was relegated to that second stage of mechanism design that turns the architecture into a complete implementation. This approach was probably valid, since different circumstances call for different degrees of security. But there is an open question as to whether there are architectural decisions that could make this task easier. Protecting the packet transport layer from abuse by the applications (most obviously in the context of Denial of Service attacks) is an area that the architecture probably needs to address, but the early design did not consider this issue. Overall, the linkage between the key architectural features of the Internet and the requirements for security seem a bit fragmentary and weak.

Availability and resilience In the 1980's we did not understand how to think about availability in general. We understood that packets might get lost, so we designed TCP to recover. But there is nothing in the architecture itself to help with this problem (unless you consider that at this point, the functions of TCP are essentially a part of the architecture). We understood that links and routers might fail, so we needed dynamic routing. The Internet packet header provides a TTL field to allow for dynamic inconsistency in routing. This is an illustration of the point that architecture does not always define how a requirement is met, but tries to make it possible (or easier) for a system designed based on that architecture to meet that requirement. Our intuition was that no other architectural support was needed for routing, or for availability more generally. As I will argue in Chapter 8, an architecture of a future Internet needs to take a more coherent view of availability.

Economic viability There are essentially no features of the Internet's architecture that relate to economic viability, other than the consequences of the core modularity. One way to think about economic viability is that all the actors in the ecosystem created by the architecture must have the incentive to play the role assigned to them by that architecture. In particular, if there is a class of actor that does not find an economic incentive to enter the ecosystem and invest, the design will not thrive. This way of looking at things was roughly understood early on, but we had no tools to reason about it. In fact, the issues have really only become clear in the last decade, with ISPs (which make large capital investments) trying to find ways to increase revenues by "violating" the architecture: peeking into packets, exercising discrimination of various sorts, and so on. As well, the current debates about when interconnection (e.g. peering) should be revenue neutral and whether paid peering should be acceptable illustrate the complexity of the economic landscape. There is nothing in the architecture about accounting, billing, money flows or other issues that relate to economics.

Management As I describe it, the original Internet architecture did not contain any design elements intended to address the issues of network management. We received some criticism from our friends in the telephone industry about this; they said that a major part of the design of the telephone system was to address issues of management: fault detection and isolation, performance issues and the like. Many of the basic data formats used to transport voice across the digital version of the telephone system contain fields related to management, and we were asked why we had not understood that. Our basic headers (e.g. the IP packet header) did not contain any data fields that defined building blocks for network management.

Meet society's needs This very general heading captures a range of issues such as privacy (on the one hand), lawful intercept (on the other hand), resilience of critical services, control of disruptive or illegal behavior by users, and so on. There is very little in my 1988 paper that speaks to these issues. It may not have been clear in 1988 that the way Internet addresses are specified and used (for example) has a material influence on the balance between privacy, traffic analysis, lawful intercept and the like. These issues have now emerged as important, but I do not think we have clear ideas even now about how to deal with them, and in particular how to deal with them in a way that leaves a degree of subsequent flexibility to the implementation and the realization.

One could ask if the principle of architectural minimality is the correct approach. Perhaps the architecture left too many problems for the designers that then had to define the second-level mechanisms such as routing. Perhaps a more expansive definition of what we classify as "architecture" would lead to better outcomes when we deploy the resulting system. Alternatively, perhaps a different approach, with a different conception of what is minimally necessary, might lead to better outcomes. These mechanisms were designed based on our best intuition at the time, but it is reasonable today to rethink these decisions from scratch–what might the architecture do to better support goals such as security and management, which we dealt with poorly if at all in the 1970's. In the next chapter, I develop a framework (one of several in the book) that can be used to compare architectures, and then in Chapter 5 I look at some different conceptions of what an Internet architecture might be, again mostly with a preference for minimality but a very different view of what it is "on which we must all agree".

Chapter 4

Architecture and function

4.1 Introduction

Chapter 2, with its long list of requirements, may in fact distract the discussion from what is perhaps most central: the network has to be fit for purpose–it has to perform a useful set of functions in support of the applications that run over it and the users that employ those applications. So before turning to the question of how the Internet (or a different possible Internet with a different design) might address those various requirements, I want to start with the question of how we describe, in architectural terms, what it is that a network "does".

Computer Scientists often use the word *semantics* to describe the functional capabilities of system–the range of things it can do. However, when it comes to computer networks, what they do is very simple, compared (for example) to an operating system or a database system. The loose packet carriage model of "what comes out is what came in" is intentionally almost semantics-free. The packets just carry bytes. Packet boundaries can have some limited functional meaning, but not much. The original design presumed some constraints that we might view as "semantics", such as global addresses, but the progress of time has violated these and the Internet keeps working. TCP does impose some modest semantic constraints, but of course TCP is optional, and not a mandatory part of the architecture.

What defines the Internet, and the range of behavior that is available in the Internet, is the *expressive power* of the packet header, which has more to do with its format (what we might call its *syntax*) than any semantics. Most fields (e.g. packet length) are unremarkable, some (like the TOS bits) have been redefined several times in the history of the Internet, some (like the options) have atrophied, and some (most obviously the IP addresses) have had a most interesting history in which the only constants are that they are 32 bit fields, that whatever value they have at each end must remain constant for the life of a TCP connection,[1] and that at any locale in the network, they must provide the basis for some router action (e.g., forwarding). They can be rewritten (as in NAT), turned into logical addresses (as in multicast or anycast), and they can be microcoded in a number of ways to capture address hierarchy (net/rest, A/B/C, CIDR). All that really seems to matter is that they are 32 bits long, and that at any point, they must have at least local meaning to a forwarding process.

The evolution in thinking with respect to IP addresses sheds some light on architectural thinking. The initial idea that addresses were drawn from a single global address space and mapped uniquely to physical ports on physical machines turned out not to be a necessary constraint, but just a simple model to get started. We were initially fearful that if we deviated from this definition, the coherence of the network would fall apart, and we would not be able to ensure that the Internet was correctly connected, or debug it when it was not. Indeed, these fears are somewhat real, and it is possible today to "mess with" addresses in such a way that things stop working. But mostly, the Internet continues to work, even with NAT boxes, VPNs and private address spaces, because the consequences of messing with addresses are restricted to regions within which there is agreement to assign

[1] *The source IP address is used at the receiving end to dispatch the packet to the right process. In addition, the TCP computes a checksum over the packet to detect modification of the packet in transit. It incorporates into the checksum parts of the IP header (called the pseudo-header in the spec). For this reason, the IP address in the packet can only be changed taking these limitations into account.*

a common meaning to those addresses. Those self-consistent regions need not be global; it is the scope of the self-consistent binding from addresses to forwarding tables that defines them.

In the limit, each "region" could just be two routers, the sender and the receiver for each hop along the path of the packet. (This would somewhat resemble a scheme based on label rewriting.) Regions this small would be hard to manage without some sort of overarching framework for state management (and would have other drawbacks as I discuss later), but a single global region–the starting point for the Internet design–has also proven to have complexities. In practice, the operational Internet has gravitated to regions that represent some sort of rough balance among the issues that arise from big and small regions.

My point is that the format of the packet header is a defining feature of the Internet, in contrast to assertions about the semantics of addresses. It is for this reason that I focus on the *expressive power* of the packet header as a key factor in the specification of a network architecture.

4.2 Per-hop behaviors

We can generalize from this discussion of addressing and ask more abstractly about the local behavior of routers (and other network elements) and the resulting overall function. In fact, the network is built up of somewhat independent routers. What applications care about is that the local behavior at a sequence of routers (the "per-hop behavior", or PHB) can be composed to achieve some desired results end-to-end.[2] If the packets get delivered (which is really the only thing that defines today's properly operating Internet, except in the context of defense against attack), then the details of how PHBs are configured (e.g., the routing protocols or the like) are a matter left to the regions to work out. The expectation about forwarding is a core part of the architecture, how routing is done is not. (If the packets do not get delivered, then debugging may be more or less a nightmare, depending on the tools for coordination and analysis, but this is a separate issue, which I address in Chapter 10).

Today, a router has a rather simple set of behaviors. Ignoring QOS and source-routes for the moment, a router either picks (one or more) outgoing paths on which to forward a packet, or drops it. The router can have as much state as inventive people define for it–static and dynamic forwarding tables, complex routing protocols, and static tables that define unacceptable addresses (e.g., so-called Martian and "bogon" packets). The router can also rewrite many parts of the packet header. But even today, and certainly looking to the future, not all elements in the network will be "routers". Elements, once they receive a packet, can perform any PHB that does not cause the end-to-end behavior to fail. So when we consider PHBs as the building block of network function, we should be careful not to limit ourselves to a model where the only PHB is "forwarding".

4.3 Tussle

One of the distinctive features of networks and distributed systems is that they are composed of actors whose interests are not necessarily aligned. These actors may contend with each other to shape the system behavior to their advantage. My co-authors and I picked the word "tussle" to describe this process [Clark et al., 2005a]. Sometimes one of the actors is a clear "bad guy": e.g. someone wants to infiltrate a computer against the wishes of the owner. This tension leads to devices such as firewalls, which are an example of a PHB that is not simple forwarding, but rather forwarding or dropping based on the content of the packet. Firewalls are an attempt by the receiver to overrule the intentions of the sender: a PHB that the receiver wants executed on the packet, but the sender does not.

Sometimes the issues are not black and white, but more nuanced: I want a private conversation, law enforcement wants to be able to intercept any conversation with proper authorization. I want to send a file privately, copyrights holders want to detect if I am serving up infringing material. To the extent these tussles are played out "in the net" (as opposed to in the end-nodes or the courts), they will be balanced through the relative abilities of the

[2] *The term "per hop behavior" was coined as part of the effort in the IETF to standardize the mechanisms that underpin the* diffserv *QoS mechanisms [Nichols and Carpenter, 1998, Section 4].*

different actors to exploit the expressive power of the network. So our discussion of expressive power, and the tools that implement it, will be strongly shaped by the reality of tussle. Looking at the balance of power created by a specific feature in the architecture is a way to integrate considerations of security into the design process of an architecture.

4.4 Reasoning about expressive power

As I said at the beginning of this chapter, most computer systems are characterized by a rather details specification of the functions they can perform–what I called the *semantics* of the system. However, the functional capabilities of the Internet are not defined by its specification. If (in general) a network element can be programmed to do "anything" as its PHB, then the resulting overall function is the result of the execution of these PHBs in some order, where the execution of the PHB is driven by the fields in the packet header, and the order of execution is defined by the routing of the packet among these devices. Of course, since the devices themselves can define the routing, the resulting expressive power (the computational power, if you will) is presumably rather complex. Computer scientists are accustomed to thinking about the implications of semantics: what are the limitations of some semantic construct. We are less accustomed (and less equipped with tools) to think about the expressive power of a packet header–what functions are consistent with some format and syntax. It is sort of like asking what ideas can be expressed in sentences of the form "subject, verb, object". The question seems ill-defined and unbounded. Even harder is to catalog what cannot be expressed. But this question is the one that actually captures the limits of what the Internet can and cannot do. So we should try to think about how to think about it.

This view of packet processing has not been seriously explored,[3] because in the Internet of today, the overall function we want to achieve is very simple–the delivery of the packet. If that is the desired overall function, there is not much demand for the complex concatenation of arbitrary PHBs within the network. But as we think about wanting to do more complex things as a packet moves from source to destination (many having to do with security), the range of interesting PHBs will grow. (See Chapter 5 for examples.) So it is worth some consideration of what factors define or limit the expressive power of a network.

In this section, I pose a three-dimensional framework to describe PHB execution: *alignment of interests, delivery* and *parameterization.*

Alignment of interests

The first dimension of the model is to capture the relationship between the sender of the packet and the owner of the element that implements the PHB. This dimension directly captures the nature of tussle. I will propose two cases: *aligned* and *adverse.*

Aligned: In this case, the interests of the sender and the element match. Simple routing, multicast, QoS, etc., usually falls in this obvious class. The sender sent the packet, the router forwards it, and this is what both parties expected.

Adverse: In this case, the PHB performs a function that the sender does not want. A firewall is a good example here, as would be other sorts of content filtering, deep packet inspection, logging and so on.

Delivery

The second dimension of the model is to ask why or how the packet arrives at the element that implements the PHB. There is a simple four-case model that covers most of the circumstances: delivery is either *intentional, contingent, topological* or *coerced* .

[3] *With the exception of some of the Active Network research, which I discuss below and in Section 5.3.*

Intentional: In this case, the packet arrives at the element because it was specifically sent there. For example, with source routes, the route is a series of addresses, each of which directs the packet to the next such router. As another example, a packet arrives at a NAT box because it was intentionally sent there.

Contingent: In this case, the packet may or may not arrive at a given element, but if it happens to arrive, then the PHB will be executed. This is the basic mode of datagram operation–if a router gets a packet it forwards it. There are no pre-established paths from source to destination (which would be examples of intentional delivery). Each router computes routes to all known destinations, so it is prepared to deal if a packet happens to arrive.

Topological: In this case, there is nothing in the packet that causes it to arrive at a particular device, but instead the topology of the network (physical or logical) is constrained to insure that the packet does arrive there. Firewalls are a good example of topological delivery. The sender (assuming he is malicious) has no interest in intentionally sending his attack packet to a firewall. He would prefer to route around the firewall if he could. The receiver wants some assurance that the firewall will be in the path. The receiver will normally not be satisfied with contingent protection. So the remaining tool available is to constrain the connectivity or routing graph so that the only path (or paths) to the receiver pass through the firewall.

Coerced: This can be seen as a special case of intentional or topological delivery in which the sender is compelled to subject itself to a PHB, even though the interests of the sender and the owner of the PHB are adverse. An attacker attempting to reach a machine behind a Network Address Translation box has no choice but to send the packet to that element–there is no other means of reaching beyond it. In this case, we can expect the sender to cheat or lie (in terms of what values are in the packet) if it is possible.

Parameterization

The third dimension of the model is that the packet triggers the execution of a PHB, and thus the data in the packet is in some sense the input to that PHB, like arguments to a subroutine. The values in the packet are the input parameters to the PHB, and if the packet is modified, this is similar to the rewriting of variables in the invocation of a subroutine. (In other words, to use the vocabulary of programming language, the parameters in the packet are passed to the PHB by reference rather than by value.) The element that executes the PHB can have lots of persistent state (which can be modified as a result of the PHB), and can have distributed or "more global" state if suitable signaling and control protocols are devised.

In this context, I will again offer two cases, although these more define ends of a spectrum than distinct modes: *explicit* and *implicit.*

Explicit: While the PHB can in principle look at any data fields in the packet, in common cases there will be specific fields set aside in the header as input to specific PHBs. This is the common case for packet forwarding: since packet forwarding is the basic operation of networking, there is an explicit address field used as input to the forwarding lookup. The Internet (sometimes) supports QoS, so there is an explicit field in the packet that is the input parameter to the QoS algorithm.

Implicit: In other cases, there is no specific field used as input to the PHB: the PHB looks at fields intended for other purposes. Firewalls block packets based on port numbers, some ISPs assign QoS based on port numbers, packets are sometimes routed based on port numbers (e.g., when Web queries are deflected to a cache or an outgoing SMTP connection is deflected to a local mail server.) If the PHBs have state, they can also base their actions on implicit information such as the arrival rate of packets.

Implicit parameters can be expensive. In the worst case (deep packet inspection), the PHB may process the entire contents of the packet as input to its operation. Clearly, this is not as efficient as a pre-designed action

where the PHB picks a preset field (e.g. an address field) and uses this for a table lookup. So implicit arguments must be used sparingly, but in the case of adverse interests, implicit parameters may be the only option.

This model suggests that there is some rough analogy between the expressive power of a network and a programming language of some sort, where the "computation" is a series of subroutine executions, driven by the input parameters carried by the packet, and where the order of execution is defined by the routing protocols, together with the expressive power of the packet to carry the addresses that drive the forwarding. Of course, the addition of tussle and nodes that are hostile in intent with respect to the sender adds a twist that one does not find in programming languages, and in fact this "twist" may be one of the most important aspects of what the network actually "computes". So the power of an analogy to a programming language remains to be explored.[4]

This taxonomy classifies activity based on the alignment of interest among the senders and the PHBs in the network. Another way to classify activities is to look at the alignment of interests between sender and *receiver*. In the case that the interests of the sender and receiver are aligned, then the PHBs would normally be used to enhance the service being provided, unless they are inserted into the path by an actor with interests adverse to the communicating parties. They are *functional*, in that the application being used by the communicants are invoking them as part of the service. (While only the sender can directly control the sending of the packet and its contents, there are certain architectures, which I discuss in Chapter 5, where the receiver as well as the sender can directly exercise control over what PHBs are applied to the packet.) If the interests of the sender and receiver are aligned, then if there is an adverse PHB in the path, it must be there because of some third party (e.g an ISP or a government authority) has interposed it, or because the network itself has previously suffered an attack such that some of its elements have been taken over by an attacker. The resulting questions are first, whether the architecture is providing support to functional PHBs through some aspect of its expressive power (delivery, parameters, etc.) and (the negative aspect of the analysis) whether the architecture needs to provide support to protect the communicants from the misuse of this expressive power, and whether the architecture needs to provide support for the task of detecting and isolating a faulty or malicious element. (See Section 4.9 and Chapter 8 for a discussion of fault diagnosis. [[[Confirm later.]]] If the interests of the sender and receiver are not aligned (in which case the receiver either wants protection during communication or does not want to receive traffic at all), then the PHBs are serving a different purpose: they are deployed to protect the receiver from the sender, a role which creates different potential roles for the architecture. I will return to security analysis in Chapter 7.

4.5 Pruning the space of options

What I just described is a 2x4x2 design space. But in fact it is less complex than that. The method that helps to sort out this space is "tussle analysis", which starts with understanding the alignment of interests.

Aligned: If the sender wants the PHB to be executed, then intentional delivery and explicit arguments make sense. Contingent delivery may be suitable in some cases (e.g. the basic forwarding function), but explicit arguments (e.g. the address field) still make sense.

Adverse: If the sender does not want the PHB to be executed, then it cannot be expected to provide any explicit arguments to the PHB, so the design must be based on implicit approaches. Nor can the PHB count on intentional delivery, so coerced delivery is the best option, with contingent or topological delivery as a fallback.

[4] *This idea is by no means original to me. In an early paper with the delightful title of* Programming Satan's Computer *[Anderson and Needham, 2004], the authors observe: "a network under the control of an adversary is possibly the most obstructive computer which one could build. It may give answers which are subtly and maliciously wrong at the most inconvenient possible moment." Their focus is on the design of cryptographic systems, but their point is more general: "In most system engineering work, we assume that we have a computer which is more or less good and a program which is probably fairly bad. However, it may also be helpful to consider the case where the computer is thoroughly wicked, particularly when developing fault tolerant systems and when trying to find robust ways to structure program and encapsulate code."*

Some examples

NAT boxes Network Address Translation devices (NAT boxes) implement a PHB that is not simple forwarding, but include rewriting of the destination address field. They are as well a wonderful example of how one can disrupt two of the most fundamental assumptions of the original Internet and still have enough functions mostly work that we accept the compromise. The assumptions of the original Internet were that there was a single, global address space, and there was no per-flow state in forwarding elements. NAT boxes, of course, have per-flow state, and early NAT devices, lacking a protocol to set up and maintain soft state, depended on a "trick": they use the first outgoing packet to set up the state, which then persisted to allow incoming packets to be forwarded. This trick does not allow state to be set up for services waiting for an incoming packet that are "behind" the NAT box. (Protocols have subsequently been developed to allow an end-node to "open a port" to a service behind the NAT device. [5])

NAT boxes are an example of intentional delivery with explicit parameters (the addresses and port numbers). If the interests of the end-points are aligned, NATs are mostly a small nuisance; if the interests are not aligned, they provide a measure of protection, and in that respect fall into the *coerced* category.

Firewalls Firewalls, as I described above, are an example of a PHB that is adverse to the interests of the hostile sender (the potential attacker) and thus must depend on implicit information. The firewall has the poorly-defined task of trying to distinguish "good" from "bad" behavior, based on whatever hints can be gleaned from the packets. Normally, all a firewall can do today is a very crude set of discriminations, blocking traffic on certain well-known ports and perhaps certain addresses. The roughness of the discrimination is not necessarily a consequence of the details of the current Internet, but perhaps the intrinsic limits of making subtle discriminations based only on implicit fields in the packets.

This outcome is not necessarily a bad thing. Sometimes users want the blocking to succeed (when they are being attacked) and sometimes they want it to fail (when some third party such as a conservative government is trying to block their access to other sites on the Internet). If we decide to make the job of the firewall easier, we should consider whose interests we have served.

Tunnels Tunnels, or packet encapsulation, is often thought of as a way to control the routing of a packet, but more generally it is a way to interpose an explicit element in the path toward a destination. The encapsulated packet is the explicit information used as input to the end-point of the tunnel. Sometimes the starting point of the tunnel is contingent or topological; some times it is coincident with the sender; sometimes it is intentional. For example, TOR can be seen as an example of nested tunnels, each with explicit information as input to the PHB at each TOR forwarder.[6]

4.6 Tussle and regions

Consider the example discussed above of a firewall, put in place by the receiver to block attacks by the sender. In this adverse circumstance, the receiver must depend on implicit arguments and topological delivery (or coerced, if the architecture permits). For this to work, the region of the network within which the receiver is located must provide enough control over topology (connectivity and routing) to ensure that the firewall is in the path of the packets. The receiver must have sufficient control over this region of the network to make sure that the topology is as desired, and enough trust in the region to be confident that the forwarding will be done as requested.

[5] *The Port Control Protocol [Wing et al., 2013] and the Internet Gateway Device Protocol, part of the UPnP protocols [Open Interconnect Consortium, 2010] allow an end node to set up a new port mapping for a service on the end node.*

[6] *TOR, or The Onion Router, is a set of servers scattered across the Internet with the goal of allowing anonymous communication between parties. By the clever use of nested encryption, a message is sent from TOR node to TOR node, while hiding the identity of the sender from the receiver. Each forwarder peels off a layer of encryption (hence the name–an analogy to peeling the layers of an onion). For information on TOR, see https://www.torproject.org/.*

To generalize, what this illustrates is that different actors within the network (the sender, the receiver, the ISPs, other third party participants) will have the right to control certain parts of the network (or the expectation that certain parts will be operated consistent with their requirements), and within each such region of the network, the expressive power of the parts found there (the PHBs and the routing) will be used to further the intentions of that actor.

The factor that will determine the outcome of the tussle (e.g. the balance of power) is not the PHBs (which, as I noted, can be more or less anything), but the information in the packet that can serve as the input to the PHB, and the order of processing of the packet.

The order of processing arises from the natural nature of packet forwarding: the packet originates in the region of the sender (who thus gets first crack at any desired PHBs), then enters into the global network, and finally enters into the region of the receiver and the PHBs found there. The information that is in the packet at each stage is a consequence of this ordering. For example, the sender can include data in a packet that is used by the PHBs in the region of the sender and then stripped out so that the other regions cannot see it. While the packet is in the global "middle" region, some or most of the packet can be encrypted to prevent it being examined, and so on.

But as I have noted, PHBs can do more or less "anything" that can be derived from the information in the packet, and the routing is under the control of each of these regions. The fixed point in this design is the packet header itself. So when we think about putting more or less expressive power into the header (e.g. a more or less expressive format), we should consider whether the different options shift the balance of power in ways that match our preferences.

4.7 Generality

I have been talking about PHBs in a rather abstract and general way. As I have used the term, it could equally apply to a low-level function like forwarding or an application-specific service like content reformatting or detection of malware. The taxonomy of delivery modes, alignment of interests and parameter modes applies equally to both general, packet level PHBs and higher level services. One could use the term PHB generally to mean any service element that is inserted into a data flow, or restrict the term to lower level functions like firewalls or routers. Since my goal is to discuss the role of architecture, I will prefer to restrict my use of the term PHB to cases where there might be a benefit to adding to the expressive power of the architecture as part of invoking the PHB. In general, application-level services would would not fit into this category, but this is a presumption, not a given, as some architectures directly support the insertion of application-level services into the flow of packets.

Structurally, it would make sense to assume that higher-level services are intentionally inserted into the pattern of communication by the design of the app. The design of the email system specifies that mail is sent to a mail forwarding agent, and the packets are addresses to that element–intentional delivery. In this case, especially where the packets of the data flow are reassembled into a larger unit for processing (an application data unit or ADU), the explicit parameters used by the service are in the body of the packets, not the packet header. That sort of data is not part of the architecture–it is not something on which there has to be agreement; quite the opposite. However, it would be an error to assume that all application-specific services are invoked by intentional delivery of the packets. Especially where the interests of the communicants and the network are not aligned, the network may try to intercept the communication using topological delivery in order to inspect (e.g., DPI) or modify the contents; an intervention that is thwarted if the data is encrypted, which in turn leads to complaints by network operators that encryption prevents them from managing their network properly. I consider this sort of tussle in Chapter 7, but from the point of view of balance of control, it would seem that a network operator should have to make a very strong argument that it is appropriate for them to be inserting a 'service" into a communication where at least one end-point did not request or expect that service to be there.

However, it is conceivable that there might be some understanding that PHBs provided by the network should have some visibility into what is being sent. As part of the overall architectural design of the system, and balancing the interests of the different parties, it is a valid question as to whether there should be any parameters that

allow the sender to reveal what sort of treatment the packets should receive, to allow for accounting and traffic planning and the like. My preference for architectural minimality (and as well, concerns over security I discuss later), would lead to a conclusion that while adding expressive power to the header may be very beneficial, the option should be used sparingly.

4.8 Architectural alternatives for expressive power

Using the lens of *expressive power*, here are few architectural concepts that would change (usually enhance) the expressive power of a design. Some of these have been incorporated into the alternative architectures I discuss in the next chapter. I mention some of those proposals briefly here.

Addressing

It is generally recognized that the current approach of using the IP address both as a locator and as an identifier was a poor design choice. Mobility is the obvious justification for this conclusion. In today's Internet, dealing with mobility is complicated by the fact the IP address is used both for forwarding and for end-node identity. Separating these two concepts into two different data fields in the packet would allow the location field (e.g., that data that is input to the forwarding PHB) to be changed as the mobile host moves. This division does not solve two resulting problems: how to keep the location information up to date, and how to make sure the identity information is not forged. Linking identity to location provided a weak form of security: if two machines have successfully exchanged packets, the location is sufficiently unforgable that it can stand as a weak identifier. But by separating the two problems, they can each be resolved separately, and managed differently in different situations as circumstances require.

An alternative design approach might result in two fields, or perhaps three, each serving a distinct purpose.

- Locator: This field is used as input to the forwarding PHB of a router. It may be rewritten (as in a NAT device), highly dynamic (in the case of a mobile device) and so on.
- End point Identifier (EID): This field is used by each end of the connection to identify itself to the other end(s). There are in general three issues with such a field: how to make sure a malicious sender cannot forge a false identifier, how each end associates meaning with this field (is there some sort of initial exchange of credentials associates with the EID, or do high-level protocols associate some meaning with it once the connection is in place), and third, should elements other than the end-nodes (e.g. PHBs in the network) be allowed to see and exploit this value?
- In-network identifier (INID): if the decision is taken that the EID is private to the end-nodes of a connection, then there may be need for some other identifier that can be seen and used by PHBs in the path from the sender to the receivers. This possibility raises many sub-questions in turn, such as how the INID is obtained, whether there are security issues associated with its use, for what duration is it valid, and so on.

So while the general idea of the locator-identity split is well understood, there is no clear agreement on how to design the system that would result. Most of the architectures that I will discuss in Chapter 5 implement some sort of location-identity split, and illustrate the range of approaches that have been taken to address this issue.

Increasing the expressive power of a design

If there seems to be some value (some increase in function or generality) from the ability to provide richer input data to a PHB, it is worth at least briefly speculating on how this might be done. I have argued that since a PHB can in principle compute "anything", the expressive power of an architecture will depend on what arguments can be presented to the PHB–in other words what data is captured in the packet header. Here a a few options, quickly sketched.

Blank "scratch-pad" in the packet A simple idea is to leave a fixed, blank area in the packet header, to be used creatively from time to time. One need only look at all the creative ideas for reuse of the fragment offset field to appreciate just how powerful a little extra space can be. To avoid the issues that arose with the IP option field, the expectation for this field should be that a contingent element would not normally look at it. Only elements that have the specific requirement for an input value would parse the field. This might most easily be implemented as a rule that says only the intentional recipient of a packet will examine the scratch-pad area.

The drawback of this scheme is that there might be more than one PHB along the path from the sender to the receiver, so there might be a conflict as to how the scratch-pad should be used. So we might consider a more complex scheme.

Push-down stack model A more complex model for explicit data in packets is a pushdown stack of records of explicit data, carried as part of the packet header. In this model, the packet is explicitly directed by the sender to the first element that should perform a PHB using data from the stack. That element (conceptually) pops the first record off of the stack of explicit information and uses it as input to the PHB. Then, using either stored PHB state or information in the record that was just popped off the stack, it identifies the next element to which the packet should go. This PHB can push a new record onto the stack, or leave the one provided by the original sender, based on the definition of the intended function.

Issues of performance would suggest that the design would not literally pop a record off a stack (thus shortening the packet and requiring that all the bytes be copied.) A scheme involving offset pointers could be devised that would achieve the desired function.

The push-down stack model can be seen as a more nuanced alternative to the IP option field. One way to describe the problem with the IP option was that it was conceived more in the spirit of contingent execution rather then intentional execution. The sender sends the packet addressed to the destination, and routers along the path can look at the options to see what they are supposed to do with it. In the context of aligned interests and per-flow state, we can see a movement toward intentional delivery of packets to nodes with specific PHBs. The push-down stack model (and the more simple scratch-pad model) are more attuned to the intentional delivery model.

This sort of mechanism seems to build on the rough analogy between PHB sequencing and some sort of programming language. And packet encapsulation is a rough version of a push-down mechanism, in which the whole header is "pushed" onto the stack by the encapsulating header. A related use of a push-down stack in the header can be found in two of the architectures I will describe in Chapter 5, i3 and DOA, which use a push-down stack to carry the sequence of IDs that order the sequence of PHB executions.

A heap The proposal for a Role-based Architecture (RBA) [Braden et al., 2003] (part of the NewArch project) is perhaps the closest example of an architecture that captures the idea of general PHBs and the expressive power of a packet header. In this proposal, PHBs are called roles, and the data that is input to each node is called the Role-specific Header, or RSH. The packet header is described as a heap of RSH's. The implication of the *heap* is that the roles are not always executed in a pre-determined order, so the idea of push and pop is too constraining. RSH's are an example of explicit arguments. The proposal discusses both intentional and contingent delivery, where the intentional addressing would be based either on the ID of a role, or the ID of a role at a specific node. The paper does not delve into tussle to any degree, or work through the case of roles that are adverse to the interest of the sender, so there is not much attention to implicit arguments or to topological delivery. However, the idea of a network as a sequence of computations performed on a packet based on explicit input arguments is the core concept of role-based architecture.

Active Networks The concept of Active Networks, initially proposed in [Tennenhouse and Wetherall, 1996], was that packets would carry small programs that the routers would execute in order to process the packet. In

other words, the packet rather than the router would define the PHB. This idea may well define the end-point on the spectrum of expressive power. I defer the discussion of Active Networks to the next chapter, in Section 5.3.

Per-flow state

I have described the current Internet as more or less a pure datagram scheme, in which each packet is treated in isolation, there is no per-flow state, so all the parameters to the PHB must come from the packet header. Per-flow state in the router can enrich the range of PHBs that can be invented, by linking the treatment of different packets in a sequence.

Signaling and state setup In the original Internet, the designers avoided any hint of a signaling protocol or setting up per-flow state in the routers. There were several reasons for this preference. One was simplicity–if we could do without we would avoid yet another thing that could go wrong. In particular, once per-flow state is instantiated in a router, then it has to be managed. When should it be deleted? What happens if the router crashes? The simplicity of the stateless model makes it easier to reason about resilience and robust operation.

Another reason is overhead. It seems a waste to go to the overhead of setting up state for an exchange that may involve only one packet. Much better to have a system in which the sender can "just send it". But if this works for one packet, why not for all the packets?

However, control messages can be an important aspect of the expressive power of an architecture. Control messages may play a selective role in the design. Per-flow state might only be needed in specific elements to deal with special cases. Second, we are now dealing with per-flow state (e.g. in NAT boxes) whether we design for it or not. And some emerging ideas such as indirection schemes depend on per-flow state. So it seems worth revisiting this design decision.

State initiation bit If we are prepared to consider per-flow state as part of the design, we need to consider whether the protocols should include a standard way to establish and maintain this state. The original preference in the Internet design was to avoid an independent control plane as a mandatory component of the network. (Of course, there is no way to prevent parties from attaching controllers to the network if they choose to, but these would not be a part of the architecture.) The original design preference was to carry control information (to the extent that it existed at all) using fields in the data packets, which flowed along the data forwarding path. It is possible to imagine a similar scheme as a standard means for an end-node to establish and maintain per-flow state in intermediate elements.

Such an idea would enrich the expressive power of the packet header by building the idea of state establishment into the design, which would link the treatment of a succession of packets.[7]

Without claiming that all the details are worked out, one can imagine that just as TCP has a state-establishment phase and a connected phase, protocols that establish state in intermediate elements could follow the same pattern. A bit in the header (similar to SYN) could signal that the packet contains state-establishment information. This packet might require more processing overhead (and thus represents a vector for DDoS attacks), but in normal circumstances would only be sent at the initiation of a connection. Once the state is established, some much more efficient explicit indication in the packet could link subsequent packets to that stored state. The two sorts of packets could have different formats.

Maintaining state in intermediate elements Assuming that the state is soft-state (a choice that could be debated), the protocol should include a means to reinstate the soft state if it is lost. One could imagine a new sort of ICMP message signaling that some expected state is missing. To recover from this, the sender would have to

[7] *A related activity in the IETF is SPUD, an acronym variously expanded as Session Protocol Underneath Datagrams, Substrate Protocol for User Datagrams, or Session Protocol for User Datagrams. Like any protocol that creates a control/communication path between end nodes and the network, SPUD raises security questions which received attention due to the Snowden leak [Chirgwin, 2015].*

transition back from a fully "connected" mode into a state-setup mode. One could imagine that the sender could re-establish the state in two ways. First, it could do so "from scratch" by sending whatever initial information was used. Second, the intermediate node that holds the state could send back to the source a bundle (perhaps encrypted) of state information that could be used to re-establish the state efficiently, re-sent from the source on demand.

Such a scheme might make sense in the special case of intentionally sending a packet to an anycast address. In this case, the sender is sending to a logical service, but the actual physical machine implementing the service might change. In this case, it might be necessary to reestablish some state in that box.

In-network state associated with receivers The discussion above covered the case of a sender establishing state along a path as part of session initiation. But an equally common case is state set up along a path that arises from the receiver rather than the sender. Setting up and maintaining this state is actually the trickier part of the scheme.

As an illustration of the problems, consider the case where, as a part of protecting the receiver from attack, connection validation is outsourced to a set of indirection elements. Since a sender (either legitimate or malicious) may connect to any one of these (perhaps using an anycast address), every one of these elements must have available the information necessary to validate all acceptable senders, or else there must be an authentication protocol for those devices to send off credentials to a back-end service. At a minimum, the protection devices need to be able to find this service.

In practice, this pattern sounds more like hard state, somewhat manually set up and torn down, rather than dynamic soft state.

In other cases, soft state may make more sense. A transient service behind a "firewall of the future" may want to open an incoming port (assuming that a future network has ports, of course), and this may best be done as a dynamic setup of soft state. In this case, mechanisms will need to be provided to make sure the state is still in place, even though the receiver is not necessarily sending any data packets.

4.9 PHBs and control of network resources

I observed above that with the exception of architectures that allow for complex PHBs, the objective of the network is very simple–deliver the bits. But a necessary component of delivering the bits is that the network has to manage its resources to that end. These functions, which come under the heading of control and management, are critical but less well studied than the actual forwarding of data. In the early days of the Internet, just getting the packet forwarding right was so challenging that we did not have much attention left over to think about network control. As a result, a key control issue–network congestion and our inability to control it effectively–was a major impediment to good network performance (actually delivering the bits) until the mid-80's, when Van Jacobson proposed a congestion control scheme that is still in use today [Jacobson, 1988]. Since then, there has been a great deal of research on congestion control algorithms, but the relationship between architecture and network control is poorly understood.

An important component of a router's PHB is the manipulation of data related to the management and control of the network. Routers perform tasks that are fairly obvious, such as counting the packets and bytes forwarded. The PHB related to system dynamics, such as congestion control, may be more complex, and will relate to what data the router retains about the traffic it is forwarding. I will return to this issue, and the relation of architecture to network control and management, in Chapter 10.

Debugging

All mechanisms fail. Complex mechanisms fail complexly. If we design a network that permits all sorts of complex routing options and invocation options for PHBs, the potential for failure will certainly go up. Tools to debug and recover from such failures will be critical if we are to meet goals of availability and usability.

PHBs that are contingent are the hardest to debug, since the sender did not invoke them intentionally. The idea of trying to diagnose a failure in a box the sender did not even know about is troubling. This fact suggests that when effective diagnosis is desired, the design should prefer intentional invocation of PHBs.

If the interests of all parties are aligned, it would make sense that the tools for debugging would be effective and useful. However, if the interests of the parties are adverse, the situation becomes more complex. If, for example, an attacker is being thwarted by a firewall, it may be in the interest of the firewall to prevent any sort of debugging or diagnosis of the failure. The goal (from the point of view of the defender) is to keep the attacker as much as possible in the dark as to what is happening, so as to prevent the attacker from sharpening his tools of attack. So while tools and approaches for debugging and diagnosis must be a part of any mechanisms to provide expressive power for a future Internet, tussle issues must be taken into account in their design.

(Certain classes of failure are easy to debug, even for contingent PHBs. Fail-stop events that cause the element not to function at all can be isolated and routed around just like any other router failure. "Fail-go" events do not require diagnosis. It is the partial or Byzantine failures of a contingent PHB that may cause diagnosis problems for the sender. It is for this sort of reason that intentional invocation of PHBs is to be preferred unless the goal of the PHB is to confound the sender.)

4.10 Expressive power and evolvability

In this context, the term evolvability refers to the ability of the network architecture to survive over time and evolve to meet changing needs while still maintaining its core coherence. Chapter 6 explores this issue in depth. But here I consider the relationship between the expressive power of an architecture and how that architecture may evolve over time. The history of the Internet provides some informative case studies.

In the early days, the designers of the Internet thought that the concept of a single global address space was part of the Internet architecture, and we bemoan the emergence of NAT devices, VPNs etc, as an erosion of the architectural coherence of the Internet. To some extent this is true; NAT makes the deployment of passive services behind the NAT barrier more complex, and leads to such inelegancies as STUN. On the other hand, it is also clear that in the large, the Internet has survived the emergence of NAT, and perhaps global addresses did not need to be such a central assumption of the presumed architecture.

Perhaps less mourned but more relevant is the atrophy of IP options. IP options were developed to allow for future evolution of the architecture, and they could have provided a substantial degree of expressive power. However, IP options were hard to process in the fast path of routers, and were deprecated in practice to the point where they are essentially gone. They vanished. One could speculate about the implications of this fact:

- Perhaps this degree of expressive power was not in fact necessary, and made the network over-general.
- Perhaps IP options were not well designed, and required much more processing than a better-designed option.
- Perhaps the loss of IP options represents an un-orchestrated decision to favor short-term cost reduction over future evolvability.

However, at the same time that we have seen IP options atrophy, there have been any number of papers that try to add some new functionality to the Internet by repurposing under-used fields in the IP header, in particular the fields related to fragmentation. This behavior suggests that some additional expressive power in the header would have been of great benefit.

Whatever the mix of actual reasons is, one can learn two lessons from the above.

First, avoid mechanisms that are costly to maintain when they are not needed. For example, if there are fields in packets that are used to carry "extra" input values to PHBs, design them so that only the device that actually implements the PHB has to parse those fields or otherwise pay any attention to them. If the packet is intentionally addressed to the device, then the processing rule is clear: if the packet is not for you, don't look at the extra fields.

Second, any mechanism added to a packet header should have at least one important use from the beginning, to make sure that the implementation of the mechanism remains current. If designers propose something intended to facilitate evolution, but cannot think of a single use for it when it is proposed, perhaps it is overkill and will atrophy over time.

Finally, the addition of tools to promote evolvability may shift the tussle balance, so enthusiasm for rich expressive power may need to be tempered by a realistic assessment of which actors can exploit that power. Indeed it would seem that the goal of evolution over time is inseparable from the goal of operating in different ways in different regions of the network at the same time, in response to different perceived requirements within those regions.

Making design choices about the potential expressive power of a future Internet seems to call for a tradeoff between evolvability and flexibility on the one hand, simplicity and understandablity on the second hand, and tussle balance on the third hand. However, there is no reason to think that this tradeoff is fundamental. Creative thinking might lead to alternative ways of defining packets and routing such that we gain in all three dimensions. To explore this space, it may be helpful to ask ourselves challenge questions of the sort that a clean slate thought process invites, such as why do packets have to have addresses in them, or why do we need routing protocols?

4.11 What is new

It could be argued that in introducing the term *expressive power*, I have actually not said anything new. What is the difference between discussing the expressive power of an architecture and just discussing the architecture? I use the term *expressive power* to draw attention to and gather together the aspects of architecture that relate to its network function, as opposed to other aspects that might relate to issues of economic viability or longevity. Equally important is to conceptualize expressive power in the context of PHBs and how they are invoked. Some PHBs can be designed and deployed without any support from the architecture: we have added firewalls and NAT boxes to the current Internet more or less as extra-architectural after-thoughts. But thinking about expressive power in the context of invoking PHBs is a structured way to reason both about function and about security–indeed I will argue that the taxonomy I offered for how PHBs can be invoked and alignment of interest will provide a structured way to reason about the security implications of an architecture.

PHBs and layering

There is a convenient fiction that some Internet architects, including me, like to propagate, which is that there are a limited set of functions that are "in" the network, but that most of the elements that we find intermediating communication today (e.g., "middleboxes") are somehow "on" the network but not "in it". This fiction lets us continue to argue that what the Internet itself does (and what as architects we might be responsible for) continues to be very simple, and the "middlebox mess" is someone else's problem. It is not hard to argue that complex services such as content caches are not "in" the network, but things like firewalls and NAT boxes are harder to ignore.

One basis to define a service as "on" or "in" is which actor operates it. ISPs operate the Internet, so if the element is not under the control of an ISP, how can it be "in" the network? In this respect, an ISP might correctly say that since it is not responsible for an element that it does not operate, and since the ISP has the responsibility to make sure the packet carriage function continues to work even if such services fail, these other services must be at a higher layer. Indeed, sorting different PHBs along an axis of which depend on which is a good design principle. Very few network operators would allow an element (with its PHB) that they do not control to participate in the routing protocol of the region, for the same reason that the early design of the Internet did not anticipate that hosts would participate in the routing protocols. (This view is consistent with the architectural goal of minimal functional dependency, which I discussed in Chapter 1. An ISP that is providing a packet carriage service should work to ensure that the availability of that service does not depend on any element over which they have no control.) However, the taxonomy of how PHBs are invoked (modes of delivery, parameters, etc.) is a cleaner way to classify different PHBs than "in" or "on". As I look at the relation between architecture and economic viability

in Chapter 9 [[[check]]]I will argue that design of expressive power will in fact shape which actor is empowered to play one or another role in making up an Internet out of its parts, which is a critical factor in the economic viability of an architecture. These design alternatives, which derive from such things as intentional vs. contingent delivery, are the important issues, not a vague concept of "in" or "on".

Chapter 5

Alternative network architectures

5.1 Introduction

[[[Note to readers of this draft. The discussion of the FIA projects in this chapter is based on earlier material I have written in the course of the program. The project descriptions are perhaps out of date in parts, and are no doubt a bit brief. I am engaging each of the projects to produce a more substantial version that they each consider current.]]]

Talking about network architecture in the abstract can seem, in a word, abstract. Chapter 3 used the Internet as one case study, but it is useful to have more than one example to draw on. Having several examples helps the analysis to tease apart what is just a consequence of some particular prior design decision, and what is perhaps more fundamental. The motivation for this book arose in the context of the U.S. National Science Foundation's Future Internet Architecture project (FIA) and its predecessors. As well, there have been projects in other parts of the world that have developed distinctive visions for a future Internet. However, the NSF Future Internet program was not the first moment when the research community has considered an alternative network architecture. There have been a number of proposals, going back at least 25 years, that have looked at different requirements and proposed different architectural approaches. In this chapter I review a selection of the earlier proposals for a new network architecture, and then briefly describe the FIA projects, so that I can draw on their similarities and differences in the subsequent chapters. In the Appendix to this book, I provide a somewhat more complete review of proposals for addressing and forwarding.

As examples of proposals for an alternative network architecture I look at the following:

- Two requirements documents from the time of the proposal for the National Information Infrastructure (the NII) [National Telecommunications and Information Administration, 1993]:
  the Cross Industry Working Team Report [Cross-Industry Working Team, 1994] and the Computer Systems Policy Project [Computer Systems Policy Project, 1994],
- Application Level Framing (ALF) [Clark and Tennenhouse, 1990],
- the Metanet [Wroclawski, 1997],
- Plutarch [Crowcroft et al., 2003],
- Triad [Cheriton, 2000],
- the DARPA New-arch project [Clark et al., 2004],
- Data-Oriented Network Architecture [Koponen et al., 2007],
- the Framework for Internet Innovation or FII [Koponen et al., 2011],

- Publish/Subscribe Internet Routing Paradigm (PSIRP and PURSUIT) [Trossen et al., 2008, Trossen and Parisis, 2012],
- Network of Information (Netinf) [Dannewitz et al., 2013],
- Internet Indirection Infrastructure (i3) [Stoica et al., 2004],
- Delegation Oriented Architecture (DOA) [Walfish et al., 2004],
- Delay/Disruption Tolerant Network Architecture (DTN) [Fall, 2003],
- ANTS [Wetherall, 1999]
- Active Networks [[[TBD]]],
- What else?

A review chapter such as this faces an inherent dilemma. it can either describe the projects in turn, which provides the reader with a perhaps coherent view of each proposal but a weak basis for comparison, or it can look at different requirements and how different architectures address these requirements, which gives a better framework for comparison but may not paint a complete picture of each architecture. My approach is to do some of both–first look at architectures through the lens of their driving requirements, and then summarize the FIA architectures themselves.

5.2 Different requirements–different approaches

In some respects, architectural proposals are creatures of their time. Since the Internet has proved quite resilient over time (an issue I consider in Chapter 6), it is interesting that many of the proposals are driven by a concern that the Internet cannot survive one or another change in the requirements it faces.

As I have noted before, both I and many of the architectural designers discussed here have a preference for architectural minimality, but that minimality is shaped extensively by the set of requirements they choose to address.

NewArch The NewArch project spent a great deal of its effort trying to understand the set of requirements that a successful future architecture would have to address. The list of requirements discussed in the final report include economic viability and industry structure, security, dealing with tussle, supporting non-technical users (balanced with a desire for user empowerment), the requirements of new applications and new technology, and generality. This list has a lot in common with the set of requirements I have discussed in Chapter 2, which is not an accident. The NewArch work laid the foundation for much of my subsequent thinking. While NewArch did propose some distinctive mechanisms, which I will discuss in their place, the discussion of requirements is perhaps as important a contribution as the exploration of new mechanism.

Requirement: regional diversity in architecture

Today, the Internet, with its particular format for packets, seems to have dominated the world. In earlier times, there was much less confidence in the research community that this outcome would prevail. There were competing architectural proposals, in particular Asynchronous Transfer Mode, that were claiming to provide an alternative architecture with an alternative protocol stack. This potential outcome drove the development of a number of higher-level architectural frameworks that were intended to allow different architectures, each running in a region of the Internet, to be hooked together so as to provide end-to-end delivery semantics supporting a general range of applications.

One very early proposal that addressed this idea was ALF, but since this was not its major goal, I postpone its discussion.

Metanet The Metanet, described in a white paper by Wroclawski in 1997, laid out the requirements for such a regional network very clearly. Here are some quotes from the Metanet white paper:

> *We argue that a new architectural component, the region, should form a central building block of the next generation network.*
>
> *…*
>
> *The region captures the concept of an area of consistent control, state, or knowledge. There can be many sorts of regions at the same time - regions of shared trust, regions of physical proximity (the floor of a building or a community), regions of payment for service (payment zones for stratified cost structures), and administrative regions are examples. Within a region, some particular invariant is assumed to hold, and algorithms and protocols may make use of that assumption. The region structure captures requirements and limitations placed on the network by the real world.*
>
> *…*
>
> *[D]ata need not be carried in the same way in different parts of the network - any infrastructure which meets the user's requirements with high confidence can be used to construct a coherent application. Packets, virtual circuits, analog signals, or other modes, provided they fit into a basic service model, are equally suitable. The overall network may contain several regions, each defined by the use of a specific transmission format.*
>
> *…*
>
> *Paths of communications must thus be established in a mesh of regions, which implies passing through points of connection between the regions. We call these points waypoints.*
>
> *…*
>
> *Three essential aspects of the Metanet are a routing and addressing system designed for region-based networks, end-to-end communication semantics based on logical, rather than physical, common data formats, and abstract models for QoS and congestion management; mapped to specific technologies as required.*

While the Metanet white paper lays out these requirements, it does not propose a specific architectural response–this is posed as a research agenda.

Plutarch In developing an architecture to meet these requirements, a key question is what, if anything, the regions share in common. Are there common addresses or names, for example, and at what level in the architecture? A specific proposal for a multi-region architecture is Plutarch, by Jon Crowcroft and his co-authors. Plutarch is an experiment in minimality–an attempt to put together a cross-region "glue" architecture that makes as few assumptions as possible about common function, common naming, and so on. In Plutarch, regions (the Plutarch term is *contexts*) have names, but they are not globally unique. Within a region, addressable *entities* have names, but they are also not unique beyond the scope of a region. Regions are hooked together by interconnection entities (the Plutarch term is *interstitial functions* or IFs) that have names within the regions. To deliver data, Plutarch uses a form of source address, which is of the form (entity, region, entity, region,…entity). The sequence of entity values name the interconnection entities that connect to the next region, where the next entity name has meaning, until the final entity name describes the actual destination point. Source strings of this form are only meaningful in the context of a particular source region, where the initial entity name is well-defined and unique.

Plutarch includes a mechanism for state establishment at the region boundaries, to deal with the conversions that are required. In the view of the authors, there might be many regions, but perhaps only a few *types* of regions (ten or less) so the number of conversions that would have to be programmed was practical.

FII A key assumption of Plutarch was that the various architectures were pre-existing, and had to be taken as given. This assumption drove many of the basic design decisions, since Plutarch could make only the most minimal set of assumptions about the feature of each regional architecture. In contrast, the Framework for Internet Innovation (FII) made the assumption that the various architectures would be specified in the context of the overarching FII design, so FII could make much stronger assumptions about what the regional architectures would support. At the same time, the authors of FII again strove for minimality–they wished to constrain the different architectures as little as possible while meeting the basic requirements they identify. FII identifies three critical interfaces. The first, similar to Plutarch, is the region interface. The second is the API at the application layer. Plutarch does not emphasize this interface, but it is implicit in the design of Plutarch that the end-points share a common view of the semantics of the interchange. The third critical component of FII is a common scheme to mitigate DDoS attacks. Their view is that DDoS attacks must be mitigated at the network level, and require a common agreement on an approach. Their approach, the *shut up message* or SUM, requires that all regions implement a rather complex trusted server mechanism, and requires an agreement to carry certain values intact across the region boundary.

The central mechanism they describe at the region interface is an agreed means to implement routing. Their approach is pathlets [Godfrey et al., 2009], but they stress that an alternative mechanism might be picked. However, since there must be global agreement on the scheme, it has to be specified as part of the architecture. In fact, there are a number of values that have to be passed across the region boundaries, which implies that there must be an agreed high-level representation for the values: the destination address, the information necessary to mitigate DDoS attacks, and so on. Any regional architecture that is to be a part of the FII system must comply with the requirement to support these values and the associated mechanisms. In this respect, as noted above, FII imposed a much larger set of constraints on the regional architectures than does Plutarch. In part, this reflects the different design goal. Plutarch is intended to hook together preexisting architectures. FII is intended to allow new regional architectures to emerge over time, within the pre-existing constraints imposed by FII. It is the view of the FII that the constraints are minimal.

In fact, there might well be some argument as to whether FII is under-specified. For example, the authors take the view that there is no need for any abstract model to deal with congestion or quality of service, in contrast to Metanet, which considered congestion to be one of the key problems that must be dealt with globally.

Discussion One of the key challenges with both Plutarch and FII is the balance between how much per-flow state is created and retained at the regional boundaries, and the range of data conversions that can be done there. FII assumes that the devices at the regional boundaries have no knowledge of the semantics of the transport protocol, so all parameters related to the payload itself must be embedded in the payload transported within the various regional architectures. The FII paper hints that across all regional architectures there must be a common concept of a "packet", which can be converted into a specific representation in each regional architecture.

It is perhaps informative to compare the goal of Plutarch or FII with the goal of the original Internet. The original goal was hooking together disparate networks: the ARPAnet, a satellite network and a packet radio network. How does the solution the Internet took to this challenge differ from the approach of Plutarch or FII? When dealing with the interconnection of disparate technologies, there are two general approaches: *overlay* or *conversion.* In a network architecture based on *conversion*, such as Plutarch or FII, the assumption is that the interconnected networks provide, as a native modality, a service that is similar enough that the service of one can be converted to the service of the other. Given this approach, what a conversion architecture such as Plutarch or FII must do is to define an abstract expression of that service in such a general way that the conversion is likely, while still making it possible to build useful applications on top of the service. In contrast, an *overlay* network defines an end-to-end service, perhaps expressed as a common packet format and the like, and the underlying service of each type of network is used to carry that service over its base service. So, in the case of the Internet, the basic transport service of the ARPAnet was used to carry Internet packet, rather than trying to somehow

convert an abstract Internet packet into an ARPAnet packet.

When the Internet was being designed, the designers did not think of it as an overlay network. The term did not exist back then, but more to the point the term has come to have a slightly different meaning. As the Internet architecture has gained dominance, the need to deal with different regional architectures has faded. Today, the term *overlay* network is used to describe a service that runs on top of the Internet to provide some specialized service, such as content delivery. The possibility that such a specialized service might want to run over heterogeneous lower layer network architectures, such as the Internet and "something else," is not particularly relevant. But in the beginning, it was the presence of those disparate networks that made it possible for the Internet to exist.

In this context, FII is conceived as solving a very particular problem–by abstracting the service model away from the details of how it is implemented (e.g., the packet formats, and the like) it should be possible to move from one conception of the embodiment to another over time, as new insights are learned about good network architecture design. It is the specification of that abstract service model, and the demonstration that it is really independent of the current embodiment, that is the key architectural challenge for an architecture based on *conversion.*

As I write this book in 2016, the latest technology that might call for a heterogeneous regional architecture is what is currently called Internet of Things (IoT). This technology space, previously called sensor and actuator networks, involves devices that may be very low power, fixed function, and perhaps wireless. There is a hypothesis that the current Internet protocols will not serve these sorts of devices well. The challenges go beyond simple performance–the IoT environment raises issues related to management and configuration, issues that the current Internet architecture does not address at all. However, my suspicion (again, as I write this in 2016) is that since many IoT devices are fixed function, the interconnection between an IoT network (if it has a distinct architecture) and the current Internet will happen at the application layer, not at a lower transport layer. In other words, there will not be a strong motivation to treat an IoT network as a region across which we establish end-to-end connections at the data transfer layer to devices on the current Internet.

Another set of requirements that trigger regional diversity arise in Delay/Disruption Tolerant Networks (DTNs), where regions with different sorts of latency and intermittency are connected together. i discuss DTNs below.

Requirement: performance

ALF A number of proposals address one aspect or another of performance. For example, architectures that include tools to allow a client to find the closest copy of some content are certainly addressing performance. But few of the proposals address the classic concept of performance, which is getting the protocols to transfer data faster. One proposal that focused on performance was Application Layer Framing, or ALF. However, the aspect of performance addressed in ALF was not *network* performance, but end-node *protocol processing* performance. In particular, the functional modularity of ALF was motivated in large part by a desire to reduce the number of memory copies that a processor must make when sending and receiving packets. In principle, ALF allowed the protocol stack to be implemented with as little as two copies of the data (including the application layer processing), which was seen at the time as a key factor in improving end-to-end throughput. This issue seems to have faded as a primary concern in protocol processing, but in fact it may be the case that the overhead of copying data by the various layers of the protocol stack is still a limiting factor in performance.

ALF also allowed for regional variation in architecture; in particular, the authors were considering the Internet protocols and ATM as candidate regional architectures. This degree of variation implied that packets would have to be broken into cells or incoming cells combined into packets at a regional boundary, which in turn implied a lot of per-flow state at a regional boundary. The common element of payload across all the regional architectures was a larger data unit called an Application Data Unit, or ADU. ADUs were to be transmitted as sequences of bytes, which could be fragmented or reassembled as desired. The loss of part of an ADU due to a lower layer failure caused the whole ADU to be discarded, which presumably implied a potentially large performance hit for a lost packet or cell.

Requirement: Information Centric Networking

The idea behind Information Centric Networking, or ICN, is that users do not normally want to connect across the network to a specific host, but rather to a higher-level element such as a service or a piece of content. The content or service might be replicated at many locations across the net, and the choice of which location to use is a lower level decision that is not fundamental to the user's goal. (Of course, the lower level considerations, which might include performance, availability and security, do matter to the user, and should not be ignored all together.)

TRIAD TRIAD is an example of an ICN that is largely inspired by the design of the Web. The names used for content in TRIAD are URLs, and the user requests content by sending a lookup request containing a URL. To over-simplify, TRIAD, routers contain routing tables based on URLs (or prefixes of URLs), so lookup requests can be forwarded toward a location where the content is stored. The lookup request is actually a modified TCP SYN packet, so once the lookup request reaches the location of the content, a somewhat normal TCP connection (using lower level IP-style addresses) is then established to transfer the data.

TRIAD does not actually require that *all* routers support a URL-based forwarding table. TRIAD assumes a regional structure (a connected set of Autonomous Systems or ASes, similar to today's Internet) and requires that each AS will maintain a set of routers with a URL forwarding table, and can forward a setup request to one of those routers when it is received by the AS. So most routers could function as today, with only an IP-based forwarding table. Another feature of TRIAD is a loose source-routing function, so that different addressing regions can be tied together. This scheme would allow a TRIAD Internet to reuse IP addresses within different regions, so as to avoid running out of addresses.

The key challenge with TRIAD is to design a URL-based routing system that scales to the size of the Internet and the anticipated number of content names in the future. Their proposal is a BGP-like route announcement scheme, where what is forwarded would normally only be the first and second levels of a DNS name. So the forwarding table would have to be of a size to hold all the second level DNS names, which is many millions but not billions. For a URL where the third or subsequent name components describe a piece of content not stored at the location of the second level name, that location would store an indirection binding to allow the lookup request to be redirected to the proper location.

Multiple content sources can announce the same DNS prefix, and the routing protocol would then compute paths to whichever source is closest, so the TRIAD scheme provides a form of DNS-based anycast. Further, elements that announce a DNS name can perform functions other than the simple delivery of the content. For example, an element might transform the content into a format appropriate for the requester, as it retrieves it from the original source. In this way, certain sorts of middlebox function can be supported by the TRIAD architecture. The TCP-like connection from the requesting client would be intentionally forwarded to the transform element, in contrast to a "transparent" element that imposes itself (contingent or topological receipt) into the path without the knowledge of the end point.

DONA The Data-Oriented Network Architecture (DONA) system is in many respects similar to TRIAD. A lookup message (called a "FIND" request in DONA) is sent from the requesting client, which (when it reaches a location with the content) triggers the establishment of a TCP-like connection back to the client to transfer the content. A key difference is that in DONA the DNS names are replaced by flat, self-certifying names. Content is created by a *principal*, an entity defined by a distinct public-private key pair. A name for a piece of mutable content is of the form P:L, where P is the hash of the principal's public key and L is a label unique within the namespace of P. For immutable objects, L can just be a hash of the content. Similar to TRIAD, names are propagated in a BGP-like routing system to special routers in each AS that support name-based forwarding. A principal can announce names of the form P:L (which give the location of a specific content object), or P:*, which provides the location of the principal. Again, the principal can announce these names from multiple locations, providing an anycast-like character to the lookup process.

When content is retrieved, what is actually returned is a triple of the form <data, public-key, signature>. The recipient can first verify that P is the hash of the public key, and then verify the signature (which would have been computing using the private key matching that public key). In this way, a recipient can verify the authenticity of content without needing a verified connection to the original source. Content can be cached (ignoring issues of stale or malicious cache entries with old version of the mutable data).

Because names are flat in DONA, the number of distinct routing entries will be much higher than with TRIAD. DONA proposes an important scalability assumption. The DONA Internet, similar to today's Internet, is assumed to have a somewhat hierarchical structure, with Tier-1 providers at the core, as well as the option of peering at any lower level. In today's Internet, a peripheral AS need not keep a complete forwarding table but can have a default route "toward" the core. Similarly DONA routers outside the core need not store a complete name-based forwarding table. Only in the DONA equivalent of the "default-free" region of today's Internet must a complete name-based forwarding table be stored.

The authors of the DONA scheme provide an analysis of the rate at which routers must process FIND messages and routing update messages (which they call REGISTER messages), and make a claim in their paper that the performance demands are feasible.

DONA, like TRIAD, can use the basic lookup semantics to implement a number of advanced features. These include caching, a substitute for the session setup semantics of a protocol like SIP, middlebox functionality, a publish-subscribe semantics, and client-controlled avoidance of an overloaded or mis-functioning content server. This latter scheme is important to avoid a class of malicious behavior that leads to a loss of availability, where a malicious server purports to deliver content matching the name P:L. The receiver can detect that the content is invalid, but without some way to "route around" that malicious source, there is no way to ask for a different copy. To address this problem, an extension to DONA allows the requester to ask for the "second-closest" copy, or "k-th closest", rather than the default closest copy. How the routing protocols can support this semantics is an interesting challenge.

Named Data Networking Named Data Networking, or NDN, described in more detail below, takes some of the ideas from TRIAD and DONA and pushes them into the data plane. In particular, instead of using a name-based lookup packet to trigger a TCP-like content transfer phase, NDN uses content names rather than addresses in every packet, and removes from the scheme any concept of routable lower-level addresses. The names, like TRIAD, are URL-like in format, but now every router must have a name based forwarding table. Names for content in NDN describe packets, not larger data units, and include both a URL-like element for forwarding as well as a self-certifying name for security. Interestingly, the DONA proposal also notes that names could describe chunks of data, not the complete data unit, but this is a secondary consideration for DONA.

PSIRP/PURSUIT The Publish/Subscribe Internet Routing Paradigm (PSIRP) and its successor, PURSUIT, is a different example of ICN architecture. Objects in PURSUIT are higher level, more closely related to what applications, services or users might require, rather than packets. In this respect, PURSUIT objects are similar to DONA or TRIAD objects. Each object is identified by a *rendezvous ID* (RID), and published to make it available in one or more *scopes.* Abstractly, a scope is a namespace within which an RID is unique, instantiated as a set of services (servers) across the network that can translate that name into a location. Scopes themselves are named using a form of RID (in other words scopes do not necessarily have globally unique IDs) so the name of an object is a sequence of RIDs. Objects are *published* by their creator by contacting a scope to assign an RID to the object. *Subscriptions* express interest in receiving the object. The architecture is explicit about both publish and subscribe in order to create a balance of control between source and destination. Scopes also contain a topology manager (TM), which is responsible for keeping track of where copies of the content are stored, either in a cache or at the location of the publisher. When a subscribe request is received by the scope, the RNs contact the TM, which determines how to deliver the content to the requesting subscriber.

Network of Information (Netinf) Netinf, like DONA used flat, globally unique names to identify data objects (in contrast to NDN that names packets). The security architecture of Netinf, like some other schemes, defines the name of an object to the hash of its contents. Netinf, like NDN, defines a standard format for the data object (in this case a MIME representation) so that any element in the system, not just the end node, can verify that an object corresponds to the name associated with it. The authors stress the importance of being able to validate a data object independent of where it comes from, since (like many ICN proposals) Netinf includes the management of content caching as part of the architecture.

The Netinf retrieval model is similar to DONA: a *get* message is routed using the ID of the object to a site where the object is found, at which point a transport connection is opened to deliver the object. The Netinf designers contemplate a system with regional diversity, as discussed above, and discuss two approaches to realizing this: either state is established at region boundaries as the *get* is forwarded so that subsequent transport connections can be linked together through that boundary point to deliver the object, or some sort of return source route is assembled in the *get* message to allow a response to be returned.

Netinf discusses two modes of ID-based routing. One is a system of Name Resolution Servers (NRS), in their proposal organized as a hierarchy rather than a DHT, which can forward the request toward a location where the object is stored. Their scheme seems similar in general terms to the scheme described in DONA. As well, they discuss direct routing on IDs in the routers in the system, perhaps in a local region of the network. In this respect, the scheme is similar to that of MobilityFirst, where the GNRS does not return the exact location of an object, but just the region (an AS, or network, or whatever) where the routers maintain a per-object forwarding table. Netinf refers to this sort of information as a *routing hint*, and point out that the hint need not get the request all the way to the region of the destination, but only toward it, where another NRS may have more specific information. They note that these schemes can support *peer caching*–an end-node could inform an NRS that it holds a copy of some content, or respond directly to a GET that is broadcast in some region. With respect to routing, Netinf and NDN have some similarities: both architectures allow for a range of routing/forwarding options that may get a request to a location of the content. However, Netinf allows for a query to a NRS as well as a local per-object forwarding table. The cited Neginf paper included an analysis arguing that a global NRS of object IDs is practical.

Netinf contemplates several sorts of caches. One is an *on-path* cache. This sort of cache works by what I called *contingent* delivery: if the request packet happens to encounter a node with the content, the content will be returned. Netinf also allows for *off-path* caching, where an NRS node maintains explicit knowledge of where a copy is stored, and explicitly provides the address of that node to the router forwarding the request, so that the request is explicitly delivered to the cache.

Discussion Unlike NDN, PURSUIT subscriptions and publications are persistent, with the state for that managed at the network edges. In contrast, NDN requests for data (*interest* packets) are reflected as state "inside" the network at each node. PURSUIT, like DONA or TRIAD, has a lookup phase to find the best location of the content, followed by a transfer phase that actually retrieves the content. One could imagine using TCP for this phase, although PURSUIT is defined as a free-standing architecture that need not use the Internet as a supporting architecture. The large, two-level address space of DONA is replaced in PURSUIT by the nested sequence of RIDs that define the nested set of scopes within which the rendezvous information about the object is contained. In this respect, the names are hierarchical, somewhat like TRIAD, but are in no way modeled on URLs. There is no implication that the nested names in PURSUIT have "meaning" in the same way that DNS names have meaning. The scope names and object names in PURSUIT are just IDs, and do nothing but identify which scope is being used to provide the rendezvous service. If there are "meaningful" names in PURSUIT, they will exist at a higher level.

DONA (as well as Netinf) can be seen as pushing the idea of flat identifiers to the limit (the DONA names P:L have global scope), which is a good strategy for a research project. In this respect, there is a similarity with a key component of the MobilityFirst architecture, discussed below. The problem addressed in DONA and MobilityFirst

architectures are slightly different: in DONA the challenge is to give information objects stable names, and to find them as they move. In MobilityFirst, the challenge is to track mobile devices (which could include objects on those devices) as they change their point of network attachment. These slightly different use cases will imply differences in the rates of registrations and lookups, but both schemes call for a global system that can resolve flat names with low latency. The nested scope structure of PURSUIT implies more steps to a lookup, but avoids the need for a single global registry of object names.

I noted in Chapter 3 that the Internet required but did not specify a routing protocol that could deal with Internet-scale end-node addressing. The Internet of today depends on a degree of topological aggregation of addresses (the Internet routes to blocks of addresses, not individual addresses) but the size of blocks is not specified in the architecture. Rather, it is a pragmatic response to the capabilities of current routers. Similarly, the challenge for schemes such as Dona, Netinf, NDN and MoblilityFirst is whether a routing scheme can be devised that can meet the requirements of the architecture–a scheme which in each case they require but do not specify. (In each case, the proposals include a sketch of a possible solution to give credence to the scheme, but like the Internet, the routing scheme used in practice would be expected to evolve over time.)

Requirement: architecting for change

As I will discuss in Chapter 6, there are several views as to how to design an architecture so that it survives over time. Different schemes here take different views of this objective. FII assumes that over time new architectures will be designed, and thus the goal of the higher-level FII design is to define a minimal set of constraints on these different architectures so they can interwork, thus providing a path for migration. The XIA system, discussed below, assumed that there would not be a need for that radical a replacement of the overall architecture, and assumes a common packet format, which FII explicitly rejected. XIA allows for incremental evolution of the addressing format, which has proved the most challenging aspect of the current Internet to change.

Requirement: intermittent and high-latency connectivity

The Internet and its kin were conceived in the context of immediate delivery of packets: immediate in the sense that they were suitable for interactive applications. A well-engineered region of the Internet today will often deliver packets within a factor of 2 or 3 of the latency of light. However, not all operating contexts have characteristics that can even attempt to support this class of interactive applications. The Interplanetary Internet initiative laid out the extreme form of the challenge–intermittent connectivity (perhaps as satellites come in and out of range) and latencies of seconds if not minutes.[1]

In response to these challenging requirements, a class of network called Delay/Disruption Tolerant Networks (DTNs) has been proposed. The discussion here is based on a seminal framing paper [Fall, 2003]. As described there, the core idea is that the basic forwarding model must be store-and-forward (rather than direct end-to-end forwarding) and that the unit of storage is not the packet (which might not be a uniform standard across the system, but a higher-level entity (similar to an Application Data Unit) which they call a *bundle*. The DTN architecture is based on regional variation in underlying architecture (for example classic Internet on any given planet), and is thus similar in some respects to proposals such as Metanet, Plutarch or FII. (The DTN proposal makes specific reference to Metanet.) However, DTN does not attempt to stitch together regional architectures to achieve something end-to-end that resembles the behavior of the individual regions. DTN assumes devices at region boundaries that terminate transport connections, receive and reassemble bundles, and potentially store those bundles for long periods of time until onward transfer is possible. Different transport protocols might be used in different regions, depending on whether the latencies are on-planet (where TCP would be suitable) or multi-minute.

The names for DTN bundles are of the form <region-name, entity-name>. This structure is similar to what is seen in MobilityFirst, where the region names are globally meaningful and the entity names are only routable

[1] *The interested reader may refer to http://ipnsig.org/ for background on this initiative.*

within the region. In the DTN paper, each of the names is a variable-length text string, rather than any sort of flat self-certifying ID. However, those names could be introduced into DTN without disrupting the scheme. Like many of the other schemes I have discussed here, what DTN depends on is a routing scheme that is not specified as part of the architecture, but must be realized. In this case, the routing scheme is not dealing with routing to objects–it is routing to a potentially small number of regions. The challenge for DTN routing is to deal with the complex and multi-dimensional parameters of the overall network, which may include satellite links that provide intermittent connectivity on a known schedule, or (as a terrestrial example) so-called *data mules* such as a rural bus or delivery vehicle with a wireless base station that can pick up or drop off bundles whenever it drives by. They propose a framework in which routes are composed of a set of time-dependent *contacts*, which are assembled by a routing algorithm, perhaps somewhat the way pathlets are composed.

Due to the long delays in the system, the classic end-to-end reliability model of the Internet with acknowledgements flowing back to the source will often not be practical. Instead, DTN requires that the inter-region store-and-forward elements be reliable enough to assume responsibility for the assured forwarding of bundles. In this respect, the basic communication paradigm of the DTN architecture resembles Internet email, with (presumed reliable) Mail Transfer Agents and no architected end-to-end confirmation of delivery. As well, since store-and-forward nodes will have finite storage, DTNs raise complex flow control issues.

What distinguishes DTNs from (for example) FII is not the structure of the names, but the requirements that the designs put onto the key elements–requirements that the architecture specifies but leaves to the implementer to realize, such as inter-region routing and flow control.

MobilityFirst, with its emphasis on mobile devices, also discusses the need to store objects at key points in the network if intermittent connectivity prevents them from being delivered. [[[Not sure if they store packets or ADUs. I don't know of any ADU naming in MF. Ask about this aspect of the design.]]]

Requirement: mobility

The previous discussion has hinted at most of the key requirements for an architecture that supports mobility: separation of location from identity, intermittent connectivity, variable performance characteristics, and the ability to track the location of moving devices, networks and content. Architectures differ in how they deal with theses issues. MobilityFirst assumes a low-latency Global Name Resolution Service that can look up the current location of a mobile entity while a packet is in transit. Other schemes assume a less aggressive approach to mobility, in which the sender must determine the new location of the entity and retransmit the packet. MobilityFirst allows for short-term storage of packets if onward transit is not possible. NDN assumes that routing protocols suited for different sorts of regions (such as broadcast for wireless regions) can be used as a part of dealing with mobility of content. If devices move in NDN, they will have to re-transmit any pending *interests*, but since the mobile device initiates this action, there is no need for any sort of name resolution service to track mobile devices.

Requirement: expressive power–services in the net

This class of architecture has the goal of invoking service "in" the network as packets flow from sender to receiver. The proposals in this class directly address the question of the expressive power of an architecture, the balance of control held by the sender and the receiver, and the different sorts of delivery modes and PHB arguments (see Chapter 4). I look at five examples of this class of architecture: TRIAD, i3, DOA, Nebula and ANTS.

TRIAD The first of the architectures I discussed that support this function is TRIAD. TRIAD establishes a connection to a abstract entity (typically a item of information) using a setup packet with a URL in the initial packet. As I described in Section 5.2, the provider of content advertises that content by inserting the top-level components of its URL into the TRIAD name-based routing function. Once this mechanism is in place, it can be used to route any sort of session initiation request to a location advertised by the provider, so the scheme is a somewhat general form of intentional delivery to a service point. The URL provides an explicit parameter to whatever PHB that service node implements. However, the granularity of the TRIAD routing is only the second

(or perhaps third) level of DNS name in the URL, so whatever service node is referenced by that URL will end up being invoked by a large number of URLs.

Internet Indirection Infrastructure (i3) In i3, receivers express an interest in receiving a packet by creating a ID, and announcing to the i3 system the pair (ID, addr), where addr is (for example) an IP address. (See below for what "announcing" actually means.) The term used in i3 to describe what the receiver is creating by this announcement is a *trigger*. After this, the receiver might then create an entry in a DNS-like system that mapped some user-friendly name to that ID. The sender would look up that name in this i3 DNS system, which would return the ID to the sender. The sender would then initiate communication by sending a packet addressed to the ID. i3 associates the ID with a node in the i3 system where the trigger is stored. In the simple case, once the packet has been delivered to this node, the trigger provides the actual address of the receiver, and the packet is forwarded onward to the receiver using that address.

i3 actually supports a more complex use of IDs, to allow both the sender and the receiver to forward the packet through a sequence of servers (PHBs) on the way from the sender to the receiver. A receiver (or a third party) can include in a trigger a *sequence* of items, not just a single address. This *trigger sequence* can include both IDs and addresses. As well, the sender can include in the packet a sequence of IDs, rather than a single ID (that might have been retrieved from the i3 DNS). These IDs represent the services (PHBs) that the sender wishes to have performed on the packet. The sender puts this sequence of IDs in the packet and sends it to the node associated with the first ID (the location where the trigger is stored). At that point, the service looks up the trigger, retrieves the trigger sequence, prepends it to the sequence in the packet, and then forwards it to the first item on the modified list, which could be another ID or an address.

If that first item is an address, the packet is sent directly to that node using the lower level mechanism. That site might be a location where a PHB is implemented. After the PHB is executed, that node will remove its address from the front of the sequence and once again forward the packet to the item now at the head of the sequence. The execution order of the different services identified by the various triggers is potentially complex, although if the sender prepends its IDs in front of the ID from the receiver, the execution should proceed in a sensible order. (If the sender does something more complex to the sequence, such as putting some of its IDs after the ID of the receiver, this might be a signal that the interests of the sender and receiver are not aligned, and something like an attack is contemplated.)

In i3, the nodes where triggers are stored are organized as a DHT. The first packet of a sequence is sent by the originator to a nearby node in the DHT, which forwards it according to the distributed algorithm of the DHT (the prototype used CHORD) until the packet reaches the node responsible for that ID, which is where the triggers are stored. After the first packet is processed, the mechanisms of i3 provide to the sender the IP address of the node in the DHT where the trigger is stored, so after the first packet of the sequence, the sender can forward the subsequent packets directly to the correct DHT node. However, every packet follows the triangle path from sender to DHT node to receiver. If there is a sequence of IDs in the path, the packet will flow through a number of DHT nodes. This process could be highly inefficient, depending on which DHT node the ID maps to, so the authors suggest an optimization where the receiver create a session ID (they call this a *private* ID), which has been carefully selected to hash to a DHT node near the receiver.

There are a number of enhancements in i3, including the idea of longest prefix match on IDs, to allow selective delivery based on the location of the sender. But the performance of i3 will be fundamentally determined by the routing required to send the packets to the nodes in the DHT where the triggers are located.

Delegation Oriented Architecture (DOA) DOA has much in common with i3 (including a joint author) but simplifies and makes more efficient the forwarding mechanism. Like i3, forwarding in DOA is specified by flat, unique IDs, in this case associated with a network entity (as opposed to a service in i3). As with i3, the conversion from an ID to a network-level address (e.g., an IP address) is done using a DHT. Similar to i3, what is associated in

the DHT with an ID can be either an IP address or one or more IDs. However, in contrast to i3, where the packets are forwarded through the node in the DHT hosting the ID, in DOA the DHT is used to retrieve the matching data. If the returned data is an IP address, the packet can then be sent directly to that location. If what is returned is a further sequence of IDs, the lookup process is repeated by the sender until it yields an IP address. The sequence of IDs is included in the header of the packet, so as the packet proceeds through the network from service point to service point, further lookups of the IDs will occur.

What the DHT actually returns is called an *e-record*. The security architecture of DOA requires that the ID is the hash of a public key belonging to the entity creating the e-record. The e-record includes the ID, the target (the IP address or sequence of IDs) an IP hint that provides improved efficiency (see the paper), a TTL after which the e-record must be discarded, the public key of the creator, and a signature of the e-record made using the private key associated with that public key. So a recipient holding an ID can confirm that the e-record was made by the party that holds the public key from which the ID is derived. DOA assumes but does not specify some higher level naming mechanisms that would map a user-friendly name (e.g., a URL) to an ID.

Since e-records can be verified independently of where they are obtained, there are useful optimizations that can improve the performance of DOA. For example, when a client (which would be identified by its own ID) contacts a server, it can include the e-record for its ID in the data, thus side-stepping the necessity for the server to query the DHT to return a packet to the client.

Both the sender and the receiver have some control over how a packet is forwarded in DOA. The sender can prepend to the data returned from the DHT a sequence of IDs representing the needs of the sender. So the packet will first flow to services specified by the sender, then to services specified by the receiver. I believe that the expressive power of i3 and DOA to define an execution order for PHBs is equivalent, but there may be creative ways of using the two architectures, perhaps ways not contemplated by their creators, that could create very complex execution orders. Because in DOA the DHT is used as a query mechanism rather than a forwarding mechanism, it is probably simpler for a sender to determine what the execution order of the PHBs will be. A sender could look up the first ID, look at the returned data, recursively look up the sequence of IDs in that data, and eventually construct the final sequence of service points through which the packet will flow. Of course, an intermediate node could rewrite the header or otherwise misdirect the packet, but since all the services in the path are specified by either the sender or the receiver , if a service mis-directs traffic it is a signal that the service is not trustworthy. Normal execution would not involve the invocation of a service unless either the sender or the receiver requested it. (In DOA all delivery is intentional–there is no contingent delivery in DOA at the level of services specified by IDs.)

DOA must deal with the situation where the interests of the sender and receiver are not aligned. If the sender is an attacker, he can be expected to manipulate the data that comes back from the DHT before sending the packet. He might omit some of the IDs and attempt to send a packet directly to the final ID in the sequence, thus bypassing some services specified by the receiver (presumably these might be protection services rather than functional services). DOA does not provide a clean way to map adverse interests into a coerced delivery mode, where the sender must send the data to the IDs in turn. There are two options in DOA that can deal with this situation. If the later IDs map to a different address space from the sender, then the sender cannot address the final receiver directly. (The coercion in this case would be a form of topological delivery, using a DOA node that is an address translator to coerce the sender to traverse it.) DOA also specifies a way for an intermediate node to sign a packet using a key shared between the intermediate node and the receiver, so that the receiver can check that the packet actually passed through that node. This scheme gets complex if multiple intermediate nodes need to sign the packet, but can insure that the packet has been properly processed. However, it cannot prevent a malicious sender from sending packets directly to the address associated with the final ID, so DOA cannot prevent DDoS attacks based on simple flooding. The paper notes that some other mechanism is required for this purpose.

Nebula Nebula, discussed in more detail below, defined a much more robust mechanism to control routing of packets through services, in order to prevent (among other things) the problem in DOA of an untrustworthy node that might mis-direct a packet, skip a processing step, and so on. In Nebula, the sender and receiver can each specify the set of services they would like to see in the path between them, but the query to the DHT to retrieve a set of IDs is replaced by a query to the NVENT control plane, where there is an agent representing the interest of every such server, and all such servers must consent to serve the packet before it can be sent. NVENT computes, with some clever cryptography, a Proof of Consent (POC) that is returned to the sender. The POC serves as a form of source route, and only by putting this POC into the packet can the packet be successfully forwarded; Nebula is "deny by default". As the packet traverses the sequence of services, the POC is transformed into a Proof of Path (POP), which provides cryptographic evidence that all the service nodes have been traversed in the correct order. These mechanisms address the issue briefly mentioned in the DOA paper where it is suggested that a cryptographic signing be used to confirm that the packet has been processed by an intermediate node.

ANTS ANTS is an example of Active Networks, as initially proposed in [Tennenhouse and Wetherall, 1996]. The concept proposed there was that packets carry small programs that are executed by "active routers" when the packet arrives. The program in the packet determines the packet treatment (for example, exactly how the packet is forwarded). In other words, the packet rather than the router specifies the PHB applied to the packet. ANTS is a reduction to practice of this idea. In ANTS, the program is not actually in the packet; rather, the name of the program (a cryptographic hash of the code) is in the packet, along with initial invocation parameters (what they call the type-dependent header fields). The type-dependent header fields are the explicit arguments to the named PHB; the packet is intentionally delivered to the node by the address in the packet. A packet carrying this information is called a *capsule*.

The idea of active code in a packet raises a number of thorny challenges related to security, such as protecting the router from malicious code and protecting flows from each other. ANTS takes a pragmatic approach to these problems, and requires that the code be audited by a trusted party before being signed and deployed. As well, the code execution is *sandboxed*–placed in a constraining environment that limits what the code can do. The code is distributed to each active router along the path using a clever trick. Part of the header is the IP address of the previous active node that processed this packet. If this router does not have the code, it asks that router to send it. That previous router presumably has the code because it just processed the packet itself. In this way, the first packet in a flow proceeds from active router to active router, sort of dragging the required code behind it. Code, once retrieved, is cached for some period so subsequent packets can be processed efficiently.

When a capsule is forwarded, the type-dependent header fields can be modified, in order to modify the execution of the program at subsequent nodes. The active code can also store state in the router, which permits a range of sophisticated PHBs. However, the code cannot perform arbitrary transformations on the router state. The code is restricted to calling a set of specified low-level router functions; the paper describes the active code as being "glue code" that composes these functions in ways that implement the new service. These low-level router functions (what they call the ANTS Application Program Interface or API) now become a part of the ANTS architecture. Ideally, all active routers would provide these same functions, so they end up being an example of "Issues on which we must all agree for the system to function", which was one of my criteria in Chapter 1 for classifying something as architecture. The ANTS paper identifies the router functions, the structure of capsules, the address formats, the code distribution scheme, and their use of a TimeToLive field to limit packet propagation as the elements of the ANTS architecture.

One cannot write arbitrary PHBs in the context of the ANTS system. The goal of the design was to allow different sorts of forwarding algorithms to be implemented for different classes of packets. In this respect, ANTS has something in common with XIA, which has different routing schemes for different classes of packets. XIA actually has more expressive power, since in ANTS the active code cannot implement a different, long-lived routing protocol among all the nodes–the state variables in the router associated with a given program are

short-lived. In XIA, there can be a separate routing protocol running to support each class of identifier. But there is a similarity between an XIA identifier of a particular type and an ANTS program name. XIA does not discuss how the code for a new identifier class is deployed–this is left as a exercise in network management. XIA allows for a series of identifiers to be in a packet header–it would be interesting to work out whether an ANTS-like scheme could allow more than one program invocation to happen in processing a capsule. The PLAN scheme also allows a incoming packet to select from a range of routing functions to derive how it is to be forwarded.

There are schemes similar in objective to ANTS. The PAN proposal [Nygren et al., 1999] uses a language that permits high-performance execution to argue that this style of active networking is practical. The PLAN proposal [Hicks et al., 1999] similarly proposes a new language specifically designed for this sort of active code. ANTS used Java as its programming language, which had some performance implications.

Requirement: shaping industry structure

One of the requirements I listed for an architecture was its economic viability and the industry structure that its structure induced. Few of the academic proposal for architecture directly address this goal, but architects from industry have been quick to understand and respond to this requirement. In 1992, when the government, under the urging of then Senator Gore, announced the vision of the National Information Infrastructure, industry was quick to respond with what it considered its critical concern, which was the modularity of the design and the resulting implications for industry structure. They understood that since the private sector was to deploy the NII, the structure of the industry was key to its success. Two different groups developed a response to the call for an NII.

CSPP The CSPP, now called the Technology CEO Council, responded to the concept of the NII with a high-level vision document, perhaps consistent with drafting by a group of CEOs. It does not talk about architecture, but contains a requirements list that is in some respects similar to what I have discussed here (access, first amendment, privacy, security, confidentiality, affordability, intellectual property (protection of), new technologies, interoperability, competition and carrier liability (freedom from)).

XIWT The Cross-Industry Working Team, convened by Robert Kahn at the Corporation for National Research Initiatives, dug deeper into the potential architecture of an NII. They described a *functional services* framework and a *reference architecture* model. After their own list of requirements (sharability, ubiquity, integrity, ease of use, cost effectiveness, standards and openness), this group focused on the *critical interfaces* that would define the modularity of the NII. They emphasized two interfaces that were not well-defined in the Internet: the Network-Network interface (the data and control interface that defines how ISPs interconnect) and the Network Service Control Point to Network interface, which would allow for intelligent control of the network, perhaps supporting the ability of third parties to control the behavior of the network. This latter interface is somewhat reminiscent of the intelligent network interface being contemplated by the telephone system to allow the control of advanced services, and is a hint that the members of the XIWT were not totally committed to the idea of the "dumb, transparent" network. Perhaps in the proposal for a Network Service Control Point we see an early glimmer of Software Defined Networking.

5.3 Active Networks and virtualization

The ANTS scheme mentioned above is an example of Active Networks, a concept proposed in [Tennenhouse and Wetherall, 1996]. Active Networks is a point of view about design, not a specific architecture, and a number of different approaches, with different architectural implications, have been proposed under the banner of Active Networks. ANTS, mentioned above, is only one of them. The general idea is that a router should not be a statically programmed device with function fixed by the vendor, but a device in which new code can be added at run-time to implement new functions. Various examples of Active Networks can be classified along

several dimensions: the purpose of the active code that is added to the router, how the active code is installed, what set of packets the code is applied to and whether the headers of the packets need to be modified to exploit the capabilities of the active code. Depending on the answers to these criteria, the different schemes can be described as more or less "architectural", depending on the degree to which they define and depend on "Issues on which we must all agree for the system to function".

Delivering the code One of the original concepts in Active Networks was that active code was delivered in a packet and was executed when that packet arrived as part of forwarding that packet. As the concept evolved, alternatives emerged in which the code is delivered and installed to be executed on subsequent packets, either packets that are part of a given flow, or in some cases on all incoming packets. Thus, the Active Network concept includes both programmability at the packet level and extensibility of the node. An example of a scheme that focused only on router extensibility is Router Plugins [Decasper et al., 1998]. The Smart Packet system [Schwartz et al., 1999] puts a very compact code element into the packet that is executed when the packet arrives at a given node. The PLANet scheme [Hicks et al., 1999] supported both programmability of the packet and the ability to extend the router functionality. It thus exploits two different languages, Packet Language for Active Networks (PLAN), which is carried in the active packets, and OCaml for router extensions. The ANTS scheme did not put the program in the packet, but had each router retrieve the code as needed. This removed the requirement that the code fit into the packet. ANTS packets specified which code was to be executed when they arrived, so execution of the code was limited to the flows of packets that requested the treatment.

The function of the active code For a variety of reasons, most Active Network proposals carefully limit what the active code can do. One reason is security–code that can make arbitrary modifications to the router state would seem to pose an unacceptable risk of malicious perversion of router function. Schemes that allow the dynamic installation of highly functional code have to restrict the ability to install such code to trusted network managers. Second, especially if the code is delivered to the router in a single packet to control the processing of that packet, the code has to be compact. For these reasons, many Active Network schemes provide a set of primitives that the active code can call, and the active code (several schemes call it "glue code") composes these primitives to achieve the desired function. Different schemes focus on different functional objectives, and characterize these objectives by the primitives the scheme defines. The ANTS scheme described above has the objective of flexible forwarding of packets, and the primitives support that goal. In contrast, the Smart Packet scheme has the goal of supporting sophisticated and flexible network management, and thus provides primitives to read management data from the router. The PLANet system includes both flexible forwarding and network diagnosis as functional objectives. The paper on PLAN describes how PLAN active code can be used to implement diagnostics similar to *ping* and *traceroute*.

Program isolation and security The concept of active code raised many challenging security issues, perhaps most obviously how to keep a piece of active code from interfering with flows that do not want to be affected by the code. "Pure" active schemes address this problem by having the code delivered in the packet and executed on behalf of that packet. In other words, the role of any piece of code is limited to the packet that carried it. ANTS addresses this problem by having the packets name the code that is to be executed on behalf of that packet. PLANet describes an option at the other extreme, where one source can send a packet that installs a router extension that changes the queuing discipline of the router for *all* flows through the router. The Active Bridge project [Alexander et al., 1997] describes the downloading over the network of active code (which they call *switchlets*) that bootstrap a box with ethernet bridge functionality, starting from a machine that only supports loading dynamic code. This code, once loaded, is used to handle *all* packets passing through the switch, so clearly the ability to install this sort of code must be restricted to the trusted network manager. The Active Bridge system also uses a strongly typed language (Caml) to provide isolation among different modules, so the system achieved

security both through restricting which actors can download code as well as strong, language-based mechanisms to protect and isolate different code elements from each other.

Network virtualization The Active Network concept is that routers can be dynamically augmented with enhanced functionality, and different packet streams can potentially receive different treatment based on the execution of different functionality that has been downloaded in one way or another. One variant of this concept is that a router is just a specialized computer that supports *virtual routers* (by analogy to other sorts of device virtualization). It permits different versions of software implementing router functionality to be dynamically loaded into into different *slices* of the physical device. This allows one physical device to support several network architectures (of the sort I have been discussing here) at the same time, so that they can co-exist on the same infrastructure. Multiple routers connected together and running code supporting the same network would then be a *virtual network*.

Virtual networks have something in common with active networks, but there is a difference in emphasis. The idea of virtual networks has emerged in large part from a desire to allow multiple network architectures to co-exist on a common physical infrastructure, as opposed to the goal of having different packet streams within a given architecture receive different treatment, or to perform advanced network diagnostics. Virtual networks require the installation on the physical router of code that performs all of the network functions associated with an architecture (forwarding, routing, network management and control, and so on), which means that virtualization code is a complete protocol implementation, not "glue code" that sits on top of functional building blocks associated with a given architecture. For this reason, the isolation among the different code modules has to occur at a lower level of abstraction, and the installation of the code is almost certainly much more tightly controlled than if a packet carries its own processing code with it. So active networking tends to focus on the specification of a language for defining functional enhancements, clever schemes for code dissemination and the expressive power created by different code modules and packet headers, while network virtualization tends to focus on low-level schemes for code isolation. Network virtualization also depends on schemes to disseminate code, but they tend to have a very different character than some of the active network schemes, since they must depend on some basic forwarding architecture to distribute code that then instantiates a different architecture.

One of the earliest virtualization schemes that I know of is Tempest [van der Merwe et al., 1998], which does not focus on diverse network architectures (all of the virtual networks are based on the ATM architecture) but on diverse control schemes for the same data forwarding mechanism. That paper, from 1998, introduces the term *virtual network*, perhaps for the first time. The idea of a virtualized router, and building virtual networks by configuring connected virtual routers, is well-understood, and in fact commercial routers today are in some respects virtualized, in that they can run multiple copies of the forwarding software at the same time, although it is the same software. In other words, today's routers will allow an operator to build multiple networks out of one set of physical elements, but they all run the Internet Protocol. Building multiple IP-based networks is an important capability in the commercial world, in that it allows Internet Service Providers to provision private IP networks for customers (for example, large corporations) that run their own private networks.

In the context of the Future Internet Architecture program, a virtualized router was proposed early on as a way to allow the research community to deploy multiple experimental architectures on one common platform [Anderson et al., 2005]. This idea, put forward in the early stages of the FIA program, was in part the motivation for the development of the Global Environment for Network Innovations (GENI) project, an NSF-funded experimental network platform.[2]

Architectural considerations Different Active Network schemes (and network virtualization schemes) have different architectural implications. Schemes that carry executable code in packets must standardize a number of conventions, which will become part of the architecture. They have to standardize the programming language,

[2] *For information on GENI, see https://www.geni.net/.*

how programs are embedded in packets and how they are executed, and what functions that code can call. They may depend on some underlying forwarding architecture, such as IP. These schemes will specify many aspects of the packet header, including the parameters that the packet carries that are handed to the code when it is invoked. In contrast, schemes that more resemble network virtualization try to minimize what is standardized (or made a part of the architecture). They have to specify how code is written and installed into routers, but the only requirement they impose on the packet header is some simple identifier that the virtualized router can use to dispatch the packet to the correct virtual router (or *slice*, or *switchlet*, depending on which scheme is being discussed).

Most of the Active Network schemes use *intentional* delivery of packets to the smart router where the active code is to be delivered. PLANet and ANTS fall in this category. In contrast, the Smart Packet scheme included the option of contingent execution of the in-packet code, so all smart routers along the path had to examine the packet to see whether it required evaluation at that site. The Smart Packet scheme encoded this information as an IP option. The NetScript system [Yemini and da Silva, 1996] has some of the flavor of both active nets and virtual networks. The NetScript system allows the network operator to install very low-level code (essentially different architectures), but emphasized the use of a crafted programming language to enhance secure execution.

In general terms, Active Networks tend to shift the architectural focus away from exactly how packets are forwarded, and rather to how code is installed and run. They shift attention from the architecture of the data plane, which becomes more plastic, to the architecture of the management and control interfaces that allow the network to be configured.

5.4 The Future Internet Architecture project

There were several different architectural proposals that were developed as a part of the NSF Future Internet Architecture program, some of which I mentioned earlier. I describe them below.

Expressive Internet Architecture (XIA)

The emphasis of the XIA scheme is on expressive addressing in packets, to allow the network to use a variety of means to deliver the packet to the intended destination, and to provide a range of services in the network. The rich addressing/forwarding mechanisms in XIA allow a packet to carry several forms of addresses at once. For example, they can carry the content id (a CID) of a desired piece of content, but as well the ID of a server hosting that content (a SID), or the host where the service is located (a HID) or an administrative domain in which the CID is known (an AD). This richness is described as expressing *intent*, and the other addresses allow various forms of fallback forwarding. This flexibility allows the end-host to select from a richer set of network services. It also should contribute to the longevity of the design, as it would permit a more incremental migration to a new sort of XID than the current migration from IPv4 to IPv6.

Some specific features of the XIA scheme are:

- The various IDs, collectively called XIDs, are specified to be self-certifying. For example, they may be the hash of a public key, or the hash of the content to which they refer. This design allows the end-points (e.g. the applications or perhaps the actual human users) to confirm that the action they attempted has completed correctly: that they connected to the host they intended, got the content they intended, and so on. Put otherwise, these mechanisms transform a wide range of attacks into detected failures.
- XIA gives the end-point options for control to bypass or route around points of failure in the network.
- The SCION mechanisms (part of XIA) provide a structure to break up the overall network into what are called trust domains, which allow a variety of end-point controls over routing.

XIDs provide a means to confirm that the correct action occurred once the end points have the correct XIDs. However, most network operations start with "higher level" names that describe what is desired, such as URLs,

email addresses, and the like. Since different applications may involve different sorts of high-level names, the XIA architecture does not define how these names should be converted to XIDs in a trustworthy way. The XIA architecture gives requirements as to what the application and its supporting services must do, but does not dictate a mandatory way of doing it.

MobilityFirst (MF)

The MobilityFirst architecture is motivated by the desire to deal with issues raised by mobile end-nodes–in particular movement of devices from one network access point to another, and transient outages when devices become unreachable. In this architecture, naming/addressing is done at two levels. At a higher level, a number of name services similar to the DNS, which they call Naming Services (NSs), map from a host, service, sensor, content or context (a context is a set of things that match some criteria) to a flat ID, a GUID. At a lower level, there is a service, the Global Name Resolution Service or GNRS, that maps from a GUID to its current location, which is a Network Address, or NA. A network address has a network part NA:N and a port part NA:P.

The essential idea behind the MobilityFirst design is that both the destination GUID and the destination NA are included in the header of a packet. This allows rapid forwarding based on the NA:N, but also allows elements in the network to deal with mobility and redirection by making dynamic queries of the GNRS as the data is moving through the network. If the NA included in the packet by the source is not the current location of the destination (e.g. the network N cannot resolve the GUID), routers in the network can attempt to look up a new NA. The design also has the ability to store data in transit at intermediate points if the destination is temporarily unavailable due to wireless connectivity issues. When data is stored, it is identified by its GUID, until a new NA can be determined for the destination.

To enhance security, both the GUID and the NA are public keys, so anyone having a GUID or a NA can confirm that the binding is valid. The namespace of NAs is thus flat. Their design assumption is that there might be as many NA:N values as there are routing table entries today.

The piece of mechanism that must be designed to make MobilityFirst realistic is the GNRS, which must be able to map a large set of GUIDs (perhaps 100B) to the corresponding NA in "real time", as the packet is in transit.

Named Data Networking (NDN)

The NDN architecture is distinctly different from the current approaches to addressing and forwarding. Instead of sending a packet to a host, in NDN one sends a packet addressed to a piece of information, and gets the information in a return packet. In NDN, there are two sorts of packets, *interest* and *data*. An interest packet is sent to request some named content, and the data packet returns that named content. In neither case is there a "host" address in the packet, only the name of the desired information.

An interest packet contains the name of the content being requested. A data packet contains the name of the content, the data itself, and a signature, which confirms the contents, as described below.

Names of content are hierarchical, and begin with the authoritative owner of the content, followed by the name of the specific content. Any owner/creator of content has a public/private key pair, and uses the private key to produce the signature. Thus, anyone with the public key of the owner can verify the data packet: in particular the integrity of the content and the binding to the name.

A distributed mechanism will allow nodes to build up a catalog of public keys for different owners, a key certification graph. In this way, routers can learn public keys, which will allow them to validate packets (as appropriate) as they forward them. (In particular, forged data packets that claim to match an interest can be detected in the net, not just at the final end-point.)

The name of data also describes its location. When information moves to a new location, there is a variation of a data packet called a link that encapsulates a content packet and is signed by the operator of the current location. This signature allows anyone with the public key of the current location to verify that the location named in the packet is the actual sender of this packet.

It is assumed that on top of this mechanism there will be a variety of search tools, content providers and so on

that, among other things, provide for translation between other sorts of names and queries and the specific names used by NDN.

A key technical aspect of NDN is that when an interest packet is routed toward the location of the content, a copy of the interest is kept at each router. In NDN, there is thus per-packet state in each router along the path followed by the interest. The copy of the interest records the router port from which the interest came, as well as the name of the content being requested. The interest packet itself does not carry any "source address" from where the interest originated: this information is recorded in the per-packet state in all the routers along the path back to the original requestor.

When a router forwards a data packet, it has the option of keeping a cached copy for some period of time. This cached copy can be used to satisfy a request, rather than having to fetch the content from the original location. This mechanism allows for the efficient delivery of popular content.

Nebula

Nebula is concerned with the implications of cloud computing on a future Internet architecture. The Nebula architecture offers the application access to and control over a much richer set of services than are available in the Internet. The responsibilities of the network are highly reliable availability, predictable service quality, and assuring that the requirements (policies) of all the relevant actors are taken into account as traffic is routed across the network. The relevant actors include networks themselves, and as well higher-level service elements.

An example will help to illustrate what is meant by that last responsibility. Nebula, like essentially all architecture proposals, assumes that the network is made up of regions that are separately built and operated, often by private-sector providers (like today's ISPs). These providers will have policies governing which sorts of traffic (e.g. for which classes of senders and receivers) they will carry, and so on. Today, the only tools that can be used to express these policies are the expressive power of BGP, which may be very limiting. In addition, the application may want to control the routing of traffic by directing it to higher-level service elements. A simple example might be a "packet scrubber" that tries to detect and remove malicious traffic, or a higher-level processing element such as a virus detector in email. A service might wish to assert that it will only receive packets that have first gone through the scrubber, even though the scrubber is not directly adjacent to the service. In Nebula, the network itself can enforce this sort of routing policy.

To do this, the Nebula architecture has two relevant parts: a data plane (NDP) that can enforce arbitrary forwarding policies, and a distributed control plane (NVENT) that can compute these policies. Every actor in the network will have an agent in the NVENT layer, and these layers run a distributed algorithm to construct a set of forwarding rules (a policy) for any requested transfer. While the control plane is still under development, there is a specific proposal for the data plane, and a claim that it can enforce arbitrary policies with respect to valid routes through sequences of actors.

Nebula is not a pure datagram network–to send a packet the NVENT policy mechanisms must first compute and return to the sender a string of information that will authorize the data plane to forward the packet. However, these routes can be computed in advance and cached. NDP is "deny by default"; without this NVENT information to put in the packet, it will not be forwarded. What is returned by NVENT is a "proof of consent" (POC), which is a cryptographically signed sequence of values that (for each step in the processing/forwarding) encode all the previous steps that must have processed the packet before this actor receives it. Agents in NVENT representing all the regions must cooperate to construct this POC. Clever use of crypto and XOR allow this to be coded in a way that is linear in the number of actors. As the packet is forwarded, each actor that processes the packet computes, using a similar set of methods, a "proof of path" (POP), which allows each subsequent actor to confirm that the previous step actually did process the packet. Thus, by comparing at each stage the POP at that point with the POC that was computed by NVENT for that stage, the NDP can confirm, for each actor in the path, that the packet has already passed through all the required previous actors. For a detailed explanation of how this mechanism works, see the paper that describes ICING [Naous et al., 2011].

ChoiceNet (CN)

In contrast to the other FIA projects, which describe a specific forwarding mechanism (e.g. the data plane), ChoiceNet is focused at a higher level: the control plane and what they call the *economy* plane. The assumption is that the data plane (which might be implemented using one of the other FIA proposals such as Nebula) will provide alternatives or choices among services: such as IPv4, IPv6, different paths with different qualities, or other services in the network. For example, a user might choose to pay more to get a higher quality movie. The goal is to make these options explicit and allow the user to pick among them.

The assumption is that the network will have a connection setup phase, and the user will express his choices at that time. The setup phase is implemented in the control plane, where component services can be selected, or composed to make new services. The result will be implemented in the data plane.

A way to conceive of the economy plane is to think of an app store for services. Different service offerings would be advertised, there would be rating systems, and so on. The user would make simple choices, which would translate into actual services by the composition of elements in the control plane.

Introspection or verification is an important part of a control plane. Did the user get what he paid for? ChoiceNet includes components that measure what is going on to verify what has been provided. More specifically, an overall service may be composed of many parts. They all get a share of money, but should also get the correct share of blame for failure. So ChoiceNet will provide some external verification as part of any service. Service proofs and payment verification are exchanged between data and control plane.

The term "user" in this discussion might be an actual person, or a software agent, or expert (human agency) like a sysadmin setting up service for user.

One challenge for ChoiceNet has to do with whether the users can realistically make these choices, whether the offered services (in the service app store) will be well-enough specified that the user will not be misled, and so on.

Some comparisons of the FIA projects

In general, all the schemes that specify a data plane share a feature that distinguishes them from the current Internet: a two-step rather than one-step name-to-address resolution scheme. In the Internet, a high level name (e.g. a URL) is mapped to an IP address by the DNS. This happens in one step. The IP address is being used as both an identifier for the end-point and its location. All of these schemes have a separate identity and location schemes, and separate mechanisms for mapping from name to identity and from identity to location, except for NDN, which has effectively eliminated any concept of a location. Most of the schemes have a way to assign an identity to things other than physical end-nodes, including services and content.

In contrast to the current Internet, which uses IP addresses as a weak form of identity for end-nodes, all of these schemes implement the idea that the identifier of an end-point entity, whether a host, a service, a piece of content or the like should be "self-authenticating". Mechanically, this is done by making the identifier a hash of a public key or the content. Assuming that the entity holds the private key, a challenge-response exchange can confirm to each end that the other end is as expected. This check prevents many sorts of attacks in the network, including DNS poisoning, packet mis-direction, and so on, from being successful.

However, detecting a failure or an attack is not enough to assure successful operation–all it can do is give a clean signal of failure. To provide successful operation in the face of these sorts of attacks, two more functions are required: first a means to detect where in the network the failure or attack is happening, and a means to avoid or "route around" this region. I discuss this framing in more detail in Chapter 8 Many of these schemes contain mechanisms to try to isolate the region of a failure, and many of them give the end-point control over the route selection to some degree. This choice reflects a preference for a basic design principle of the current Internet: since the network is not aware of what applications are trying to do, the network cannot detect when a failure has occurred. Only the end-points of the communication, which are aware of the desired application semantics, can detect problems and attempt to restore service.

The projects are rather different with respect to the range of services that they provide to the higher layers.

- Nebula and ChoiceNet: these designs assume that service building blocks in the network can be composed to present a rich selection of end-to-end services to the applications.

- XIA and MF: these designs provide a small number of service classes, corresponding to different classes of IDs–for example content, services and hosts. Each of these classes would correspond to a forwarding behavior in each router. MobilityFirst also allows for additional functions to be installed on routers in the path. MF does not support per flow QoS.

- NDN: this design implements a single, general service that returns a set of bits associated with a name. It allows for variation in service quality (e.g. QoS) using a field in the packet similar to the IP header of today. The only way that an application could embed service elements into the path from a client to the content would seem to be to use some sort of "trick", and create a URL in a request packet (what NDN calls an *interest*) that names an intermediate service as its target and then embeds the name of the desired content into the body of the URL.[3]

One way to understand these distinctions is that if the set of anticipated service classes is limited and specified (as with XIA) the relationship between the provider behavior (or a router PHB) and the resulting end-to-end service can be defined as part of the specification of the service class. On the other hand, if the set of anticipated services is open-ended (as the example of the HIPAA-compliant path used by Nebula, or a path that avoids a particular region of the world), the composition of the service from component parts must include end-point control over the path, and a more complex and sophisticated composition algorithm, which implies a separate control plane. All these schemes presume that there will be private sector actors, similar to the ISPs of today, that provision, control and operate regions of the network.

In general, these architectures give these ISPs a larger role in the operation of the network.

- NDN: they are responsible for the dynamic caching of packets of data, validating the legitimacy of the data, and so on.

- XIA: they provide a range of services (tied to types of XIDs) that can include content caching, multicast, anycast to replicas of service or content, and so on.

- Nebula: they provide a validation that packets have followed the path that was generated by the data plane.

- MobilityFirst: like XIA, ISPs provide a range of services; they also host third party computing services on their infrastructure and provide mobility-specific services such as short-term caching, redirection and the like. Collectively, they implement the core function of binding name to location, the GNRS.

- ChoiceNet: the data plane is not specified in ChoiceNet, but it must provide a set of interfaces to the control plane, through which the data plane can be configured to deliver services. Enhanced services, and the ability for the user to select them, is the central point of ChoiceNet

With respect to routing, all these schemes take the view that the network is build of regions (like the autonomous systems or ASes of today) that are separately managed and deployed. Nebula and ChoiceNet give the end-points control over routing at the level of picking the series of ASs to be used, but give each AS control over its internal routing. XIA and MobilityFirst assume a rather traditional two-level routing scheme, as does NDN, but routing in NDN has a very different flavor, since it is identifiers and not locations that drive the forwarding decisions.

Essentially all these schemes try to avoid the problems that have arisen from the hierarchical nature of DNS, and its dominant role as a naming service, by allowing multiple naming services to co-exist. The goal is to avoid a

[3] *One would have to consider if this "trick" of putting one URL inside another would confound the security architecture of NDN, where the names are used as a basis to relate certificates to content.*

single "root of trust". However, this approach raises its own set of issues, which in general have not yet been resolved.

I talked in Chapter 1 about the aspect of architecture that I called *functional dependencies*. The architecture should identify and make clear which operations depend on others, and what services have to be up and running for the basic function of the network (packet forwarding) to operate successfully. I noted that the Internet reflects a preference for a minimum set of functional dependencies. Several of the schemes I have described here have a richer set of functional dependencies.

- XIA: XIA is somewhat similar to the current Internet in its functional dependencies. Routers have a somewhat more complex task, since they have to compute more than one set of routes for different sorts of identifiers. It, like the Internet, presumes some sort of higher-level name service like the DNS.
- Nebula: Nebula depends on a presumably complex control plane (NVENT) that computes and authorizes routes. If the distributed control plane malfunctions, there is no way to send packets, since Nebula is "deny access by default".
- MobilityFirst: In MobilityFirst, forwarding depends on the GNRS. The GNRS is a complex, global system that must map from flat ID to network ID in real time. The robustness of this system is critical to the operation of MobilityFirst.
- NDN: NDN depends on a routing scheme that provides routes for names, not numbers. However, it does not depend on anything else. It shares with the Internet the goal of minimal functional dependencies. In order for a node on NDN to request information (send an interest packet) it is necessary for that node to know how to construct the name of the information (which may require a bit of context) but in principle the nodes can construct that context in real time if necessary, depending only on each other rather than any third party service.

5.5 Different requirements–similar mechanisms

I discussed above a set of architectural proposals that had the goal of allowing the insertion of service elements (what I call PHBs in Chapter 4) into the path of a flow of packets. In general, they use some mechanism that lets the provider of the overall service make an assertion that binds the name of that service to one (or a sequence of) PHBs as the packets flow from client to service. A quick comparison will suggest that the mechanisms used to implement this goal have much in common with the mechanisms used by architectures focused on content retrieval. In both cases, the provider of the content or service makes some binding from the name of that content or service to some location in the network to which the client should send packets. Some of the other architectural proposals make this point explicitly. For example, the designers of DONA stress that it can be used for a variety of purposes, not just for retrieval of content. Similarly, the designers of NDN stress that while content retrieval is the starting goal used to explain the NDN architecture, the mechanism is much more general.

Here is a summary of some of the naming/addressing/forwarding mechanisms I have discussed:

Proposal	Name Mechanism	Resolution Mechanism
NewArch	Forwarding Directive	Relies on underlying routing protocols
TRIAD	URL style	First packet: Forwarding through AS-level routers with name-based forwarding.
DONA	P:L, where P is hash of public key of creator and L is label unique to P	First packet: Forwarding through AS-level set of routers with name-based forwarding table. Subsequent packets: network-level source address
i3	Sequence of global, flat IDs	Forward packet through DHT to location where ID is hosted.
DOA	Sequence of global, flat EIDs	DHT system to convert EID to IP or further EIDs
FII	Region:ID, where ID is unique in region.	Forwarding driven by Pathlet source route.
MobilityFirst	Region:ID, where the ID is a global, flat identifier, and Region is a hint	Route packet to Region. If ID is not known, consult GNRS for valid NET
XIA	Content, Host, Net IDs in DAG	Route to first entry in DAG, fall back to next entry if first routing does not succeed.
NDN	URL style	Longest prefix match forwarding on URL for every packet.
PURSUIT	Nested scope/entity IDs	Recursive query of Rendezvous Servers in the nested scopes.
Netinf	Global flat ID	Name Resolution Service or Name-based routing.
DTN	Region:Entity, where Region is global and Entity is known within region.	Inter-region routing scheme.

In general, there are two sorts of high-level names that are used–those that are "flat"[4] (and often self-certifying–the hash of a public key or content) and those that have a hierarchical structure typified by a URL. A further distinction is whether the names have global uniqueness and routeability, or are only meaningful within some scope.

TRIAD and NDN use URL-style names, and both depend on a name-level routing protocol that distributes knowledge of (at the least the high-level component of) the name across the global network. However, TRIAD only propagates the top two or three levels of the DNS name, and only to select "name-based routers" in each

[4] *The term "flat" implies that the addresses have no structure to make the forwarding process more efficient. The current Internet assigns addresses so that regions of the network are associated with blocks of addresses; this means that the routers only need to keep track of these blocks (the prefixes of the addresses) rather than every destination in the Internet. Even so, a forwarding table in a core Internet router today has over 400K entries. The size of the forwarding table in a router is an important consideration in the design of a forwarding scheme.*

Autonomous System. NDN propagates some part of the name (perhaps to a finer degree than TRIAD) to every router in the system.

For flat addresses to work, there has to be some way to use those addresses as a basis for forwarding in the routers. One extreme would be to attempt to construct per-address forwarding tables in each router. As I noted in Chapter 2, the scalability of that approach was explored in [Caesar et al., 2006] and the results are not encouraging. Many of the schemes sidestep the issue of scale by using as the address both the flat identifier and some sort of forwarding "hint" that can be used by the router. DONA, i3, DOA, MobilityFirst and XIA all use flat identifiers with global meaning, but with different tricks to deal with the complexity of routing to a large number of flat identifiers.

- DONA uses a name of the form P:L (where P is the hash of a public key) to allow the option of propagating routes of the form P:*, thus allowing a route to a principal rather than an individual piece of content. The forwarding table may still be very large (on the order of the number of content objects in the system, rather than servers or physical end-points.) This scheme is perhaps the most aggressive in terms of pushing the limits of a routing protocol. The routing challenge in DONA may be of the same scale as with NDN, even though they use different style names.

- i3 and DOA associate identifiers with services or entities, not content, so the number of IDs is probably smaller than with DONA. Both use DHTs as their preferred mechanism to resolve these IDs.

- MobilityFirst has flat, global self-certifying identifiers for hosts, services and content. Normally, when a sender looks up a user-friendly name and gets back the global identifier, it also gets back a "hint": the id of a region (similar to an Autonomous System) within which the ID is likely to be known. It is assumed that within the scope of an Autonomous System it is reasonable to have a flat forwarding table at the level of content objects. Only if the ID is not known to the region is it necessary to deliver the packet based on the ID. For this purpose MobilityFirst includes a Global Name Resolution Service, which is similar in function to the mapping service in i3 or DOA, except that there may be many more names, if the IDs in MobilityFirst include names of content objects. There are two approaches being explored in MobilityFirst to build a demonstration GNRS: one based on a DHT and [[[Ask Arun for the latest description of where the research is going.]]]

- XIA similarly has flat, global self-certifying identifiers for hosts, services and content. To avoid having to depend on a global service to map IDs to locations, XIA takes advantage of the rich address structure in XIA, which is a DAG of IDs. The first ID expresses the intent of the sender (i.e., the desired item), and subsequent IDs in the DAG provide a fall-back forwarding option if the first ID is not known at a forwarding point. For example, the ID of a network (an Autonomous System, similar to MobilityFirst) can be included in the DAG, so that the packet can be forwarded toward the right AS where (again, as in MobilityFirst) it is presumed the forwarding tables will know about the ID that expressed the actual intent of the sender. The architecture of XIA does not include the requirement for any service that can take in an arbitrary ID and resolve it to a routable lower-level address.

All these schemes depend on the assumption that within some region of the network, forwarding on flat addresses will be practical. While [Caesar et al., 2006] drew pessimistic conclusions about the ability to forward based on flat addresses across the entire Internet, schemes to forward based on flat addresses in a region seem more practical. For example, [Kim et al., 2008] describe a scheme that can handle millions of flat addresses in a region. The approach they use is in fact somewhat similar to the GNRS of MobilityFirst: a Distributed Hash Table (DHT) in the various routers that can map an address to a forwarding decision. The scheme takes advantage of the more controlled environment of a region (for example, that the routers in the region can be reliably enumerated using a link-state routing protocol).

While most of the schemes above assume that names (whether of entities, services or content) have global meaning, TRIAD, FII, PURSUIT and DTN assume that entity names have meaning only within some region. Some schemes, such as MobilityFirst and Newinf, assume that names are globally unique but not always globally routeable. Both FII and DTN do not require that entities have IDs that are globally unique. There is no mechanism in either of them to look up an entity name and find out the region within which it is located. The user must find the location of the entity by query of a higher-level service. The NewArch proposal (see below) is similar in that it tried to avoid the creation of any names with global meaning.

NewArch NewArch is somewhat distinctive in this taxonomy, in that its abstract forwarding architecture (Forwarding, Association and Rendezvous Architecture, or FARA), did not include any sort of global ID of end points. The assumption in FARA was that some sort of ID would be needed so that sender and receiver could verify themselves to each other, but that these IDs should be a private matter between the end-points. FARA tried to avoid putting into the architecture (or more specifically, into the packet header) any ID that would act as a global identifier of the recipient. FARA assumed that communication occurred between *entities*, a term that could be applied to an application on a machine, a machine, a cluster, and so on. It was an abstraction that could be manifested in a number of ways. What the architecture required is that it be possible to construct a *forwarding directive* or FD that would allow the network (and perhaps the operating system of the receiving host) to forward the packet to that entity. In most reductions of FARA to practice, it was assumed that the FD would be some sort of source route. Once the packet had been delivered to the entity, there was a further element in the packet, which identified the *association* of which this packet was a part. A common form of association would be a TCP-like connection, and the association value in the packet would identify the right state variables in the entity for that association.

It was assumed that the association ID had no meaning outside the set of associated entities. It was assumed that the FD might contain some name for the destination entity as one of its elements, but that this would have meaning (like many source route elements) only in the context of that step in the forwarding process. For example, it might be a process ID inside a host. One of the goals of FARA was to hide, to the extent possible, exactly what parties were communicating across the network by avoiding IDs with global, long-lasting meaning. Of course, the FD may well contain a machine-level address, which provides considerable information about the communicating parties. But FARA did not preclude having stateful forwarding elements (such as a NAT device) and an element in the FD that was some sort of dynamic selector for the forwarding state stored in that device.

FARA presumed, but did not specify, some mechanism by which an entity that wished to receive communication would be able to construct a FD that would cause traffic to reach it. In a network like the Internet, this might just be an IP address followed by a process ID. Different instantiations of FARA might have FDs of a different form. FARA presumed but did not specify that there would be some sort of *rendezvous system* or RS to allow senders to find receivers. The receiver would *initiate* an entry in the RS with some higher-level name that could be used later to query the RS and retrieve the FD (or the instructions as to how to construct it). The *discovery* operation would take the high-level name as an input and return the FD. The RS also allowed the receiver to pass to the sender a *rendezvous string* (or the instructions as to how to construct it) which tells the sender how to format the information in the first packet of the communication to the receiver. Again, the meaning of this string was seen as a private matter between the sender and receiver–the mandatory function of the RS was just that a opaque string could be passed through it from receiver to sender. (This mechanism could be seen as a limited form of adding expressive power (a "scratchpad" in the packet)–weak since it is only intended for use by the end-points and is only used in the rendezvous mechanism.

If the FD turned out to be invalid (the desired entity was not at the location indicated by the FD), FARA deliberately did not provide any sort of global ID that could be used to find out the current location of the entity. The logic was thus: all the global name resolution schemes require that the receiver, if it moves in the system and changes the path to reach it, must update the information stored about it in the DHT, or routing table, or

whatever. Given that the receiver has to take some action to update its FD if the old one no longer works, it makes just as much sense to use the RS for this purpose. So if a sender cannot reach a receiver, its recovery action should be to make a fresh query of the RS and see if the FD has changed.

Dealing with adverse interests

Most architectures are described through the lens of aligned interests between sender and receiver. But most also contemplate to some extent the problem of adverse interests–when the sender is not trustworthy or perhaps simply an attacker. One can look at the different schemes in terms of how they deal with adverse interests. One must look at both directions of the communication. Most of the architectures are described in terms of sender and receiver (where the sender is the initiator of the connection–both ends will end up sending) or in terms of client and server, or in terms of content requestor and content provider. Whatever the framework, either end can show malicious intent with respect to the other.

The architectures introduced in this chapter that serve to configure services in the path from sender to receiver make explicit their role when interests are not aligned. All of the architectures use some sort of "firewall" device as an example of their utility. They deal to a varying degree with the problem of making sure that the attacker cannot bypass the protection device, but at least they acknowledge the issue. In contrast, the architectures that are focused on the delivery of content pay less attention to this problem–in particular to the problem that the receiver of the content may need protection in case the content contains malware. The DONA proposal is an exception; it explicitly discusses inserting a middlebox into the path of the returning content, although the described mechanism seems clumsy, as the task of inserting the protection service is delegated to one of the nodes that does name-based routing. TRIAD and NDN do not dwell on the issue of protecting the requestor from the content it requests; they describe a scheme in which the requestor simply gets the content as sent. They focus on the (admittedly hard) problem of routing the content request to the closest copy of the content, and do not discuss the problem of deploying services in the path from the content to the requestor, either protection services or functional services such as format conversion, which is an example used in the DOA paper of a useful service that might be delegated to an element in the net.

Counterpoint–the minimality principle

Some of the architectures I have described in this section offer a very rich delivery service. They support sending a packet in order to retrieve named content, or contact a service, or to a destination (or set of destinations in the case of multicast.) A contrarian (and minimalist) view might be that no matter what the high-level objective of the sender, in the end a packet is delivered to some location(s). These higher-level names (content or service) must somehow be translated into a destination so that the packet can actually be forwarded. The minimalist principle would suggest we ask why the network itself, in a manner specified by the architecture, should have the responsibility of doing those translations. Perhaps that function ought to be assigned to some higher-level service. The sender of the packet would query this high-level service, perhaps just before sending the packet, in order to get a low-level destination to which the packet is then directed. An example of such a design for multi-level naming is described in [Balakrishnan et al., 2004].

Why is it useful to embed the translation service into the network itself? In some cases, the network may be in the best position to make the translation. If the goal is to reach a version of a service (or a copy of some content) that is the fewest network hops away, or the closest in terms of round trip, or over a path that is not congested, the network might well have better access to this information. On the other hand, if the goal is to select a server that is not overloaded, or not failing, the network does not know this. The application is in a better position to implement this sort of selection. To make the problem harder, an application might want to make a selection based on both of these sorts of metrics, which implies some sort of cooperative decision-making–an idea that to my knowledge has never been embedded into an architectural proposal.

The same minimality question could be asked as a challenge to architectures such as DOA, which provide architectural support for the placement of services into a path from a sender to the final destination. Perhaps this

function should be implemented at a higher level. This is the way the Internet works: it provides no support for the configuration of services, and still works. The placement of services into a flow of content is either done at the application layer (as with Mail Transfer Agents in email) or is done using devices such as "transparent caches", which depend on contingent (or topological) delivery of the content to perform their function.

A possible criticism of architectures that focus on delivery to higher-level entities is that they have pulled down into the network architecture part of what used to be an application-layer function, without pulling *all* of the service requirements into that layer. The design of email, with its application-specific forwarding architecture, allows for certain sorts of protection services to be put in place in an application-specific way. The initial design of the Web did not address the issue of adding service elements to Web content delivery, which has led to various sorts of ad hoc implementation strategies. Pulling a content retrieval model that is Web-like into the network layer may actually make development of sophisticated application-specific forwarding architectures harder, not easier.

Expressive power

It is interesting that most of the architectures I describe here do not include in their design any explicit arguments to service points (PHBs). The schemes with URLs as content identifiers can encode any sort of information in the variable length ID, of course. Nebula has a very rich set of explicit arguments–the Proof of Consent and the Proof of Path. In general , most of the architectures seem to assume that the service elements will operate on the data payload (doing format conversion, inspection for malicious content and the like). Many of the architectures do distinguish the first packet in a connection, because that packet requires extra work to resolve IDs to more efficient network addresses. A careful review of all these architectures would include a catalog of any state in any service elements, how that state is maintained, and to on.

Almost all of the architectures try to avoid contingent delivery except for the basic packet forwarding mechanisms. They use intentional delivery, with the sender and/or the receiver specifying the path of the packet across the service points. The use of intentional delivery is probably an aid to better debugging when things go wrong.

Chapter 6

Longevity

6.1 Introduction–the goal of longevity

In comparison to many artifacts of computing, the Internet has lived to an old age–it is over 35 years old. Opinions differ as to the extent that it is showing its age, and among some researchers, there is a hypothesis that the Internet of 15 years from now might be built on different principles. Whether the network of 15 years from now is a minor evolution from today's network, or a more radical alternative, it should be a first-order requirement that this future Internet be designed so that it also can survive the test of time.

I have used the terms "longevity", or "long-lived", to describe this objective. The objective is easy to understand, but the principles that one would use to achieve it are less well understood. In fact, there are a number of different theories about how to design a network (or other system) that survives for a long time. In this chapter I argue the point of view that many of these theories are relevant, and that one can achieve a long-lived network in different ways, by exploiting various combinations of these theories in different degree. While some theories are incompatible, many are consistent with one another.

The approach I take here is inspired by the book Theories of Communication Networks [Monge and Contractor, 2003]. The topic of that book is not networks made of routers, but social networks made out of people and their relationships. They identify many theories that have been put forward to explain the formation and durability of social networks, and their thesis is that many of these theories are valid, to different degrees, in different such networks. So it is necessary to have a multi-theory, multilevel framework in order to explain the character of any given social networks. Since there are many examples of social networks in the real world, one can do empirical research to try to determine how (for example), theories of self-interest, collective action, knowledge exchange, homophily and proximity shape a given network. While we have fewer networks as examples than these authors do, we can still attempt to catalog the theories that have been put forward to explain why a network might or might not be long-lived.

6.2 Classes of theories

With some degree of over-simplification, many of the theories of longevity can be classified into three subclasses, as follows:

Theories of change: These theories presume that over time, requirements will change, so a long-lived network must of necessity change. Theories of this sort sometimes use the word "evolvability" rather than "longevity" to describe the desired objective, since they assume that a network that cannot change to meet changing requirements will soon cease to be useful. The word "change" as used here, usually has the implication of *uncertain* change; if the future trajectory of the requirements on a system could be completely characterized, one could presumably fold these into the initial design process, if the cost were not prohibitive. The XIA and FII proposals would fit in this category.[1]

[1] *The various architectural proposals I use as examples here are introduced and discussed in Chapter 5.*

Theories of stability: in contrast to theories of change, theories of stability presume that a system remains useful over time by providing a stable platform on which other services can depend. The NDN proposal might fit into the category.

Theories of innovation: These theories assume that change is beneficial, not just (or rather than) inevitable. These theories stress the importance of change and innovation as economic drivers. The FII proposal is specifically an example of this category.

These classes of theories are not incompatible. Theories of innovation are often theories of stability, in that the stability of the network as a platform allows innovation on top of that platform by what innovation theory would call complementors. Taking an example from operating systems, it is the stability of the interfaces to the operating system that invites application designers to take the risk of developing and marketing new applications for that system.

6.3 Architecture and longevity

I have defined the term "architecture" to describe the basic design concepts that underlie a system like a network: the top-level modularity, interfaces and dependencies, the assumptions that all parties must take as globally consistent, and so on. Within a theory of stability, architecture plays a natural role: it is part of what defines the stability. With respect to theories of change, however, the relationship is more complex. If architecture defines those things that we want to have longevity, how does architecture encompass change

Stable architecture that supports change: in this view, the architecture embodies those aspects of the system that do not change. It is the stability of the architecture that permits the overall evolution of the system. The XIA proposal, with its flexible address header, is an example of this category.

Evolving architecture: in this view, the architecture itself can (and does) evolve to address changing needs. If the architecture cannot adequately evolve, this leads to violations of the architecture, which (according to these theories) leads to a gradual loss of function, and an increasing difficulty of further change, an ossification of the system that gradually erodes its utility. The FII proposal is an example of this category, where the higher-level architectural framework allows the introduction of new embodiments over time.

The theory of ossification

The theory of ossification was perhaps first put forward with respect to operating systems by [Belady and Lehman, 1976]. They pose their First Law of Program Evolution Dynamics, the Law of Continuing Change, which states that a system that is used undergoes continuing change until it is judged more cost effective to freeze and recreate it. According to this point of view, systems lose the ability to evolve over time, and eventually have to be redone from scratch in order to allow continued change. So this theory is a theory of change, but an episodic theory, which predicts that systems (or architectures) have a natural lifetime, and need to be renewed from time to time by a more revolutionary phase.

New theories of design suggest that it may be possible to derive an architecture from a set of requirements by a rigorous and formal process. It is an open question how such an architecture will deal with change. If one changes the requirements and then derives a new architecture, the differences may be pervasive: essentially a new design rather than a modification of the old one. But if one takes an architecture derived in this way and modifies it after the fact, all of the theory that applied to the original design process no longer applies. This sort of action is like taking the output of a compiler and patching the machine code. It is thus possible that architectures that have been algorithmically derived from requirements will be brittle with respect to change, or (in term of these theories) easily ossified.

6.4 The theory of utility

All discussion of longevity must occur in the context of a network that is used. A network that is long-lived is a network that continues to be used over time. So it is a precondition of a long-lived network that it be useful in the

first place. (Chapter 2 lays out my framework for considering the extent to which an architectural proposal is fit for purpose.) So any theory of longevity must have inside it some theory of utility, which explains why the network is useful. The first theory of longevity I identify is based on a specific theory of utility.

The theory of the general network

According to this theory, a fully general system, which could meet all needs, would not need to evolve, and would thus be long-lived. The theory of the general network is thus a theory of stability.

The challenge, of course, is to define exactly what a general network might be.

The theory of the ideal network and impairments: according to this theory, networks provide a very simple service that can be described in its ideal (if unrealizable) form. One statement is as follows:

> An ideal data network will reliably deliver any amount of data to (and only to) any set of intended and willing recipients in zero time for zero cost and zero consumption of energy.

Of course, such a network cannot be realized. Some limits, such as the speed of light, are physical limits that cannot be violated. Others, such as cost, seem to improve over time as a consequence of innovation. Taken together, these limits, sometimes called impairments, define how far any practical network diverges from the ideal. In this theory, a maximally general network minimizes the various impairments, and to the extent possible, allows each set of users to trade off among the impairments to the maximum extent possible. Thus, queuing theory seems to capture a fundamental set of tradeoffs among speed, cost (utilization) and delay. A network that (for a given class of traffic) does as well as queuing theory would predict, and allows the users to move along the performance frontier defined by queuing theory, would be seen as a maximally general network.

According to this theory, if a network is maximally general with respect to the fundamental impairments (a theory of stability) and is open to change with respects to impairments that change over time (a theory of innovation), then such a network will be long-lived.

Many have seen the Internet as a good, if pragmatic example of a general network, and see its longevity as a consequence of that fact. The use of packets as a multiplexing mechanism has proved to be a very general and flexible mechanism. Packets support a wide range of applications, and allow for the introduction of new technology as it evolves. Tools for Quality of Service allow the application to control the tradeoff among such parameters as cost, speed, and delay.

The theory of real options

Real option theory captures the idea (in common with options as a financial instrument) that one can attempt to quantify the cost-benefit of investing now to "keep options open", or in other words to deal with uncertainty. It is thus a theory of change, to the extent that change equates to uncertainty. It is also a theory of the general network, but in economic terms, in that it suggests that one can spend money now to purchase flexibility later to respond to uncertain change. It does not describe what the resulting general network is (in contrast to the offered definition above), but just postulates that generally is often to be had, but at a price.

Real option theory is perhaps more often applied to the construction of a network (e.g. how much spare capacity to purchase now) than to the architecting of a network. But the theory none the less reminds us that generality may come at a price, and that price is one of the impairments to the definition of the ideal network postulated above.

6.5 The theory of tussle and points of control

The discussion of the ideal network does not fully capture what happens inside networks, because the ideal is stated from the perspective of only one class of actors–the parties desiring to communicate. The statement of the ideal does not afford any attention to other actors, such as governments that want to carry out lawful intercept on traffic, to employers and others who want to limit what can be carried over their networks, and so on. The list of

stake-holders that can be identified in the current Internet is substantial, and each of these stake-holders tries to put forward their interests, perhaps at the expense of other stake-holders.

This ongoing process has been called tussle [Clark et al., 2005b], and seems to be a fundamental aspect of any system (like the Internet) that is deeply embedded in the larger social, economic and regulatory context. According to the theory of tussle, systems will prove to be long-lived if they are designed to minimize the disruptive consequence of tussles, and in particular so that tussle does not lead to violations of the architecture of the network. Various aphorisms have been used to describe how a system should be designed to tolerate tussle.

> The tree that bends in the wind does not break.
>
> You are not designing the outcome of the game, but the playing field.

The idea behind these aphorisms is to design your systems so that they do not attempt to resist tussle and impose a fixed outcome, but to be flexible in the face of the inevitable. However, they give little practical guidance as to how one might do it, other to hint that one can tilt the playing field to bias the resulting tussle consistent with the values of the designer.

Tussle isolation: One design principle that emerges from the consideration of tussle is a new modularity principle called tussle isolation. Computer science has a number of theories of modularity, such as layering (e.g. the avoidance of mutual dependency). The idea behind tussle isolation is that if the designer can identify in advance an area where there is likely to be persistent tussle, then the design should isolate that area so that the resulting tussle does not spill over into other aspects of the network.

- DNS: if the early designers had understood that the DNS would include names over which there would be trademark disputes, that use of such names could have been made a separate service, so that the scope of the trademark disputes could be minimized.
- Secure BGP: if the designers of tools to secure BGP had understood that the real tussle would be over which actors would be trusted to vouch for different regions of the Internet, they might have designed a different trust framework that allowed these tussles to be better contained.

Placement of interfaces: In addition to isolating tussle, one can "move it around" by the placement of critical interfaces within a system–another example of a non-technical principle for modularizing a system.

Removal of interfaces: a sub-class of the above theory is the idea that by intentionally removing interfaces and making the system less modular and more integrated, one can increase the power of the firm that owns the system, and limit competitive entry as well as other forms of tussle. This theory is an example of a theory of stability (as well as a theory of market power and hegemony–see below.)

Asymmetric struggle: Many tussles are defined by the fact that different stake-holders have access to different sorts of tools and methods. Network architects define module interfaces and shift function around, governments pass laws and regulations, network operators make investments and configure the physical network. Each of these actions can advantage the particular stake-holder and disadvantage the adverse interests. Given this fact, it is worth some study (a full exploration is beyond the scope of this chapter) as to how these various methods interact with each other. Like a game of "rock, paper, scissors", they sometimes seem to circle around each other in endless cycles. One sub-theory that characterizes some of the interactions is the theory of the blunt instrument, the idea that while each stake-holder has distinct powers, the design of one part of the system can blunt the tools of control that the others have, and thus render them less effective. Thus, for example, the use of encryption as part of the network design greatly limits the ability of other actors to observe (and thus to impose limits on) what the users are doing. In the extreme, the network operator is reduced to carrying all traffic, blocking all encrypted traffic, or refusing to serve the relevant customer–an example of blunting the network operator's instrument of control.

Tussle and longevity

The theory of tussle might be seen as the theory of change, but in fact it may be closer to a theory of dynamic stability. Stability need not imply a fixed system, but can also imply a system that has checks and balances, or feedback, to home it to a stable point. Tussle can be viewed as such a mechanisms–a set of forces that tend to bring a system back to a stable compromise point if some new input (e.g. a technical innovation) shifts it away from that point. Over time, the compromise point may shift (as social norms shift over time) and the stable point may be different in different cultures. So tussle can be seen as a dynamic and ongoing mechanism that contributes to system longevity, and further as a mechanism that allows the outcome to be different in different cultures, as opposed to a rigid system that attempts to impose global agreement in contexts where global agreement is not feasible. This variation in outcome, as well, is a contributor to longevity.

6.6 The theory of building blocks and composable elements.

The theory of the general network assumed that one could describe what an ideal, or fully general network would do. It was based on the concept of a network as a system with a very simple core function. Another point of view is that a network should be capable of offering a much richer set of services (perhaps not all at the same layer). The measure of a network would not be how well it does at limiting the impact of impairments, but how easy it is to incorporate new sorts of services between the communicating parties. In this point of view, if the network is built only of fixed-function routers, that is a limiting rather than a stabilizing outcome. My discussion about expressive power in Chapter 4 gets at this tension: should a network strive for minimal expressive power or a rich set of tools to add new PHBs as needed. The proposals i3, DOA, and Nebula attempt to capture the generality of arbitrary service composition.

This point of view becomes more prevalent if one looks not just at the simple, packet-forwarding layer, but at services "above" that layer, which might do things such as convert information formats, validate identity, provide various sorts of security services and the like. In this layered view, one would then ask of the packet layer whether it was optimally suited to support the deployment and configuration of these higher-level services. For example, to insure the proper operation of security services, it might be important to make sure that the packets cannot bypass the services as they are being forwarded. So the desire to deploy these higher layer services may change and expand the requirements at the packet level, even if these services are seen as "higher layer" services.

There seem to be two, perhaps contradictory, theories of building blocks and composable elements–the maximal and the minimal theory.

In the maximal theory, a network will be long-lived if it has rich expressive power, so that new service elements can be introduced and invoked. At the packet level, expressive power would be increased by adding more powerful addressing modes (such as source addressing, which could route a packet through a series of service elements) and additional fields in the packet header that could be used to convey additional information to the service elements. If the service elements act on larger data units that are assembled out of packets at the point where the element is located, this sort of expressive power will be controlled by data at the application layer. (Mail Transfer Agents are an example of higher-level, or application-level service elements. They act on and modify the header of the mail message, and schemes for mail encryption are defined to encrypt the body but leave the header visible so the mail system can function properly.)

The opposite, or minimal, theory about service elements and expressive power arises within the theory of tussle. In this point of view, any service element will be a point of contention and tussle, as different stake-holders try to control the service being provided. Thus, ISPs sometimes block access to third-party mail transfer agents in an attempt to force a customer to use their mail service; by doing so the ISP may be able to impose limitations on what the customer can do (for example what are acceptable email names). This theory would suggest that a network design might deliberately limit the expressive power of the design (perhaps at certain of the layers in the design), to limit the points of tussle in the network, and thus bring about longevity through stability.

The theory of programmable elements (active networks)

The theory that building blocks bring beneficial flexibility has an aggressive version in which elements within the network can be programmed dynamically, perhaps even by means of programs carried within the data packets themselves. This point of view, sometimes called Active Networks, can be argued as reducing tussle rather than facilitating it, since it tilts the playing field toward the end-user, and blunts the instruments of control that belong to the stake-holders "in" the network. The programs come from the edge, selected and installed by the end-user or his agents; the stakeholders who are in the network only provide the platform for these programs. They cannot easily regulate what those programs do, except by attempts to impose limits on how they are composed. With no ability to see what the programs do, and only a "blunt instrument" capability to limit how they are composed, the result (according to this point of view) is a stable platform (see below) on which innovation can be driven from the edge.

6.7 The theory of the stable platform

The theory of the stable platform is a theory understood by those who study innovation. According to this theory, innovation (which represents a valuable form of change) is facilitated by a stable platform with an unchanging interface and service definition. In the language of this theory, those who innovate "on top of" the platform are called complementors. If the platform itself is unstable and subject to change and innovation, this increases the cost of building complementary systems (e.g. applications) that exploit the platform (as the application must be upgraded to keep pace with the platform changes) and increases the risk (due to uncertainty about changes in the platform that might reduce the functionality of the application). This theory is an example of a theory of innovation that is a theory of stability. For an extended discussion of platform theory as it relates to the Internet, see [Claffy and Clark, 2014].

The theory of the stable platform can be stated in dynamic form: to the extent that there are a number of complementors, they will use their power to argue for the stability of the platform, which will induce more complementors to join, and so on, in a positive feedback situation. The reverse of this dynamic is also a part of the theory; if a platform is not useful, it makes no difference if it is stable. Again, the packet forwarding service of the Internet has been seen as a good illustration of a stable platform that permits innovation on top of that platform. The theory of the stable platform has been used to explain the longevity of the current Internet.

6.8 The theory of semantics-free service

The theory of the stable platform does not say anything about what function the platform should implement in order to be useful and general. The theory of the general network provides one answer to that question: the platform should provide a general service that is as close to the ideal (the minimum set of impairments) as can be fashioned.

The version of the theory of the general network offered above was that the network should just deliver bytes. In contrast to the theory of composable building blocks, the network should not have any model of what those bytes represent, or what the high-level objective of the application is. This version of the theory has sometimes been called the semantics-free network, or the transparent network, or (in more colloquial terms), "what comes out is what goes in". The assumption is that if the network begins to incorporate a model of what one or another application is trying to do, it will end up being specialized to those applications, at the cost of generality.

It has been argued that the longevity of the Internet is due to its semantic-free design, and the refusal of its designers to allow the protocols to be optimized to the popular application of the day. It could be argued that semantics-free service is an example of the theory of utility, but it is not clear what line of reasoning would be used to make this point in advance. However, the theory of the general network may imply the theory of semantics-free service, since (as it was stated earlier) the general network was defined as "delivering data", which seems to imply a semantics-free service.

This theory is a close relative to the end-to-end argument, but in the beginning that argument was about correct

operation, not about generality. The interpretation of the end-to-end argument as an argument for generality can be found implicitly in the original paper [Saltzer et al., 1984], but has become more elaborated and explicit in some of the subsequent writings about the argument.

6.9 The theories of global agreement

One conception of network architecture, as I proposed in Chapter 1, is that it defines those aspects of the system about which there must be global agreement: architecture defines those parts of the system that "work the same way everywhere". In this context, there are actually two theories about global agreement and longevity: the minimal theory and the maximal theory.

The theory of maximal global agreement: This theory postulates that the more aspects of the system are well-defined, the more stable the platform. By providing a well-specified functional specification for the platform, the difficulty and risk to the complementor is minimized. The word "maximal" is probably an overstatement of this theory–the more careful statement would be that "up to a point", increased specification and careful definition is a good thing.

The theory of minimal global agreement: This theory is a theory of change. It states that the fewer things we all have to agree to in common, the more we will be able to accommodate a range of uses with different needs. As long as the platform remains useful, having fewer points of global agreement is beneficial, and will allow the network to evolve over time without disrupting the utility of the platform. So in contrast to the maximal or "up to a point" theory, this is a "down to a point" theory, or perhaps (to paraphrase the quote of Einstein): Architecture should be made as minimal as possible, but no less. The FII proposal is an explicit example of this theory.

False agreement: Whichever version of the theory is put forward, there remains the question of when a global agreement is really an agreement, and when it is the illusion of agreement. An example from the Internet might be the initial assumption that the Internet was based on the global agreement that there was a single global address space. It was thought that this agreement was important, and one of the basic tenets of the stable IP platform, but then Network Address Translation devices were introduced, and the Internet survived. Some would say that because NAT devices impair certain classes of applications (in particular, passive servers located behind NAT devices), we should view NATs (and the loss of global addresses) as a significant violation of the stable architecture. Development of protocols, discussed in Chapter 4 that allow state to be installed dynamically in NAT devices (perhaps an example of the theory of the building block), have the potential to support essentially all the applications it did in the era of global addresses.

However the reader might choose to analyze this example, the more general question is how one would test a proposed point of global agreement to see whether agreement is actually required about the point in order to have a stable platform. Clever reconceptualization may allow what was seen as a global agreement to be set aside with no loss of power.

One might pose an informal "test of time" approach: that a presumed point of global agreement should only be judged in hindsight based on whether people actually depend on it. But this seems like a poor candidate for a design principle. On the other hand, it seems difficult to take the position that we can force dependency to force stability. The theory of utility suggests that if a feature is not useful, it does not matter if it is stable, or if it is a point of nominal global agreement.

6.10 The theory of technology independence

The theory of technology independence is another theory of stability in the face of change. This theory states that a system will be long-lived if it allows new generations of technology to be incorporated into the system without disrupting the stable platform that the system provides to the complementors. Since technology evolved rapidly in the CS world, it would seem that any long-lived system must be designed so that it is not rendered obsolete by new technology.

Again, this theory can be used to explain the longevity of the Internet. The simple, packet-based platform of the

Internet can be implemented on top of all sorts of communication technology. The Internet has accommodated circuits that have increased in speed by at least six orders of magnitude during its lifetime. It has accommodated multi-access local area networks, wireless networks, and the like. The applications running on top of the IP interface are largely unaffected by these innovations.

6.11 The theory of the hourglass

The combination of the theory of the stable platform and the theory of technology independence lead to a theory (or a picture) that is a hourglass: a picture of a narrow waist representing the common point of agreement (a global agreement?) on the IP layer, with great diversity in technology below and great diversity in application above.

Once the image of the hourglass was identified and associated with a theory of longevity, further study revealed that the Internet had many hourglasses in it: the reliable byte-stream on which email sits (the Internet standards for email work quite well on transport protocols other than TCP), HTTP, and so on. [other useful examples?]

6.12 The theory of cross-layer optimization

The theory of cross-layer optimization is a contrarian theory, contrary to the theory of the hourglass. The theory of cross-layer optimization states that over the long run, the evolution of technology will be so substantial that a stable, technology-independent platform will become limiting, and eventually, uncompetitive compared to an approach that allows the application and the technology to adapt to each other. The application designer will have a harder task than with a stable platform, but in exchange for doing the additional design work so that the application can adapt to different technologies, the designer will achieve greatly improved performance and function.

The theory of cross-layer optimization has been put forward in the past in the context of various emerging technologies, perhaps starting with multi-access local area networks. In the past, the theory of the stable platform has dominated. Today, cross-layer optimization is being put forward in the context of some wireless networks, especially wireless designed for very challenging circumstances, such as battlefield networks. It is not clear whether longevity is the primary requirement for such networks.

6.13 The theory of downloadable code

The theory of downloadable code is a theory of change, or perhaps of innovation. This theory states that the need for global agreement can be minimized by the approach of downloading code into the communicating elements, so that the agreement is achieved not by the mandate of standards but by an agreement to run compatible software.

If the code were downloaded into the network elements that forward packets, this would be the same as the theory of active networks. This theory has not achieved much traction in the real world. However, code that is downloaded into the end-node (most commonly at the application layer, or as a supporting service to applications) has been a very powerful tool to support innovation. New formats for audio and images (still, animated and video) are introduced by allowing end-nodes to download new rendering code. Standards such as PDF, Flash, various representations of audio and video and the like enter the market by means of free viewer software provided by the creator of the standard. Pragmatically, once a format can be implemented in downloadable software (as opposed to hardware, for example), proliferation of competing standards does not seem to be an impediment to progress and longevity.

The theory of downloadable code is an example of a theory of the stable platform: in this case the platform is a software platform, not a network service platform (such as the IP layer). The browser of today, with its "plug-in" architecture, becomes a stable platform on which innovation (e.g. new downloadable modules) can be built.

This observation begs the question of what parts of the network could be based on downloadable code, rather than on global agreement. Today, for example, transport protocols such as TCP are more or less a global agreement. Alternatives cannot be downloaded, because the code that implements TCP is embedded in the kernel of most

operating systems for a number of reasons: performance, dealing with interrupts and timers, multi-threading, efficient demultiplexing and buffer management, security and the like. However, is this a fundamental consequence of some aspect of transport protocols, or just a historical accident? It might be possible to design a framework (or platform, to use the earlier word) into which different protocols at this level could be downloaded, just as the web browser provides a framework for downloadable code at a higher level. Were this framework demonstrated, one could argue that the theory of downloadable code would be a better path to longevity that the theory of global agreement, even at the transport layer of the protocol stack.

6.14 Change: hard or easy?

More abstractly, the theory of downloadable code challenges us to take a rigorous look at what makes change hard or easy. The need for global agreement seems to make change hard (if everyone had to change at once).

Version numbers are sometimes put forward as a technique to manage change. Version numbers in protocols can allow two incompatible designs to co-exist, either transiently during a time of change, or (more realistically) forever. Version numbers work so long as it is possible to verify that all the components that will be involved in some operation support at least one version in common. Proposals such as XIA and FII try to facilitate change (in different ways) by making it easier to make changes gradually, across different parts of the network at different times.

Changes to production code are often viewed as very hard to make, or at least not quick to make. Vendors need to be convinced of the need for change, and then the change must be scheduled into the development cycle. Especially if the change is based on a standard that requires broad agreement, such changes can take years. However, one should not mistake the time it takes to make a change with fundamental difficulty. What makes a change easy or hard to implement is more its interaction with other parts of the system (which, according to the theory of ossification, will increase over time).

On the other hand, when the change is more a bug-fix and the need is urgent (as with the discovery of a security vulnerability) changes can be made in a matter of days or weeks, and the current trend to automate the downloading of new versions (e.g. of operating system and major software packages such as Office) can allow substantial deployment of updates in days.

Overall, there is a trend in the current Internet (and the systems attached to it, such as operating systems) to make change (updates, patches, releases of new versions) easier to accomplish. This trend begs the question of which changes are actually hard to make, and why. The theory of minimal global agreement would suggest that if the right tools are put in place to allow software to be upgraded, there is little that cannot be changed in principle, and more and more that can be changed in practice. With the trend of moving function from hardware to software (e.g. software-defined radios) functions that had traditionally been viewed as fixed and static have turned out to be very amenable to change, and not fundamental at all.

The FII proposal, as well as the DTN work, bring our attention to an aspect of the current Internet that, while not a formal part of the architecture, seems to have frozen in a way that resists change. Today, most applications get access to the Internet via a "socket" interface that presumes a two-way interactive reliable flow among the end-points, which in practice means TCP. In contrast, in a DTN many nodes may only be connected intermittently, and many applications may be able to tolerate a more "store-and-forward" mode of transport between the end-points. So a more general network API may be an important part of building a more general version of the stable platform. FII includes in its required points of agreement a set of tools to allow the network API to evolve.

6.15 The theory of hegemony

The theory of hegemony is a theory of stability. It postulates that a system will be long-lived if a single actor is in charge of the system, an actor that can balance change against stability, and balance the needs of the various stake-holders in an orderly way. By taking tussle out of the technical domain and into the planning or administrative (regulatory) context of the controlling actor, the platform becomes more predictable and thus more

appealing as a platform. So the theory of hegemony is a theory of innovation based on stability.

The telephone system, for most of its life, was an example of a system managed according to the theory of hegemony, with single providers (often parts of the government) in most regimes, and standards set through a very deliberative body: the ITU (or earlier the CCITT). One interpretation of history is that this approach led to a very stable system that was easy to use, but to a system that inhibited innovation. However, the low rate of innovation can be explained by the theory of utility: the platform provided by the telephone system, the 3kHz channel, was not very general (in other words, not useful except for the carriage of phone calls), so the failure of innovation is due to the limited utility of the platform, not the presence of the controlling interest. However, following the reasoning one step deeper, one could argue that this outcome is due to the lack of interest in innovation by the controlling interests.

6.16 The present Internet

A number of theories have been identified as contributors to the observed longevity of the Internet: the theory of the general network, the theory of the stable platform, the theory of semantics-free service, the theory of technology independence, the resulting theory of the hourglass, perhaps the theory of minimal global agreement, and (to some extent increasing over time) the theory of downloadable code (in the end-nodes). The Internet seems to reject the theory of hegemony, and the theories of composable elements and downloadable code in the network.

Global agreement:

The early designers of the Internet assumed that substantial global agreement would be a necessary step toward an interoperable network. (In those days, downloadable code was not a practical concept.) Over time (in an application of what was called above the "test-of-time" approach), the real degree of global agreement has emerged.

Addressing: The original design assumed a single, global address space in which all end-points (or interfaces, to be precise) were found. This idea has been replaced by a more complex concept, in which there are lots of private address spaces, with translation among some of them using NAT devices, but there is still a single common addressing region–a region in the "center" of the network where a pool of addresses are given a consistent common meaning. Services that want to make themselves widely available obtain an address (or an address and a port) in the common addressing region, so that other end-points can find them.

TCP: The original design was careful not to make TCP mandatory–the designers were careful to say that alternatives to TCP should be anticipated. The socket interface to TCP is not a part of the Internet standards, Over time, however, in an example of the dynamic form of the theory of the stable platform, enough applications have used TCP that it is mandatory in practice, which means that other applications take on little risk in depending on it, and TCP has emerged as a required point of global agreement.

TCP-friendly congestion control: This idea was not part of the original design–in the beginning the designers did not have a clear idea about dealing with congestion. However, in the 1990s (more or less), as congestion control based on the slow-start algorithms and its enhancements matured, there was a sense that every application, and every transport protocol, needed to behave in the same general way. So there was a call for a global agreement on the congestion behavior called "TCP-friendly". To a considerable extent, this norm was imposed, but it seems today as if there is a drift away from this approach (based on economic issues and the theory of tussle) to a model where the network takes on a more active role in enforcement.

DNS: The architects of the Internet have always been ambivalent about whether the DNS is a core part of the architecture. It is not strictly necessary: one can use other tools to translate names into addresses (as some applications do), or just type IP addresses where one would normally type a DNS name (e.g. in a URL). However, as a practical matter, the DNS is a necessary component of the Internet for any real use, and the amount of tussle surrounding the DNS (trademark, diverse alphabets, governance, TLDs, etc.) both suggest that it is a point where global agreement is required, and also that it is a prime illustration of tussle. One can look at the simple interface (send a name, get a number) as a stable platform interface under which all sort of change has happened.

The Web: The web standards have emerged as a critical platform for the growth of the Internet. While the web is "fijust one application among many" it is clearly (as of now) the dominant application, and as such, embodies many attributes that can be explored using these various theories–tussle, platform, downloadable code and so on. But without global (if rough) agreement on many aspects of the web, the Internet experience would not be what it is today. The specification and deployment of SSL and TLS a good example of the injection into the Internet of a new set of points about which there needs to be widespread (if not quite global) agreement.

The packet header: Participation in the Internet does require agreement on how a packet is formatted, and what (at least some of) the fields mean. The address field may be rewritten as the packet traverses NAT boxes, but there are still some constraints imposed by Internet addressing (e.g. the length, the TCP pseudo-header and the like) to which all players must conform. Despite the push to deploy IPv6, the IP header seems to be a manifestation of the stable platform, rather than something that is shaped by a theory of change.

6.17 The future

As I have indicated through this chapter, there are a number of considerations and theories about how to design a future network such that (among other things) it is long-lived. Several of the architectural proposals I have discussed take a very different approach in striving for longevity: stability vs. change, minimality vs. evolving services and so on. But the relevance of these choices only applies if the architecture passes the basic test: the theory of utility. If the network is not useful–if it cannot fulfill basic requirements–it will not be given a chance to demonstrate its ability to be long-lived.

In the next chapters of the book, I turn to a detailed look at some of the critical requirements I identified in Chapter 2, starting with security. But here I note some specific considerations that link these various requirements to different theories of longevity.

Security A first order objective for a future Internet is that it be more secure. Security (attack and defense) is perhaps the extreme example of tussle; it implies ongoing (and unpredictable) change, even if attack and defense stays in some tolerable balance. So the presence of attackers in the system would seem to imply that at least some part of a future Internet must seek longevity using a theory of change, not a theory of stability.

Any component in the network will be a target for attack. So the theories of building blocks and composable elements might seem to lead to a future network with more options for security vulnerabilities. This concern must be addressed by advocates for those theories.

Management The discussion of the Internet of today focused on the data plane, and considered issues of addressing and naming from the point of view of global agreement and stability. That discussion paid little attention to issues of management, in part because that area is so poorly developed from an architectural perspective. In a future Internet, management must receive more attention, for a number of reasons. This objective will lead to the creation of a new set of interfaces, and will raise a new domain to which these various theories must be applied. Many of the Interfaces will be between peer components (between ASes or ISPs) so they are not platform or layering interfaces. It is not clear what theory of longevity should apply to such interfaces.

Chapter 7

Security

7.1 Introduction

The Internet of today is generally considered to provide a poor level of security. In this chapter I attempt to discuss why this is so: the mix of historical, architectural and pragmatic reasons why Internet security is found lacking.

This chapter will perhaps not resemble a normal paper on security, which might identify a vulnerability and pose a solution. This chapter is concerned with a more general challenge, which is how to identify and classify the range of security problems that will arise in the context of a global Internet, how to allocate the responsibility for dealing with these problems to different parts of the network ecosystem, and how to decide which issues rise to the level that implies an *architectural* response. This chapter is concerned with what might be called *security architecture*, and a more traditional security paper might take up where this chapter leaves off.

7.2 Defining security

The first issue is to consider what is actually meant by the word "security". Without a clear definition of what is meant by the word, it is not very meaningful to discuss whether we have enough of it. The concept of security captures a range of issues, which may not actually have that much to do with each other–"security" is a "basket word", like "management", which I will deal with in a later chapter.

Computer science tends to define security in terms of the correct operation of a system: a secure system is one that does what it is supposed to do, and does not do unacceptable or unsafe things, even when it is under attack. This approach, of course, requires the functions of a system to be well-specified. There is an old saying among security experts: "A system without a specification cannot fail; it can only present surprises."[1]

A user might not think about security in the same way. What a user cares about is whether adequate steps have been taken to reduce the probability of bad events to a tolerable level. Users care about outcomes; technologists tend to address inputs. An analogy from the physical world may help. A home security expert might say that a home has a "secure door" if it has a good lock and is strong enough to resist being kicked in. But what the home-owner cares about is whether, all things considered, the probability of being burgled is low enough to accept.

As another perspective, a political scientist of the realist school might define security by saying that a nation is secure if it can sustain peace at an acceptable cost, or alternatively if can prevail in war. Security is not automatically equated to peace; unconditional surrender will create a state of peace, but not one of security, since the price of unconditional surrender is presumably very high. In this framing of security there is no attempt to define what "correct operation of the system" would mean; that would be nonsense with respect to a nation taken as a whole. It is a pragmatic decision of the leadership whether the costs of peace are lower than the costs of war. The military exists both to deter war and prevail at war.

[1] *I cannot determine who first said this. I have questioned a number of elders in the field, all of whom agree that they said it, but believe they got it from someone else.*

While users may care about outcomes–keeping the risk of harms to a realistic level–network designers are forced to work in the space of inputs. We are forced to address security by making the components strong (correct), exactly because the Internet is a general system. Just as we designed the Internet without knowing what it is for, we have to design its security components without knowing what security problem we are solving. Most folks would understand that it would be nonsense to ask for a door to be designed without knowing whether it was for a house or a prison cell. But dealing with that sort of uncertainty is the price of building a general-purpose network. Perhaps it will turn out to be fundamentally easier to deal with functional generality than security generality; perhaps we have just not yet figured out how to think about security in this general, abstract way. But that is the challenge I must address in this chapter.

The rest of the chapter proceeds as follows. First, I offer a way of sorting out the landscape of network security, to provide some structure to the discussion that follows. Building on that, I focus on the issues of trust, and trust management, as a key to better overall security. I then proceed to a narrower topic that brings me back to the theme of the book, the relation of architecture to these various aspects of security; I consider how architecture, within the minimalist framing, can contribute to better security.

Defining network security

Setting aside for the moment the range of possible definitions of *security*, we should look more specifically at the range of issues that make up *network security*. There is no one single issue that defines network security; in fact it might be more useful to abandon the term and just talk about the range of security issues that come up in the context of the Internet.

Security is sometimes defined by breaking the problem into three sub-goals, confidentiality, integrity and availability (the CIA triad), and I will refer to that structure when it is relevant, but in fact, for many of the issues that I list, this structure is not very helpful. To begin, I structure my discussion of network security by looking at the structure of the system, a taxonomy that derives loosely from the layered structure of cyber-space (asking where a malicious action manifests). Later in the chapter I will return to an output or "harms-based" taxonomy of security, and ask what this might teach us about how to think about the inputs–the correct operation of the system elements under attack.

Here is my taxonomy based on where the attack manifests:

- **Attacks on communication:** This is the problem, sometimes classified as information security, where parties attempting to accomplish mutual communication are thwarted by an attack, perhaps launched by the network or by some party that has gained control of some critical control point.[2]

  This is a space where the traditional triad of confidentiality, integrity and availability (CIA) has some validity, as I will discuss below. Another set of issues in this category falls under the heading of *traffic analysis*. That term describes the form of surveillance where the observer is not looking at what is being sent but who the sender and receiver are. Knowledge about who is talking to whom can be as revealing as exactly what is being said.

- **Attacks on the attached hosts:** Attacks on attached hosts can occur as a result of communication with a malicious party (who uses the capabilities of one or another layer to deliver an attack) or as a result of an unsolicited incoming packet that somehow exploits a vulnerability to launch a successful attack. In the discussion of expressive power in Chapter 4, this class of attack maps onto the case where the interests of

[2] *A few years ago, there was a furor in the U.S. because Comcast blocked a peer-to-peer music sharing application (BitTorrent) by injecting forged packets into the data stream. This was not characterized at the time as a "security" event but as a violation of the norms of service, but in the language of security, this was without a doubt an attack on a communication by the network. End-to-end encryption would have detected this particular attack, but since this was intended to be an attack on availability of service (see below) there could have been many other approaches. In the discussion of expressive power in Chapter 4, this class of attack maps onto the case where the communicating actors have aligned interests, but some element in the network is hostile to those interests.*

the end-points to a communication are not aligned. The receiver may choose to draw on resources in the network (PHBs) as a means of protection. The expressive power of the network must be analyzed to see in what ways it can be exploited by either the attacker or the defender.

- **Attacks on the network itself:** These include attacks on network elements, the routing protocols, attacks on critical supporting services such as the Domain Name Service (the DNS), and the like. Since the core function of the Internet is actually rather simple, there are only a few of these services; the interesting question is why they remain insecure. I return to this below. To the extent that this layer cannot detect and remedy the consequences of failures and attacks internally, the consequences of attacks at this layer will become visible to the layers above, which will have to take corrective action.

- **Denial of Service attacks:** Denial of service attacks (usually called Distributed Denial of Service attacks or DDoS, because many machines are exploited to launch the attack), do not quite fit into these devisions. They can be classified as an attack against the network if they exhaust the capacity of a link or switch, or as an attack against a host if they exhaust the capacity of that host. So I consider this class of problem separately.

7.3 A historical perspective

Some of the early Internet architects, including me, have been criticized for not thinking about security from the start. This criticism is to some extent valid, but in fact we did consider security; we just did not know at that time how to think about it. We made some simplifying assumptions that turned out to be false. Interestingly, much of our early advice about security came from the intelligence community (the NSA) and their particular view biased our thinking.

The NSA had a very simple model of protecting the host from attack: the host protects the host and the network protects the net. They were not prepared to delegate the protection of the host to the network because they did not trust the net. So our job was to deliver everything, including attacks, and then the host would sort it out. We now see that this view is over-simple and thus not totally realistic.

Within the CIA framing, the intelligence community gives the highest priority to confidentiality–the prevention of declassification and theft of secrets. Their view is that once secrets are stolen, the damage is done. What we now see is that users care most about availability–their ability to get the job done.

Because the intelligence community assumes an attacker with a very high skill level and motivation, they argued only for mechanisms that were "perfect". A mechanism that only provided a degree of protection just defined how much effort the adversary would have to expend, and they assume the adversary would be prepared to expend it. Today, we see that in many cases the attackers are very concerned with the amount of effort required, and it is probably a foolish idea to pursue perfection.

The CIA framing separates the world into two sets of people–those who are authorized and those who are not. If an actor is authorized, then they can see the information, and if they modify it, this is not corruption, just modification, since they are authorized. If an actor is not authorized, then the goal of the system is to deny them access.

This framing is deceptive, but it shaped our early thinking. We knew that some routers might be suspect, so there was no way we could insure that a router did not make a copy of a packet–the packet forwarding layer could not itself provide confidentiality. And a malicious router might modify a packet–the packet forwarding layer could not itself provide integrity for data in transit. We took a very simple view, which is associated with the "end-to-end" mode of thinking: only the end points could undertake to mitigate these vulnerabilities and achieve these objectives because only they could know what the objective was, and only they were (presumably) trusted and authorized to exchange this data. End-to-end encryption is the obvious approach: if the data is encrypted making a copy is useless and any modification can be detected.

When the Internet was initially being developed, encryption algorithms were too complex to be implemented in software. They had to be off-loaded to specialized hardware. This reality was a barrier to deployment; not only did every machine have to be augmented with such hardware, there had to be broad agreement as to the algorithm to be used, which was hard to negotiate. But there was an expectation that we could move to the use of end-to-end encryption at some point in the future.

This approach theoretically resolved confidentiality and integrity, and left only availability for the network to solve. Of course, "all" the network does is deliver packets, so it would seem that availability is the core requirement. In this context, it is interesting that we have no "theory of availability", which is the subject of a later chapter.

Why was this conception of security deceptive? It implied a simple world model–mutually trusting parties communicate and parties that do not trust each other do not. It concerned itself only with information security among mutually trusting actors. What we missed was that most of the communication on the Internet would be between parties that were prepared to communicate but did not know whether to trust each other. We agree to receive email knowing that it might be spam or have attachments that contain malware. We go to web sites even though we know (or should know) that web sites can download malware onto our computers. This is the space we need to make secure, not just the space of CIA communication among known, trusting parties.

In this context, the end-to-end principle is not wrong, just incomplete and in need of re-interpretation (for a reconception of the end-to-end principle in the context of trust, see [Clark and Blumenthal, 2011]). An analogy may help. If trusting parties want to send a private letter, they want assurances that the letter is not opened in transit. But if recipients suddenly realize that they may get a letter full of anthrax, then their "security objective" reverses–they want that letter opened and inspected by a trained, trustworthy (and well-protected) intermediary. End-to-end encryption between an attacker and his target is the last thing the target wants–it means that the target can get no help from trusted third parties in protection. An encrypted exchange with an untrustworthy party is like meeting them in a dark alley–there are no witnesses and no protections.

The overall security problem is not solved by telling the higher layer to use end-to-end encryption. Encryption addresses the problem of protecting communication between trusting users from disclosure or corruption, but fails to address the mirror problem of adversarial end-points using network protocols to attack each other. The problem of operation in an untrustworthy world has to be handled by involving the higher layers in the system, specifically the application layer, and it was this design problem that we neither clearly articulated nor explored how to accomplish.

In the next sections of this chapter, I look in more detail at the three classes of security, using the first set of categories above.

7.4 Attack and defense of the network itself

The physical layer of the Internet is made up of links, routers, servers, and the like. Routers and servers are computers, and thus potentially susceptible to remote attack using cyber-tools. Links themselves seem more immune to this sort of attack, and are mostly susceptible to physical attack based on close access–cutters and explosives. There are physical responses to these sorts of attacks: links can be hardened against attack (both against destruction and tapping), and routers can be placed in physically secure facilities.

The functional specification of this layer, as we normally conceive it, is rather weak: these components are expected to do what they are designed to do except when they don't. We know that links can fail, routers can crash, and so on, and it would be foolish to pretend that we expect these components to be completely dependable. But given this weak specification, how would we think about the security specification? I return to this question below, since the security analysis of this layer resembles the analysis of the layer above.

The next layer up, the Internet itself, is a global collection of links and routers, which serve to forward packets from an entry point to an exit point. The exit point is defined by an address that is specified in the header of the packet. While there are many details about what the Internet does, at its essence this is the functional specification. And the functional specification is again very weak. The service model has been called "best effort", by which

is meant that the network is expected to do its best, but failure is accepted. The network may fail to forward a packet, deliver packets out of order, deliver them multiple times, deliver them after inexplicable delays, and so on.

There is a well-understood conception of what "good service" would mean–acceptable levels of loss, delay and so on, but there is no hard and fast specification. The reason for that was clear in the minds of the early designers: a poor service is better than none. Designers should be held to a high standard of doing well at "best effort", but there are circumstances where best is not very good. If that situation were deemed "out of spec", then those times would be considered times of failure. However, there may be applications that can still make use of whatever function there is. So this weak specification is provided to the application designers, who then have to decide how much effort to put into adapting and compensating for circumstances where "best effort" is not very good. Some applications such as real time speech that depend on good packet forwarding service may themselves not function, or even attempt to function, when the packet forwarding is functioning poorly. Others, such as delivery of email, can struggle forward even if most of the packets are being lost. The higher layers just keep resending until eventually the data gets through.

In a system like this, each layer has to take into account the failure modes of the layer below in its own design. The Internet layer is designed to take account of link and router failures–it includes a dynamic routing scheme that finds new paths if a path fails. The end to end Transmission Control Protocol (TCP) copes with packet loss in the Internet. TCP numbers the packets, keeps track of which are received and which are lost, resends the lost ones, gets them in correct order, and then passes the data up to the next layer. So the overall resilience and function of the system is based not on precise specification but on a pragmatic balance of effort and investment at each layer. The better each layer, the less the layer above has to do, and (probably) the better the resulting behavior is. Different parts of the Internet can be engineered to different levels of performance and reliability (driven in many cases by pragmatic considerations of cost), and each layer above is expected to cope with this variation. Investment at the lower layers benefits all of the next layer functions, but over-investment at the lower layer may add unnecessary cost to the service. None of this is part of the Internet's "specification"; the interplay between performance and reliability at the different layers is a point of constant adaptation as the Internet evolves.

In this context, how would we characterize the "security" of the packet forwarding service of the Internet? A formally correct but useless response would be that since the network is "allowed" to fail, it need not concern itself with security. Pragmatically, of course, this is nonsense. There are well-understood expectations of the Internet today, and an attack that materially degrades that service is a successful attack. But it is a matter of degree. Degraded service may still be useful.[3] But with a loose functional specification like this, the determination of how to make the system resistant to attack is potentially ad hoc. One must look to the design mechanisms, not the specification, to see where attacks might come. Thus, one would look to the routing protocols, and ask if they are robust to attack (they are not, as I will discuss below). But the core function of the Internet is actually very simple. If there are links connecting routers, and the routers are working, and the routing protocols are computing routes, the Internet is mostly working.

The Internet provides a general service, useful for many applications in many circumstances. This generality raises a security conundrum: different contexts will face different security threats. There is no uniform threat model against which to design the network defenses. None the less, the security challenge must be faced; designers (and architects) must make pragmatic decisions about how robust the forwarding service must be to different sorts of attack. But any security analysis must begin with an assessment of the range of motivations behind such an attack, understanding that with high probability the motivation will be to carry out a subsequent attack on either an attached host or (more likely) an attack on communication.

[3] *Security experts understand that the most dangerous attacks are those that might cause massive, correlated failure of components, for example attacks on routers that exploit a common failure mode and take out so many routers that the dynamic routing algorithms of the network are overwhelmed and the network essentially ceases to function.*

A case study: Why is securing the network hard? Securing interdomain routing in the Internet

The challenge of securing inter domain routing in the Internet is a good case study of the barriers to better security; it illustrates the challenges caused by lack of trust and difficulties of coordination. The Internet is made up of regions called Autonomous Systems, or ASes. Each AS must tell the others which addressed are located within the AS and how the ASes are connected in order for the Internet to come into existence. The way this works in the network today is that each region announces the addresses that are in its region to its neighbors, who in turn pass this on to their neighbors, and so on, until this message reaches all of the Internet. Each such message, as it flows across the global network, accumulates the list of ASes through which a packet can be sent to reach those addresses. Of course, there may be many such paths–a particular AS may be reachable via many neighbors, and so on. So a sender must pick the path it prefers, or more precisely, each AS computing a route back to a particular set of addresses must pick among the options offered to it, and then offer that option to its neighbors in turn.

Originally, there were no technical security controls on this mechanism. That is, any rogue AS can announce that it is a route (indeed, a very good route) to any other AS in the Internet.[4] What may then happen, if other ASes believe this announcement, is that traffic is deflected into that AS, where it can be dropped, examined, and so on. This sort of event, in fact, is not uncommon in the Internet today, resulting in failures along all dimensions of CIA. How is it fixed today? Network operators monitor the system, problems of reachability are reported from the edge by end-users (who often have to resort to phone calls, since their systems cannot influence routing) and over some period, perhaps a few hours, the offending AS is identified and isolated until some suitable discipline can be devised.

Ignoring details, there might seem to be an obvious technical fix. Why are these announcements not signed, using some sort of cryptographic scheme, so that they cannot be forged? Indeed, this was the path down which the designers started when they set out to secure interdomain routing. But there are two formidable barriers to this, one having to do with trust and one have to do with migration to the new scheme.

The migration problem is easy to understand. In the global Internet, there is no way that everyone is going to switch to the new scheme at once. Unless some draconian discipline is applied (disconnection from the net), some actors may just refuse to undertake the effort of upgrading, and they will continue to originate route assertions that are unsigned. There are two options to deal with this. One is to reject them (which is the draconian outcome of disconnecting them) or accept them, in which case a malicious actor cannot be distinguished from a lazy actor, and we are essentially no better off. Until the last AS converts, we get little value from the scheme, unless we wrap it in complex high-level systems, such as globally distributed, trustworthy lists of ASes that have converted, so that a router knows which unsigned assertions to accept.

The issue of trust is a little more complex. When an AS signs an assertion (for example, when MIT signs the assertion that it is AS 3, and that it has a particular set of addresses that it holds within that domain), it has to use some encryption key to sign that assertion. The obvious technical approach is to use a public or asymmetric key system, where MIT has a private (secret) key it uses to sign the assertion, and a public key it gives to everyone so they can decrypt the assertion and confirm that MIT signed it. So far so good, but where does that public-private key pair come from? If MIT can just issue itself a set of keys and start signing assertions, it might seem that we are no better off, because a malicious actor could do the same thing–make up a public-private key pair and start signing assertions that it owns AS 3, controls those addresses, and so on. To prevent this from being effective, the technical proposal was to create a trusted third party that could confirm, based on its own due diligence, which public key is actually associated with the real MIT. But why in turn would anyone trust that third party? A scheme like this ends up in a hierarchy of trust, which seems to require a *root of trust* at the beginning, a single node that all parts of the Internet trust to tell them which second-level parties to trust, and so on until we get to

[4] *It was understood as early as 1982 that an AS could disrupt routing by making a false statement. RFC 827 [Rosen, 1982, Section 9] says: "If any gateway sends an NR message with false information, claiming to be an appropriate first hop to a network which it in fact cannot even reach, traffic destined to that network may never be delivered. Implementers must bear this in mind." The situation was identified as a vulnerability but not a risk. The advice to "bear this in mind" could have multiple interpretations.*

the party that asserts that it know who the real MIT is.

An engineer might think this was a simple, elegant scheme, but it runs agound in the larger world. First, what single entity in the world would all the regions of the world agree to trust? The United Nations? This issue is serious, not just abstractly but very concretely. When this scheme was proposed, several countries (including Russia) asserted that they would not assent to a common root of trust with the U.S. The agent who has the power to validate these assertions must, almost of necessity, have the power to revoke these assertions. Can we imagine a world in which the United Nations, by some sort of vote, revokes its trust assertion about some nation and essentially ejects that region from the Internet? What about those second-level entities, that almost certainly are within some legal jurisdiction and thus presumably subject to the legal regime of that region?

This fear is not hypothetical. The institutions that allocate Internet addresses are the Regional Internet Registries (RIRs). The RIR for the EU is RIPE, and is located in Holland. The Dutch police brought a police order for them to revoke the addresses of an AS. RIPE correctly said that it did not have the technical means to revoke an allocation. However, if they were issuing certificates of authenticity for AS allocations, then they would no longer be able to make that claim.

So does this scheme make the Internet more stable and secure or less? Once people understood the social consequences of this scheme, there was substantial resistance to deployment. The problem with adding a "kill switch" to the Internet is to control who has access to it.

A different design approach might mitigate these concerns, one that allows actors (e.g., ASes) to make assertions about who they are, but validates these assertions in a way that makes them very hard to revoke. That would solve the "jurisdiction" problem. But if a false assertion ever got started, how could it ever be revoked? Once we grasp the complexity of functioning in a space where not all the actors share the same incentives, not all are equally trustworthy by different measures, and that these actors of necessity are in the system, it becomes a very difficult problem indeed to design a system that is robust at ejecting actors that are "bad" but also robust at not ejecting actors that are judged "bad" if we don't accept that they are bad. Management of trust relationship, and the expression and manifestation of those relationships, becomes the defining feature of a successful scheme, not exactly how crypto is used.

So in this respect, the landscape of security becomes a landscape of trust–regions of mutual trust will be more connected, more functional, more effective, and regions that don't trust each other will still try to communicate, but with more constraints, more limitations, and perhaps more failures, especially with respect to availability. And this pattern will be found within any application that tries to tailor its behavior to the degree of trust among the communicating parties, whether the function is exchange of routing information or email.

What happens today is that "the Internet" does not try to solve these problems using technology. We fix some of these problems using management–oversight of the system by trained operators and managers. We just tolerate some of the residual consequences.

An alternative to the scheme described above to secure the AS routing system will illustrate how a different scheme fits into a socio-technical architecture. The scheme described above, with a hierarchy of trusted certifiers and a single root of trust, is technically robust, in that it will always give the right answer *if the trust relations are valid and accepted by all the parties.* This approach may be technically robust but is not socially robust. Here is an alternative approach that is less technically robust (one cannot prove that it will give the correct answer under certain assumptions) but is more socially robust. Above, I rejected the idea that MIT just make up a public-private key pair and start signing its assertion. What would happen if that scheme were adopted? At first, various regions of the Internet might get conflicting assertions, if it happened that there was a malicious actor in the system at the time when the assertions started to be signed. That situation, while not desirable, is exactly what we have today. But over time–days or weeks–it would become clear what key went with the "real" MIT. Each AS in the network could learn this for itself, or groups of mutually trusting ASes could cooperate to learn it. If necessary, the public key could be exchanged by side-channels. Once the other ASes in the Internet have decided which key to trust, they have independent possession of that fact, and there is no authority that can compel a third party to

invalidate it. The scheme decentralizes control: any AS can decide on its own to stop forwarding traffic to MIT, just as they can today.

What this scheme exploits is not a technical scheme for propagating trust, but a social scheme called "getting to know you", which humans have been running, probably for millions of years. We can be fooled, but in fact we are pretty good at it. And it is simple. It requires no trusted third parties, little administration (except that each AS should try very hard not to lose their own private key) and great adaptability to changes in the landscape of trust.

7.5 Attacks on network communication

This category of attack relates to parties that are attempting to communicate across the Internet, and are being thwarted by some malicious action. This category can be decomposed using the traditional three CIA sub-objectives: confidentiality, integrity and availability: information should not be disclosed except to parties authorized to see it, information should not be corrupted, and it should be available. With respect to network communication, these goals take a rather simple form–particularly with respect to integrity. Since the current Internet does not perform computation on content, the simple form of integrity is that data is transmitted without modification. [5]

As I discussed above, cryptographic algorithms fit into the CIA triad by giving strong assurance that data is not disclosed, and strong indications if data is modified. There are several well-understood contexts in the Internet today in which encryption is deployed, such as IPsec and TLS. Cryptography is a powerful tool to improve security. However, it is important to see how cryptographic methods tend to function in the larger context. These protect the user from failures of integrity by halting the communication. They map a wide range of attacks into a common outcome–cessation of communication. But this outcome, while potentially better than a failure of confidentiality or integrity, is just a failure along the third CIA dimension–availability. Essentially what these schemes do is turn a wide range of attacks into attacks on availability. And that is not the desired outcome–we want to offer assurances about all dimensions of CIA.[6]

If the best we can do using cryptography is to turn a range of attacks by untrustworthy actors into attacks on availability, what can we do to improve the situation? There are two ways to try to deal with untrustworthy actors: constrain or discipline them, or reduce our dependency on them–in the limit avoid them. Imposing constraints on untrustworthy or malicious actors that both compel them not to misbehave and as well compel them to perform at all are hard to devise; the *fail-stop* semantics of attack detection is the common outcome. The only way to compel correct operation is to so design the larger ecosystem so that the cost to the actor from expulsion from the system outweighs the cost from foregoing malicious behavior. This might work for an ISP who is hosting both legitimate customers and spammers (and ISPs have been expelled from the Internet for hosting spammers, essentially driving them out of business), but malicious individuals show great resilience to constraint and discipline, especially across region boundaries. This leaves the other option as a path to availability: accept that untrustworthy actors are in the system but avoid them.

Traffic analysis

The term *traffic analysis* describes a form of surveillance in which the observer does not look at what is being sent, but the source and destination of the communication. Most obviously, in the Internet context, the observer capture the IP addresses in the packets. This sort of logging, sometimes (for historical reasons) called pen/trap logging, has its roots in the logging of telephone numbers on phone calls. From a legal perspective in the U.S. (and many countries) it is easier to get a court order allowing pen/trap logging than data logging, which has led to a set of legal debates about what sorts of data can be gathered using pen/trap logging. Such data is often called *meta-data* because it is data about other data. The complexity of the Internet makes the distinction between data

[5] *If PHBs are added to the network that transform the data in transit, a more complex theory of integrity will be needed, such as [Clark and Wilson, 1987].*

[6] *This observation provides one explanation as to why so many users deal with dialog boxes warning about potential hazards by clicking the "proceed anyway" option–what they want is to make progress. Another reason, of course, is the often inexplicable content of those warnings.*

and meta-data contentious: since a packet is a sequence of headers, each with information about the next header, one layer's meta-data is another layer's data.

From a technical perspective, encryption can limit what can be seen in the network, but the headers that are processed by the routers (and other PHBs) must be visible (barring very complex uses of encryption, such as TOR), so there seem to be limits to the extent that a network design can substantially shift the balance of power with respect to traffic analysis. One exception to this presumption is the NDN proposal, which (though the use of per-packet state in each router) removes from the packet any source address. This means that an observer can tell that a piece of information has been requested, but cannot easily tell which source requested it.

It turns out that a great deal of information can be deduced by observing an encrypted stream of packets [Chen et al., 2010, Wright et al., 2008] [[[@@What to cite? there is so much]]]. It is possible to deduce a great deal about what is being communicated, a great deal about the communicants and so on. This so-called *side channel leakage* is a serious problem in high-security contexts, and may be a serious problem for typical users as tools for analysis improve. So while encryption may protect the data in transit, the idea that encryption protects the communicating users from harm related to confidentiality should be viewed with some skepticism.

One way to limit the harms from traffic analysis is to avoid routing packets through regions of the network that are more likely to practice this form of surveillance. There is no obvious way to detect in real time that packets are being subjected to traffic analysis, but if a group of users can make a judgment about which regions of the network are less trustworthy, and have some control over routing (similar to the control discussed above in the context of availability), they may be able to somewhat mitigate the peril.

7.6 Attacks on the attached hosts

Today, we see a wide range of attacks in this category, ranging from attacks that involve a malicious sequence of packets sent to a machine that was not a willing participant in the communication (an attack that exploits an unintentional open port, or a flaw in the network software and the like) to attacks that use an intentional act of communication (receiving email or going to a web site) to download malicious code.

Again, it may be helpful to return to a historical perspective to understand the current situation with respect to these classes of attacks. As I said above, there was a presumed division of responsibility: the network protected the network and the host protected the host. Security thinkers of the time did not believe it was responsible to delegate protection of the host to the network, because they had no reason to trust the network. The assumption was that the designers of operating systems could and would develop systems that were resistant to attack. Given this presumption, the job of the network was simplified: it delivered whatever the sender sent, including possible attacks, as efficiently as possible. The host sorted out what it did and did not want to receive.

In fact, this description of the network and the host is actually an over-simplification of how early security experts thought about secure networking. The security experts that consulted in the early days of the Internet were primarily from the military/intelligence community, and had as a primary concern confidentiality–preventing disclosure of classified information. This mind-set shaped some of the early deliberations about Internet security. As I noted above, this framing of security tends to ignore the issue of communication among parties that do not necessarily trust each other. As well, this framing tends to divide the world cleanly into trusted and untrusted regions of the net. In the context of classified work, it made sense to accept that there were trusted regions of the network, typically inside facilities where users had clearances and computers could be trusted. These regions might be connected together over a public, untrusted Internet, but in this case the packets across the public internet would be encrypted and wrapped in outer IP headers that only delivered the packet to the distant trusted region. This concept, called *encrypted tunnels*, made sense from a technical perspective, since only one encryption device would be needed at the interconnection point between the trusted region and the public Internet. At the time, encryption boxes were expensive, and even a point-to-multipoint device was pushing the state of the art. Having such a device per host was not practical. The concept also made sense in the security calculus of the day. There was no way an untrusted computer on the public Internet could make a connection to a trusted computer

in a trusted region, because the encryption device would not allow in a packet that was not encrypted at another region. End nodes did not need to worry about being attacked, because within the trusted region the prospect of attack was discounted, and from outside the region packets were totally blocked.

The security analysis of this sort of architecture became quite sophisticated. There were concerns about the possibility that corrupt insiders could leak information by hiding it in "covert channels", low bandwidth communication channels exploiting such features as the timing of packets in the channel. The *confinement problem* was understood in 1973 [Lampson, 1973]. These concerns did not end up being the real threats, and a focus on this framing may have distracted the early thinkers from a broader consideration of the security landscape, such as the need for users with clearances to talk to people without such clearances.

This simple division of responsibility has proved flawed, for several reasons. First, of course, the operating systems of today are flawed. Second, application designers have favored functionality over security, and designed applications with rich features (e.g., the ability to download and execute programs of various sorts), so the applications become the vector of attack. Very early on in the design of the Internet, security experts (some from the NSA) identified the problem of a "trojan horse" program, and made clear that in their opinion, if executable code was transferred across the network, the only practical protection would be to transfer it from trustworthy sources–trying to vet code for malicious content was a losing game.

So here we are today, with a need to reconsider more or less from scratch all of these assumptions. First, we have started to depend on (in other words, to trust) at least some elements in the network, as I discussed in Section 4.6. Firewalls provide a crude protection from the attacks that involve packets sent to a machine that did not want to participate in the communication. Firewalls block unwanted ports, and (if combined with Network Address Translation) hide the IP addresses of machine. For this protection to work, we must depend on the reliable operation of the firewall, and rely on the topology or routing of the network not to bypass the firewall. This sort of trust is both simple and local, but it reflects a recognition that the hosts being protected and at least the local region of the network to which they are attached should share responsibility for protection. The next question is what services might the packet forwarding layer provides to make the "security job" of the host and the higher layers easier. This question is the one I asked in Chapter 4–how can the expressive power of the network be designed to help the defender in the case that the interests of the end-points are not aligned. The network cannot make the end-points "secure", but perhaps it can be a part of the solution, rather than delivering the attacks with best effort.

A more general question that one might ask, in this light, is if the host must depend on other elements as part of its protection, which elements are better suited for this task? Perhaps depending on the network (or even more specifically the region of the network that provides service to the host), is not the only or the best idea. Perhaps there are new sorts of elements, or new actors that could provide protection services. In the language of Chapter 4, what PHBs can we devise to protect an end-point, what actor would be best trusted to deploy and operate them, and finally what architectural support, if any, is needed to utilize them. If we allow ourselves to rethink from scratch this framework for security, new design approaches might emerge that have additional actors and services beyond the host and the network.

As well, there has been considerable progress in devising ways that the operating system of the end-node, in addition to being more robust itself to attack, can help protect the application running on the end-node from attack. The concept of *sandboxing* describes an approach where the code of the application is placed in a confining environment before it interacts with the network, and this environment is conceptually discarded at the end of the interaction, thus discarding in passing any malware or other modifications that may have resulted from the interaction.

The role of applications

A network such as the Internet, as I have repeatedly stressed, is a general network that moves packets. But packets only flow because some higher layer software chooses to send and receive them. It is applications that define what actually happens on the network. It would be nice if the packet carriage layer of the Internet, perhaps properly

augmented by innovative PHBs, could protect one host from attack by another independent of the application being used, but this is not a realistic hope. The simple semantics of the Internet–best-effort delivery of packets–is (for the moment) about all the network can do. It is the higher-layer software–the application–that translates between information sent over the Internet and actions on the end-nodes. Many of the security problems we deal with today arise because of design decisions at the application layer, and it is to that layer that we must turn for an overall improvement in the landscape of security. Applications can, by their design, either create security vulnerabilities or limit them.

The lesson we learn by looking at the design of applications is in some respects a bleak one. Applications today, in the pursuit of more powerful functionality and appealing features, have incorporated functions that are known to be risky, and were known to be risky at the time they were designed. The ability to download active code (e.g., Javascript) from a web site and execute it on a client machine was understood as risky from the beginning, and was decried by the security community at the time. It was implemented anyway. We must accept that applications today are *insecure by design*, and we must figure out how to deal with this, since this preference is not going to be reversed.

One answer lies in the operating system, where features such as *sandboxing* can potentially prevent malicious code from having any persistent consequences. Another answer may lie in designing applications so that they only enable risky modes of operation when there is good reason to trust the communicating parties. Since applications define and control the patterns of communication among the entities, it is applications that can, by their design, invoke PHBs as part of their security architecture. And it is applications that can tailor their behavior based on the extent to which the participating actors trust each other. Actors that choose to trust each other may want to exploit applications in a mode that imposes fewer constraints and allows more flexible communication, while actors with less mutual trust may want a mode that provides more protection.

Applications can play another key role in an overall framework for security. My analysis to this point has swept a serious problem under the rug. What if our attempts to protect the host from attack fail, and the host falls under the control of a malicious actor. At this point, that malicious actor may undertake activities (e.g., data transfers) that seem entirely legitimate with respect to the network (they seem like transfers between mutually trusting parties), but the security goal is to block them. In other words, in the case where a machine has been compromised, the security goal reverses. The goal is to "attack" (block) what otherwise would be legitimate communication.

Perhaps some cases of this sort can be classified as malicious by behavioral monitoring–a user who suddenly transfers gigabytes of data out of a secure area might attract attention in any case. But in general the way to think about this situation is to distinguish, as I did earlier, between penetration of a host and a harm. The harm arises from the use of applications, which define the legitimate data flows. Applications can be designed so that they reduce the risk of harms, using designs that require (for example) multiple machines to concur before potentially dangerous actions are permitted. Consider, for example, that a firewall might block all outgoing data flows above a certain size unless a second machine has first authorized the transfer. Applications could be designed such that this second machine is notified of the need to issue this authorization, which could then carry out some independent check of identity, authorization and the like. The design goal would be to minimize disruption of normal work flow, but that second machine should be implemented in such a way that the penetration of the first machine by a malicious actor does not provide a means to penetrate or subvert the function of the second machine.

What I have described here is a sophisticated design challenge for the application designer. But suggesting that potentially dangerous tasks should require dual authentication is not a novel idea. My point is that this sort of constraint is going to be built into, or at least controlled by, the application, not the network. I discussed earlier the basic design approach of the Internet, which is that layers must be designed to deal with failures in the layers below. TCP deals with lost packets, and so on. What I propose here is just the application of this approach at a higher layer–the design of the overall system must take into account the possibility of failure in the layers below, in this case corruption of machines on which (part of) the application is running. The network does have a role, which is to insure that only authorized flows take place. One could imagine using software defined network (SDN)

technology to allow only flows that are consistent with the application-defined security policies.

The role of identity

My repeated reference to trust seems to beg a more basic concern–it is nonsense to talk about whether actors trust each other unless they have sufficient information about each other's identity. So identity management must be part of any framework that depends on trust management. This fact, in turn, raises the question of which entities or layers within the system should implement the mechanisms of identity management.

One view is that the architecture itself should specify how identity is managed. There have been calls for an "accountable Internet", which seems to imply that the architecture assures the identity of the participants to all interactions. I think this is a very bad design approach, as I have argued, along with my co-author [Clark and Landau, 2011]. Identity is used in a very nuanced way in society–sometimes we need strong, mutual confirmation of identity, sometimes we function well with total strangers. It is the mode of interaction that determines the need for identity, and on the net, it is the applications that define the modes of interaction. So we must turn to and rely on the applications to establish the correct level of mutual identification, and use this information to deploy the correct level of protection.

Application designers should not have to solve these problems from scratch each time a new application is designed; what is needed is advice and guidance, perhaps applicable to a class of applications, that suggests how these problems might be approached. What is needed is a collection of *application design patterns* that can be offered to designers. Trying to think about design patterns in an organized way should yield another benefit; by looking across applications to see common needs, new ideas may emerge for common services that the lower layers can offer to help improve the security of applications. It is highly unlikely that there will be some new service at the packet forwarding layer that can suddenly make applications secure, but is is possible that there are supporting services that can make the task easier. The way to find these services is to look at application requirements, generalize from them, and see what concepts emerge.

7.7 Denial of Service attacks

Abstractly, the fact of DDoS attacks can be taken as a fundamental indictment of the architecture–the essence of a layered design is that the lower layer should not be affected by the behavior of the layer above it. Since DDoS attacks can disrupt the lower layer just by sending packets, the design of the lower layer is definitionally flawed. However, one should not be too harsh in judging the design. Simple approaches to protecting the transport layer, such as fair queuing and rate limiting, can only do so much if the attacker can assemble a large fraction of a million attack machines. [[[Cite Perman thesis to put "simple" into context???]]]

Another point of view is that the architectural flaw is that a sender can send at will, without the permission of the receiver. If the Internet required the permission of the receiver before delivering a packet, perhaps certain sorts of DDoS attacks could be thwarted. However, many of the machines that are attacked are intended to provide services to any comer–services like providing Web content. These machines need to accept traffic from anyone if they are to fulfill their intended purpose.

Another perspective on DDoS attacks is that they persist only because of an "economic flaw" in the ecosystem–the state of affairs that in many cases, users pay a flat rate for access rather than a usage-based charge. This pricing model reduces the incentive of the user to remove malware–if the user suddenly got a large and unexpected monthly bill, there would be a much larger incentive to remedy the situation.

In my view, once we take into account the need of services (server machines) to be open to connections from anywhere, and the potential scale of DDoS attacks, the only realistic approach is to identify the attack traffic as such so it can be stopped or at least throttled to an extent that renders the attack ineffective. (The idea of diffusing the attack against a sufficiently large attack surface also seems to have merit.)

However, once we contemplate the idea that certain traffic will be classified as malicious, we must then think through the potential of this mechanism as itself a vector of attack. We must ask which entity would have the authority (or be trusted) to declare traffic as malicious, and which actors would be expected to honor this

declaration. Again, this is an exercise in crafting the expressive power of the architecture so that the "right" actors can preferentially exploit that power. I return to this topic when I discuss architecture and security in section 7.9.

7.8 Balancing the aspects of security

The preceding discussions suggest that there are four general problems to address: protect regions of the network from being attacked, protect communication among aligned parties, protect parties with adverse interests from harming each other, and mitigate DDoS attacks. It would be very nice if these could be addressed independently, and to some extent they can, but I will argue that there are tensions between protecting the host and protecting the communication, and part of the overall security design of an architecture will be to balance to requirement to protect communication and the requirement to protect end-nodes from each other.

One could imagine the design as proceeding as follows:

- First, make sure that critical algorithms that support key PHBs are secure. Interdomain routing is the obvious example–since routing, as currently conceived, is a distributed algorithm in which all regions of the network participate, it creates opportunities for one region to attack another. There are other PHBs, such as anycast, that may need to be better secured.
- Second, put in place schemes to protect communication. Assume that applications (which define the patterns of communication), will deal with issues of confidentiality and integrity by using encryption, and assume that the application will intentionally route communication to any service elements that are needed. To deal with the goal of availability, the application must design its communications, and take advantage of any expressive power provided by the system, to detect and localize if and where a PHB or service component is mis-functioning, and reconfigure itself to avoid it.
- Third, put in place PHBs that can prevent or constrain communication among untrusting or hostile end-points. Assume that the application can modulate its behavior based on sufficient identity information, and add or remove protective PHBs as necessary.
- Fourth, put in place suitable mechanisms to diffuse or disable DDoS attacks.

This assessment is both glib and incomplete. It is glib overall in that is seems to trivialize very hard tasks, even if they are well-defined. In more detail, it is glib, firstly, with respect to the issue of localization of malfunction and availability. However, since the current Internet does nothing in this respect, any new capability would be better than what we have today. Second, it is glib with respect to the degree it depends on the designer of the application to get all this right. For this approach to work, the application designer has to be given a lot of help and design guidance, even if this is not embedded in the architecture.

However, if this analysis is well-structured, it can suggest a research approach, even it it seems to trivialize the challenges. However, I also described it as incomplete. The lists begs the question of whether these tasks are independent–whether we can proceed with each separately, doing the best we can at any time. In fact, I believe that they are not independent; it is possible that the design space of secure operation implies a tradeoff between two perils–attacks on the communication and attacks on each other. The more protections that are put in place to protect one end-point from the other (in the language of Chapter 4 the more PHBs), the more points of attack are created that might be used to disrupt the communication. A clean, encrypted channel between two end-points is a very simple concept, with few modes of failure and few points where an adversary can exploit a PHB to disrupt the communication.

If untrustworthy PHBs can indeed be ejected from the path of communication (task two above) then perhaps this risk is hypothetical. But we see today in parts of the Internet a situation that brings this issue into sharp focus–the situation within countries with more repressive or restrictive governments who require that their ISPs act as agents of the state to regulate communication. In this case there are PHBs in the network that are, from the perspective of the users if not the state, untrustworthy, and the users have no ability to avoid using them.

For the users in that country, there is no way to avoid using that network (it may be the only network available) so communication becomes a "cat and mouse" game in which the expressive power of the network is used by both sides to achieve their goals. A sender can encrypt as much as possible, so that the only PHB it attempts to exploit is the most basic forwarding. The PHB of the state may try to force more revelation by blocking encrypted packets. The sender may tunnel to a exit node; the PHB may respond by blocking those destination addresses. By revealing less, the sender tries to prevent the PHB from doing fine-grained discrimination–it forces on the PHB a "blunt instrument" response, such as blocking all encrypted flows, which may have such collateral damage that the censor is forced to forego that behavior. So part of what an architecture (through the lens of *expressive power*) can do is provide tools to shape this game, and perhaps bias the outcome. This is a classic example of *tussle*, carried out using the tools of the architecture.

In this context, rich expressive power may be intrinsically dangerous. If a network provides rich expressive power, and applications are designed to take advantage of these powers, even as an "optional" mode, a PHB with adverse interests may take the approach of blocking modes of communication that do not exploit these options, on the grounds that the collateral damage is reduced: the user still has a mode that will achieve communication, but one that forces maximal revelation. User choice is also dangerous. One can see a simple example of this with respect to encrypted connection to Web pages. Today a Web server makes the decision as to whether to use TLS. The client has no control. If a censorship PHB blocks encryption, it blocks access to all TLS web sites. But if the use of TLS were a choice under the control of the client, blocking all encryption would "only" have the consequence of forcing the user to communicate in the clear. Choice can be a bad option if the end-node can be coerced into making bad choices. Better a simple architecture with no choice.

Expressive power is dangerous in another related way. As we add capabilities for the sender to add more expressive explicit data to the packet header, the possibility arises that third parties "in the network" (exploiting topological delivery), with interests adverse to both the sender and receiver, and with topological control, will be able to use the available mechanisms to coerce explicit information from senders as a condition of usage. Section 7.9 gives an example of this tension related to identity management. So when the goal is protecting communication from attack by the network, the best design point may be minimal expressive power with no choice given to the end-nodes over how that expressive power is exploited. This approach, almost of necessity, will make the target of availability much harder to achieve.

I have used the terms *control point* and *control point analysis* to describe a way of looking at the design of a system–an approach that involves cataloging all the actions that must be completed in order for some undertaking to succeed, and methodically cataloging all the points in the flow of actions where the design creates an opportunity for some actor to control that action. A focus on control points guides the analysis away from the data plane and toward the control plane. Control points are points of tussle, and what we have called tussle is the aspect of security that emerges when an undertaking must tolerate the presence in the system of actors with adverse interests. In this context, eliminating control points, or diffusing the architecture of control so that it cannot be co-opted by an actor with adverse interests, may be as important to the overall usability of the architecture as adding more complex network functions that are intended to improve the functional performance of the network. [[[Perhaps add a box on control point analysis…?]]]

7.9 The role of architecture

The preceding sections propose a way to view the landscape of network security.The focus of this book is architecture; the final question for this chapter is what does architecture have to do with security. According to my minimality argument, architecture should say as little as possible, but no less. And architecture does not, in and of itself, determine how a system meets its requirements (such as a requirement for secure operation), but rather provides the framework and necessary starting points for the subsequent design to meet its requirements. The previous discussion about expressive power and its potential dangers suggests a starting point of view, but here are some more specific concepts.

Attacks on the network

I discussed in Section 7.4 why securing the routing protocols of the current Internet is not a simple technical problem solved by the use of encryption, but a complex problem embedded in a space of trust management and tussle. To the extent that designers add complexity to the network (for example additional PHBs with distributed control algorithms), I would assume that the failure modes and attack modes will become more complex–an argument for simplicity. But at the architectural level, the question is what sort of expressive power might be added to make such protocols more robust, or aid in localizing faults in the protocols. Perhaps the methods used to detect malicious PHBs in the context of end-point communication can be used to detect malicious PHBs in the context of the network's control algorithms. Again, the role of architecture is not to make the system secure, but to provide critical building blocks so that subsequent mechanisms can be built to achieve these goals.

One key design choice with important security implications is the expressive power in the packet header to represent interdomain routes. Schemes like those in Nebula and SCION in XIA allow the sender to put a cryptographically signed interdomain source route in the packet. The *pathlet* proposal in [Godfrey et al., 2009] similarly requires the packet header have sufficient expressive power to describe a sequence of pathlets. In schemes like these, the sender or its agent composes the path from routing assertions made by the different regions of the network. The resulting delivery sequence can still be confounded if these basic routing assertions are not trustworthy, but the sender need not worry about whether the computation that composes the resulting source route is corrupted, since that computation is done by the source or an agent the source has reason to trust. In BGP, where the interdomain paths (the path vectors) are computed by each AS along the path in turn, any corrupt AS can disrupt the proper delivery of packets to the destination, leaving no option to the source to route around that AS.

Protecting PHBs and their invocation Apart from the tension I describe above, if the network is to contain a richer selection of PHBs, perhaps invoked using explicit parameters in the packet, the design must take into account protecting both the PHBs and the parameters of the packet header.

Once we recognize the existence of intermediate elements (and their PHB) as a part of the system, we have to take methodical steps to deal with attacks on these devices themselves. These devices are perhaps (often?) simpler than general purpose operating systems, and it may be possible to engineer them to a higher standard of resistance to penetration attacks. To the extent that these are protective PHBs–"first line" elements exposed to the full open Internet, while shielding resources behind them, it will be necessary to engineer them to a high standard of penetration resistance. More generally, we have to ask about DDoS attacks against these devices. Intermediate elements with their PHB will be attacked if this provides a way to disrupt access to the overall service. So the ability to protect first-line elements from DDoS attacks is a general problem the architecture should solve. Mechanisms such as anycast may be useful as a tool in this pursuit, so long as the issues of shared state across the replicated PHBs can be managed.

Once we introduce the concept of explicit data carried in the packet and used as input to various PHBs in the communication path, we have to ask about the security implications of this data. The classic triple of "confidentiality, integrity, availability" is a useful place to start, but we must not forget the concerns around traffic analysis. Another summary is "that which is not encrypted can be seen, that which is not signed can be changed".

For example, the proposal for a push-down stack of records of explicit data for different PHBs, as I sketched in Chapter 4, reveals a number of issues. The problem of gross corruption of the header by some hostile PHB is perhaps not worth considering–if an element is that malicious, the outcome is the same as a failure to forward, which is a more general problem that can only be dealt with by avoiding that element. The more interesting question is spying on the information, or more purposeful modification of information on the stack, to somehow break a PHB further along the path. To prevent this, in the extreme, each record on the pushdown stack could be encrypted using a public key of the element in question. This implies considerable processing overhead, and some way to get the right public key reliably. The Nebula proposal realizes a scheme of this sort. The overall complexity

is somewhat similar to a client using the TOR system, so we do have evidence that users are willing to tolerate this overhead. However, in some cases, it may be desirable for one PHB to modify the input data for a subsequent PHB, so the approach taken to secure the data should not be inflexible.

Concerns about corruption of a packet header are in fact only an example of the more general problem I discussed above–if an intermediate element is untrustworthy, in the general case the only option is to reduce ones dependency on it to a sufficient degree, perhaps avoiding it all together. This approach depends as much on the ability to detect and localize a problem as to prevent it. So the design approach that the architecture takes around explicit parameters in the packet header should focus on fault localization as well as continuing to employ elements with adverse interests. Again, the context of the repressive government must be a cautionary thought at this point.

In the discussion on applications above, I proposed that applications might want to adapt their modes of operation based on the degree to which the end-points were prepared to trust each other. The reality of operating in a network where there are untrustworthy intermediate elements suggests that there is a second dimension along which the end-nodes will need to adapt their behavior, which is the extent to which they choose to try to exploit intermediate elements and their PHBs. The less explicit information that is in the packet (the less the end-points try to exploit the expressive power of the architecture), the less opportunity is available to adverse elements to disrupt communication.

Protecting addressing modes Addresses in packets seem like the most basic form of explicit parameter, so they make a good case study of the tradeoffs as we add expressive power. Addresses in the current Internet are very simple: just one field of 32 bits (until we get IPv6). This simplicity and lack of structure imposes some operational requirements: we organize addresses into blocks for routing purposes, rather than computing routes for individual end-points. These address blocks are usually subsets of addresses that belong to an Autonomous System, so a possible form of address with more expressive power is that the address contain the destination AS explicitly, as well as an end point address. If the AS is not too big, routing on flat addresses within it would be feasible, so addresses would not have to reflect location. This proposal is a nice example of a different sort of expressive power in the header; the challenge with this idea, as with all proposals for expressive power, is to ask how it can be exploited by malicious actors.

Attacks on the routing system would seem to have about the same potential in this scheme as in the current scheme, in that whether the AS number is explicitly in the packet or derived from a mask on the IP address, a bogus routing assertion could equally send the packet off in the wrong direction. If an attacker could somehow modify the AS number in a packet, it could be sent off in the wrong direction without any attack on the routing system, but it is not obvious how an attacker would undertake that attack unless the network of the sender is untrustworthy (as in the censorship example).

Perhaps more interesting is the question of how a malicious sender could manipulate this scheme as part of an attack. The opportunities for attack begin to emerge when we look at the options that the receiver might exercise to deploy PHBs to protect itself. For example, a machine that is protecting itself from a DDoS attack might purchase a service that provides many machines scattered around the network to diffuse the DDoS attack, and let only legitimate traffic through. To direct DDoS traffic to these widely distributed machines, they might be given addresses that are in an AS which itself is multi-homed onto the network at many places, a form of AS-level anycast. The receiver might give out its address as being inside that DDoS protection AS, but if an attacker can guess the final AS within which the target is ultimately located, it can compose this address and send directly to it, thus using the increased expressive power of the header to bypass the DDoS protection. The DOA proposal explicitly noted that some additional mechanism such as having the protection node sign the packet would be required to protect against attackers that attempt to bypass the node.

One could try to mitigate this attack by structuring the DDoS system as an address indirection scheme, in which the DDoS protection devices rewrite the destination address or add some sort of capability to the packet, to signal

that the packet has been validated. In addition to the i3 and DOA proposals mentioned above, there have been other schemes proposed: [Andersen, 2003, Yang et al., 2005] to improve the security of services on the network by interposing some sort of checkpoint or intermediate relay in the path from the client to the server. This relay can permit or deny access based on the rights of the client, or perhaps rate-limit or otherwise constrain clients as appropriate. These devices depend, in general, both on state stored in the relay and additional information in the packets. Since today there are no fields to carry additional information, such schemes are required to build on such fields that already exist (implicit parameters, to use my term). Perhaps an important form of expressive power to add to a future network is some sort of credential to control forwarding of packets among PHBs, or some other mechanism to compensate for more expressive addresses by limiting the uses of these addresses as attack vectors. But more generally, the consequence of defining an address format with more expressive power is the need to add yet more expressive power that can be used to control the abuse of the address, which might in turn trigger the need for expressive power to limit that mechanism from being abused, and so on.

A simpler example of tension over expressive power is *anycast*. With anycast addressing, a number of endpoints can have the same address, and the network rather than the sender picks the one that receives what is sent, perhaps the "closest" receiver by some metric. What if the particular receiver selected by the network is not functioning correctly? Perhaps a malicious actor joints the anycast address group and tries to attract traffic in some region. If the sender has no way to exercise choice and select another receiver, this is a classic example of an availability failure due to lack of control by the sender. All the sender can do is wait for the failing receiver to be fixed or removed. The DONA proposal allows the requester to ask for "the k-th closest" copy. But if the architecture gave any form of control to the sender, would this expressive power become an attack vector for a malicious sender to pick a particular receiver out of the anycast set and launch a DDoS attack? Part of the power of an anycast address is to diffuse a DDoS attack, which implies that the sender must not be given any power to tailor the address. Is this tradeoff intrinsic?

Attacks on communication

Assuming that confidentiality and integrity are managed using encryption, the remaining problem, availability, I defer to Chapter 8. The potential consequences of traffic analysis can be greatly influenced by architecture design. The expressive power (or lack thereof) of the header can greatly influence what is exposed in the packet, and thus the perils of adverse observation, as for example the lack of a source address in NDN.

Attacks on hosts

I argued that to a large degree, the opportunities for attack are created at the application layer, and the application (supported by mechanisms in the end-nodes like sandboxing) will have to mitigate these risks. What can the network, and in particular the network architecture, do to help mitigate these problems? One answer is that the network can provide means to prevent data flows that are not authorized by trusted elements in the network. If applications are designed so that authorization from trusted elements is obtained before dangerous data flows are permitted (e.g., exfiltration of data from a trusted region), then the network should prevent rogue applications (perhaps based on malware) from initiating these flows. Mechanisms such as Software Defined Networking (SDN), which allow forwarding policies to be downloaded into routers, could be used in a way that trusted elements control the SDN policy, and applications are designed to negotiate permission from these trusted elements before taking action such as sending data.

Another potential role for architecture is to add to the expressive power of the packet header some way to convey identity information, so that hosts and applications can discriminate between trusted and untrusted actors earlier in the initiation of communication. I discuss below both benefits and risks of this idea, and a designer should think carefully whether there is benefit in having any greater indication of identity visible in the packet, or whether this information should be conveyed at a higher level (end-to-end, perhaps encrypted) so that issues of identity become private matters between the communicating end-points.

Architecture and identity

I argued above that it would be a very bad idea for a future Internet architecture to include as part of its specification a fixed method to manage identity. This approach would (in abstract terms) be embedding too much semantics into the network. But perhaps as part of the expressive power of the header there should be a field into which any sort of identity information could be put by the sender as specified by the receiver. Different applications, in different contexts, could demand one or another sort of information be put into this field, so that the credential could be checked on receipt of the first packet, perhaps by a element in the network that had credential-checking as it PHB. The NewArch proposal included a *rendezvous field* in a session initiation packet, whose meaning was private to the communicating nodes.

As with any proposal to add some form of expressive power to the architecture, this must be examined from all perspectives–protecting a receiver from a sender and protecting communication from attack by the network. For example, a conservative government might demand that some explicit identifying information be added to the packet as a condition of making a connection out of the country. Today, there is no practical way to demand that, exactly because the packet header is not expressive enough. As we make the header more expressive, we have to consider how we have shifted the balance of power among the various actors.

DDoS attacks

DDoS attacks are a problem that has to be solved (at least to some degree) at the network layer. A network must have a way to manage and protect its resources–this is not an problem that can be "kicked up the layers" to the application. But again, the question is what sort of architectural support would be useful in mitigating these attacks.

There are several ways to think about dealing with DDoS attacks. One is to increase the barriers to the construction of botnets to the point where they become impractical. Perhaps, with careful attention to all the issues discussed so far in this chapter, that might be possible, but I set that approach aside for the moment. A second approach is to make it easier to disrupt the control of a botnet once it has been created. Again, a different architecture might make that goal easier, but since there are many conceptional ways to communicate with an infiltrated machine, this approach would require a rethinking of the basic communication paradigms of the architecture. Proposals such as NDN do not allow the sending of unsolicited data packets, so all they can send is interest packets. This limitation certainly changes the landscape of attack.

Assuming that in a new architecture it is still possible for an attacker to assemble and control a substantial set of infiltrated machines, which can then send traffic with the goal of overloading a target resource, the mitigation of this attack seems to have two components: first to determine which machines are sending the traffic and second to block the malicious flows. In the context of the current Internet, much of the work on DDoS has focused on the first problem: determining which machines are originating the attack. This is an issue in the current Internet because it is possible for a sender to put a source address in a packet other than the one associated with that sender. This might seem to be a malicious action, but in fact there are often legitimate reasons to do this: mobile IP (RFC 5944) requires that the source address in a packet be that of the "home agent" (a persistent address) rather then the current address assigned to the mobile device. The IETF tried to address this issue by proposing a "Best Current Practice" (BCP 38, RFC 2827) that recommends that source ISP check and validate the source address of packets they originate. However, there is no requirement that ISP conform to BCP 38 (other than some mild and probably ineffective peer pressure and shaming) and complying with BCP 38 imposes additional costs and complexity on ISPs. BCP 38 was promulgated in 2000, and has achieved some but by no means complete compliance.[7]

The situation with the current IP architecture raises the question of whether, in some alternative design, it could be made impossible for an attacker to forge a source address. A designer could take the naive approach of making a test similar to the one proposed in BCP 38 "mandatory", but how would such a mandate be enforced?

[7] *See https://spoofer.caida.org/, which reports on their attempts to measure compliance. Their report as of April 2016 is that about 57% of ASes, covering about 80% of routed IP addresses, detect false source addresses.*

As I noted in section 4.10, features described as part of an architecture that are not actually necessary for the operation of the network have a tendency to atrophy over time. To assure that source addresses are validated in packets, an architecture would ideally make that validation an integral part of forwarding the packet, so that the step could not be skipped in pursuit of performance.

Some architectural proposals explicitly allow the source address (the address to which a return packet should go) to be different from the address (location) of the sender. DOA, which concerns itself with delegation of services to other points in the network, makes clear that the sender of a packet can indicate in the source information the sequence of services to which a returning packet should transit, which is different from the processing of the initial packet. This design would seem to open up many opportunities for DDoS attacks. The DOA paper (see Chapter 5) discusses the use of an intermediary to protect a server from a DoS attack, but does not seem to address much attention to the use of malicious source (return) addresses. At the other extreme, schemes like Nebula, which require a sender to obtain a Proof of Consent from the control plane before sending a packet, would seem to preclude the forgery of source addresses.

Traceback A number of papers have proposed to augment the current Internet architecture with a *traceback* scheme, to allow a victim to identify the sender of a malicious traffic (to within some range of accuracy) even if the source address has been forged. These schemes, in general, exploit some mix of two mechanisms–*packet logging*, where the routers are augmented to keep track of packets passing through them, and *packet marking* where routers add to packets they forward some indication of that router's identity. It would seem that, almost independent of architecture, the first sorts of schemes will impose significant processing costs on every router, and the second must deal with how this information is written into the packet in a practical way. An example of packet marking, the Source Path Isolation Engine (SPIE) is described in [Snoeren et al., 2002], where routers compute and record a digest of every packet they forward, and record this digest in a Bloom filter. In principle, a victim, by computing the digest of a single attack packet, and sending a query into the net that follows paths where the digest has been recorded in successive routers, can determine the source of that packet. While the exact details of how the digest is computed clearly depend on the specifics of the IP header, this scheme would seem to be generalizable to different architectures.

Most proposals for *packet marking* make the assumption that it is impractical to record the complete sequence of routers forwarding a packet into that packet. The IP Record Route option did provide this capability, up to a fixed maximum number of hops.[8] A simple packet marking scheme requires each forwarding router to record its identity into a single field in the packet with some probability. A victim, receiving enough packets, will get the identity of each packet along the path, and (armed with some topology information) can reconstruct the path back to the sender. Perhaps a better scheme is for the packet to provide enough space to record the address of *two* packets: if the field has not been filled in, a router records its address in the first field, which then triggers the next router to put its address into the second field. The marked packet thus records a link or segment of the path, between two routers. Again, with enough packets, a victim can reconstruct a path to the attacker by concatenating these records. See [Savage et al., 2000] for a description of this *edge marking* scheme. A scheme by [Song and Perrig, 2001] describe different encoding schemes for the link, including more security for the marking. A hybrid scheme is described in [j. Wang and l. Xiao, 2009], in which the packet records the exit router from the source AS and the entry router into the AS of the victim, and those routers log information about the packet.

A key feature of many of these packet marking schemes is that they are designed to work in the current Internet, and thus spend much of their effort in designing a way to fit the marking into the existing IP header. The only real option available is to repurpose the usually unused fragment offset fields. In fact, the papers are so focused on the the necessity of confirming to the extreme constrains of the IP header that the papers do not give much

[8] *This option itself is not useful to tracking attack packets. The function is not widely implemented today, and further depends on the sender inserting the option into the packet, which an attacker is not likely to do. A scheme that can be used to track attackers must be mandatory, and not subject to defeat by the sender.*

insight how one might best do packet marking in a different architecture where the header could be designed for this purpose. Looking at the various alternative architectures described in Chapter 5, very few of them include a serious and complete analysis of dealing with DDoS attacks. DOA discusses the use of a service to block a DoS attack, but does not discuss in any detail how this service might work.

Blocking the attack The previous discussion of traceback (or ensuring that the source address in the packet cannot be forged) begs a perhaps more significant question: if the victim knows the actual address of the sender, how can the victim exploit this information. The key to DDoS mitigation is blocking the traffic, not just figuring out which machines are sending it. Most of the traceback papers I cite above do not address the question of how blocking can be done in a secure fashion; they leave this aspect of the problem as future work. This fact may be one of the reasons why none of these schemes has caught on in practice. A number of schemes have been proposed to block unwelcome traffic, including the indirection schemes I mentioned in Chapter 5, such as i3. A number of other schemes have been proposed. SOS [Keromytis et al., 2002] protects an end-point from DDoS attacks by putting in place a set of filters in the region of the network that hosts that end-point, such that all traffic must flow through those filters. (In the language of Chapter 4, SOS depends on *topological* delivery to the filters in order to protect the receiving end-point.) They try to prevent a further class of attack by keeping the addresses of the filters secret. Mayday [Andersen, 2003] elaborates on the design approach of SOS. There are a number of papers that propose and elaborate the idea of putting some sort of *capabilty* in packets sent from valid receivers, so that filters can distinguish valid from unauthenticated or malicious traffic. These include [Anderson et al., 2004], TVA [Yang et al., 2005], Pi [Yaar et al., 2003], SIFF [Yaar et al., 2004] and Portcullis [Parno et al., 2007].[9] . These proposals do not describe themselves as a "new Internet architecture", but they all require new fields in the packet header (new *expressive power*), new function in routers, and new protocols and mechanisms for connection setup. They should thus qualify as new proposals, and should be evaluated using all the criteria I have laid out in this book. Presumably, various sorts of packet filtering mechanisms could be deployed in routers along the path from the source, if such a router could be identified and it was willing to provide this service. But this general concept raises in turn many questions. One has to do with incentive–why should a router perhaps distant from the victim agree to provide this service? Another critical aspect of blocking is to ensure that any mechanism put in place cannot itself be used as an attack vector. If a malicious machine can forge a request to block content from a source to a destination, it can shut down valid communication–yet another form of an availability attack.

A very different approach to limiting DoS attacks is in the Framework for Internet Innovation (FII). The FII proposal is overall an exercise in just how minimal an architecture can be, and in fact devotes most of its complexity to blocking DoS attacks, which the authors argue is the only aspect of security which must be dealt with at the level of network architecture. Their scheme requires that each host be associated with a trustworthy component that can verify the source address and block traffic from that source to a given destination on request from that destination (using what they call a *Shut Up Message* or SUM). They define two fields in their header that must have global meaning: a valid address of this trusted agent and an identifier that this agent can map to the actual sender. (Note that with this design the actual location or identity of the sender need not be revealed to observers in the network. Only this trusted agent has to be able to map this identifier to the actual source.) The FII paper discusses in considerable detail the design of the SUM mechanism, with the goal of making sure that this mechanism itself cannot be abused. The resulting scheme is very complex and appears to require substantial cryptographic processing at the trusted agent.

One aspect of blocking an attack has to do with the design of the source addresses. Setting aside for a moment the issue of forged source addresses and forged blocking requests, what information in a source address would be useful (or in fact necessary) to implement a blocking scheme? Many of the architectures I have described use some form of separation between identity and location. The identity is considered to be robust (perhaps a public key hash that can be validated with some sort of challenge-response protocol, the execution of which on each

[9] *For a more complete discussion of these schemes, see the Appendix of the book.*

packet may prove a resource-intensive step that can be exploited for DoS), but the location information may be transient and with no global meaning. It may just be an identifier for the source AS, under the assumption that within the AS flat routing based on the identifier is practical. This scheme might allow for effective blocking, assuming the sender is not forging this information. On the other hand, a scheme like NewArch, in which the only meaningful information in the packet is the locator, may not provide a useful framework for blocking unwelcome traffic. A number of proposals (including the design of IPv6) have noted that for privacy reasons, a sender should be able to use a variety of locators, which partially mask the ability of an observer to map from locator to actual machine. Obviously, that AS within which the locator is meaningful must be able to resolve the binding from locator to specific machine, but the privacy goal is to prevent this from being done in general. In such a scheme, the only region of the network in which effective blocking can be done is in that AS actually hosting the source. Closer to the victim, there is, by intention, no robust mapping from source address to specific machine.

Assumptions of trust Any scheme to mitigate a DDoS attack will end up depending on the trustworthy operation of some set of components in the network, whether a trustworthy Network Interface Card on a corrupted sender, a trustworthy router along the path, and so on. Mitigating DDoS is another example of a situation where the design of a scheme does not just depend on a good technical design but as well the design of a system that is "socially correct"–a system that makes the correct assumptions about trust and incentive. In general, most of the architectures I have discussed here do not devote full attention to the issue of DDoS, which is perhaps a missed opportunity, since the range of options for DDoS attacks may depend very much on the specifics of the architecture, and a full analysis might have revealed what architectural options would be best suited to mitigate DDoS attacks.

7.10 Conclusions

Barriers to better security

Security problems in the current Internet are in part a result of technical design decisions. But the flawed technology is not the result of error or lack of attention. Applications that are insecure by design are perhaps the most difficult problem to address, and the decisions to design applications in this way are deliberate, driven by larger economic considerations of appeal and usability. Barriers to the security of distributed systems such as the interdomain routing system, email or the web are problems of coordinating and incentivizing collective action, dealing with negative externalities and costs imposed on first-movers, understanding how to cope with a lack of uniform trust across the system, and the like. To overcome these barriers will require good system design, but that design is not exclusively technical. There must be complementary aspects of technology, operational requirements, governance, and the like.

Comparing the "computer science" and "user" or "political science" definitions of security sheds some light on these issues. The computer science definition of security–that a system will only do what it is specified to do, even under attack–defends against unexpected outcomes or behavior but is not framed in terms of preventing specific harms. It is an appealing definition to a system engineer, because it seems to frame security in a way that bounds the problem to the system in question. Framing security in terms of preventing harms (e.g., preventing credit card fraud) brings many more elements into scope. For example, preventing or mitigating some forms of credit card fraud may best done by modification of the credit card clearing system. This definition of security frustrates the designer of a component, because the scope of the solution is no longer within the scope of the designer to fix. Of course, if the system in question is a multi-component system with multiple actors responsible for parts, even the "computer science" definition of security may be hard to contemplate. But only by contemplating the potential harm can one begin to determine the level of effort to put into defending the system elements. As I noted above, the design of the Internet presumes that the lower layers are not perfect, so the degree of effort put into hardening them must be a matter of judgment, not the pursuit of perfection.

The centrality of trust

What has repeatedly emerged in this analysis is that whatever technical mechanisms are used to improve security are embedded in a larger context of trust management. Trust management is the ugly duckling of security, compared to cryptography. Cryptography is lovely and complex mathematics, it has provable bounds and work factors, it is an amazing tool. Tools to manage trust are messy, socially embedded, not amenable to proofs, and the like. Sadly, crypto is almost always wrapped inside one of the larger, messy contexts. At a minimum the problem of "key management" is always present, and the problem of securing the routing protocols of the Internet (or the DNS, for another example), or improving availability, or allowing applications to adapt their behavior based on the apparent threat, all depend on trust as a central issue.

The actor that gets to pick which element to trust has the power to shape the security landscape. Mechanisms that create points of control may initially easier to reason about, but given the potential tussle that arises around any centralized point of control (e.g., certificates), a better real solution may be to prefer "socially stable" solutions such as highly decentralized control and decision-making.

7.11 Acknowledgement

This chapter has greatly benefitted from discussion with Josephine Wolff, Shirley Hung, John Wroclawski and Nazli Choucri at MIT.

Chapter 8

Availability

Since "all" most networks do today is deliver data, it would seen that their ability to carry out this function, even under adverse conditions, would be a primary consideration. The term used to describe this general character is *availability*. The term *resiliance* is sometimes used in this context, and captures the idea that the challenge of availability is to function when things are going wrong, not when everything is working as it should. An available network is a network that is resilient in the face of failures. The opposite of availability is *outage*, a term used to describe a failure of availability.

8.1 Characterizing availability

A definition of availability only makes sense within the scope of the particular functional specification of a network. A delay tolerant network that promises to deliver email within a day under normal operation (the utility of such a network is a separate question) would presumably define a failure of availability differently than a real-time delivery service that promised delivery within a factor of 2 of the latency of light.

There seem to be at least two dimensions of availability (or its lack): time and scope. For how long was the service not available, and over what portion of the network did the failure apply? The dimension of time is essential here. In the current Internet, we do not consider the loss of a packet as a failure of availability. TCP retransmits the packet, and application designers are expected to anticipate and deal with this sort of fluctuation of delivery time as "normal", not exceptional. When links or routers fail, this can cause a loss of connectivity that lasts long enough to disrupt some applications (e.g., a voice call [Kushman et al., 2007]), so it might be reasonable to describe these events as a transient lost of availability. However, these would not rise to the level where a regulator tracking *outages* would expect the event to be reported. An outage that lasts for hours or days is of a different character.

The dimension of scope is similarly essential. For a user of the Internet in the U.S., the loss of connectivity to a small country in Africa might not even be noticed. For the citizens of that country, the disruption would be severe. The measure of availability (or the assessment of the importance of a failure) is a matter of the observer's point of view.

These example also suggest that availability must be seen as a concept that requires a layered analysis, just as with security. Higher layers can (in many cases) compensate for failures at a lower layer. For example, data can be cached in many locations to improve the availability of the data even in the presence of failures of the communications infrastructure.

8.2 A theory of availability

I start with the assumption that a system in which its components are working according to specification is available. While networks may have very different service commitments, it makes little sense to talk about a network that fails its definition of availability under normal operation. This framing ties a potential loss of availability to a failure of part of the system. When something fails, two sorts of correction can occur. First, the

layer in which the failure occurred can undertake to correct the failure. Second, a higher layer can undertake corrective or compensatory action. Ideally, these two actions will not be in conflict. This implies that one valuable component of a layer is some way to signal to the layer above the nature and duration of a failure, or perhaps a specification of the normal duration of a failure.[1] I return to this issue of inter-layer interfaces for management in Chapter 10.

In order for a layer to recover from a failure, either the failed component itself must recover, or there must be *redundant* elements that can be exploited to restore service. There is thus a division of responsibility in achieving high availability–the design of the system must allow for the full exploitation of redundancy, and the system as deployed must include enough redundancy to cope with anticipated failures. Redundancy must both be present and exploitable for a system to recover from failures and restore availability.

There is thus, at an abstract level, a series of steps that must be part of a scheme to cope with failures.

- It must be possible to detect the failure.
- it must be possible to localize the failed parts of the system.
- It must be possible to reconfigure the system to avoid depending on these parts.
- It must be possible to signal to some responsible party that a failure has occurred.

Each of these may seem obvious, but all can be tricky. The list above is in the passive voice, which is deceptive. It begs the question of *what actor* has the responsibility for each of those steps.

Detecting failures: With respect to detecting a failure, simple "fail-stop" failures are the easiest to detect. The hardest are failures where the element is partially operational so that it responds (for example, to management probes) but does not fully perform. A mail forwarding agent that has failed is easy for a sender to detect (and using the DNS, there is a scheme to avoid the failed component by moving to a backup server.) A mail forwarding agent that accepts the mail and then does not forward it is harder to detect. It is possible that some future design for an Internet might argue that its architecture allows the network layer to detect *all* the failures that arise at this level, but I would find this assertion to be very bold. It is probably possible to enumerate all the elements in the network (although even this task gets more difficult as more PHBs creep into the network, perhaps only with contingent invocation.) However, as network functions get more complex, to enumerate all the failure modes (or to create a robust taxonomy that covers all classes of errors) seems a rather daunting challenge, especially in the context of security-related failures. I argue that in general, only an "end-to-end" check can confirm if something is failing (e.g., the mail is not getting through), but the end-to-end check does not help with the second step–localizing the problem. So the resolution of the "passive voice" with respect to this step is that while a layer should do all it can to detect failures, the end-nodes must play an essential role of last resort.

This line of reasoning about detection of errors applies specifically to availability issues that arise in the context of attacks on communication (see Section 7.5). Given that faults that arise from malice may be crafty and Byzantine, both detection of faults and their localization may be difficult.

Consider a very simple example–a router that drops or adds packets to a packet flow. This sort of action does not break the forwarding layer, just the end-to-end communication. Should the packet forwarding layer keep count of packets, and exchange these counts to see what is being lost or gained? Or consider the more subtle attack of changing a bit in an encrypted packet. This attack disrupts the higher-level flow. Should the network re-compute the encryption function at each node to detect that (and where) the packet is corrupted? This may turn out to be a useful mechanism, but the complexity and performance cost seems daunting.

[1] *For example, the ring topology of the SONET technology had a design target of 30 ms to recover from a single fiber cut. The specification to the higher layer was to wait for 30 ms. before undertaking any adaptive steps to deal with a perceived failure, to see if the SONET recovery was successful. If connectivity did not return in 30 ms., the problem was more severe and higher-layer action was justified.[[[Fact check]]]*

Localizing the fault: For simple faults, where the layer itself can detect the failure, localization is often a direct consequence of discovery. Routers send recurring messages to each other, which serve to construct routes and also to confirm that the remote router is still functioning. Dynamic routing protocols are designed to recover … [[[discuss distributed recovery vs. centralized, such as 4D or SDN]]] The more complicated situation arises when the end-nodes have detected the problem. In this case, there does not seem to be a single, general approach to localizing the fault. One approach is some sort of monitors or "validation" units at interconnection points within the network that could make records of what passes–by comparing the records the end-nodes could somewhat localize where there had been manipulation of the packets. Schemes like ChoiceNet (Section 5.4) have proposed such an idea. But the question remains as to what sort of architectural support would facilitate this sort of scheme–perhaps some sort of control flags in the packet that would trigger various sorts of logging and debugging. Another approach is route diversity, trying selective reconfiguration of the system, avoiding different parts of the system in turn, to see whether the problem persists.

As I discussed in Section 4.9, it is not always desirable to assure availability. When an end-node is being attacked, and prevents the attack from reaching it (perhaps using some PHB in the network) what it has implemented is a deliberate loss of availability (as seen by the attacker). In this case, where the interests of the sender and receiver are not aligned, not only should the sender be deprived of tools to 'remedy" this impairment, the sender should not be facilitated in localizing the source of the impairment. In the current Internet, there seems no obvious way to resolve this dilemma if the resolution depends on the network knowing whether the sender and receiver are aligned in their intention to communicate, and further given that it might be the network that is attacking the communication. Perhaps, as I speculate below, there is an architecture feature that might help resolve this situation.

Reconfiguring to avoid failed elements: With respect to reconfiguration, the idea is fully understood in specific contexts. The example of email above uses the DNS to allow a sender to try a backup forwarding agent. Dynamic routing uses probing to try to detect failed elements and route around them. And so on.

With respect to enhancing the availability of packet forwarding (the essential element of network layer availability) the current Internet faces a serious conundrum. The design of today's Internet is based on the defensible assumption that while some failures ("simple" failures) can be detected by the network, in general failures, especially those due to malicious attack, can only be detected at the end points, so the task of detecting them is delegated to the end. But assuming that the end-point detects a failure, what can it do? In today's Internet, the end-points have very little or no control over network functions like routing. If communication between end-nodes is being attacked, the end-nodes have no general way to localize the problem, and no way to "route around" it.

There are (or were) some means in the Internet to give the user control over which entities to trust. The mechanism of source routing, which would have supported this sort of control, has vanished from the Internet. There are several reasons why source routing has been deprecated. One is economics: if the user has choice over routes, should not that capability be links to a way of charging the user for the resources he chooses to use? Another reason is security. If the network design gave that sort of control to the end-nodes, those mechanisms themselves might become attack vectors, so they would have to be designed with great care.[2]

Today, what happens is that we accept that a range of attacks are going to result in loss of availability. If a network must function at high levels of availability, we use non-technical means to make sure that only trustworthy components and actors are in the system. So to achieve the full complement of CIA, both technical means and operational and management means must be combined as part of the approach.

[2] *For one discussion of potential security risks associated with source routing, see* `https://www.juniper.net/documentation/en_US/junos12.1/topics/concept/reconnaissance-deterrence-attack-evasion-ip-source-route-u` `html`.

At the higher layers of the system, the current Internet indeed gives the end-node a degree of choice over which versions of a network service are used. A sophisticated user may know to manually select a different DNS server if the default server is not acceptable. As I mentioned, the email system uses the DNS to provide the sender with alternative forwarding agents if one is not responding. A *very* sophisticated user may know that it is possible to edit the list of Certificate Authorities that he chooses to trust. I would claim that while these features do exist, they are not designed as a part of an overall conception of how to improve availability.

Reporting the error: With respect to reporting the error, I defer that problem to Chapter 10 on management.

8.3 Availability and security

The discussion to this point suggests that the objective of availability is enhanced by allowing both the system and the end-users to have enough control to select elements for use that are functioning correctly. In the context of an attack on communications, this objective needs to be refined. What the end-user should be doing is selecting elements that *do not attack him or otherwise disrupt him.* A crude approach might be to select alternatives at random until one is found that serves. A more constructive approach is to allow the end-nodes to structure their interactions so that they only depend on elements they consider trustworthy. If there is a malicious ISP, don't route through it. If there is a email sender that seems to send only spam, block receipt from it (this sort of treatment is what anti-abuse organizations such as Spamhous try to coordinate). Essentially, if we want all of the CIA triad for communication security, we must organize the system so that even if untrustworthy actors are in the system, we do not depend on them. We tolerate them if we must, but we do not make any interactions among mutually trusting actors depend on untrustworthy elements unless they are so constrained that we can rely on them.

This point of view has been slow to come into focus for some designers of security mechanisms, because it is a shift in mind-set. This logic is completely obvious to designers when it comes to failures: if a router has failed, the protocols must be able to detect the failure, and there must be sufficient redundant routes that a dynamic routing protocol can "route around" the failure. But for some in the security community, with its history of a focus on confidentiality and integrity, the idea that availability must depend on assessment of trust rather than a technical mechanism is perhaps disconcerting, and perhaps disappointing.

Routing and availability

The previous discussion dealt with one aspect of security (as I classified security problems in Chapter 7): attacks on communication by a hostile third party. A more basic aspect of security from the perspective of availability is an attack on the network itself that disrupts availability, most obviously by disrupting the routing protocols. Clearly, the stability and proper operation of routing is essential for network availability.

In addition to making the routing mechanisms more resistant to attack, having multiple routing schemes running in parallel might be a way to improve the resilience of the network when it is under attack. XIA implements this idea, with their different sorts of end-point identifiers (content, service, network, host, etc.), and different routing schemes for these different classes of entity, and the ability to fall back to a different scheme if the preferred one is not available.

As I discuss in Chapter 10, the Internet is moving to more centralized route computation schemes such as Software Defined Networking, or SDN. Perhaps a centralized scheme like SDN could be complemented with a simpler and perhaps less efficient backup scheme that can be used if the centralized scheme seems to be impaired. The network could fall back to this simpler scheme as a resilience mode. However, this idea, while it might improve availability when the network is under attack, could at the same time worsen another aspect of security, which is protecting a node from being attacked by other nodes. Part of the power of SDN is supporting finer-grained routing decisions based on policy in each router. Having another scheme that bypasses these controls could thwart those policy goals. This tension illustrates that different security sub-goals may end up in conflict, and in

particular that it takes crafty design to balance the dual goals of high availability even when parts of the system are failing and selective availability to block hostile traffic, especially in a "dumb network" that does not know what the users are sending.

Assuming that these tensions can be resolved, the emergence of new routing schemes, perhaps operating at the same time, raises the question as to whether a new field should be considered in the packet header, (somewhat as XIA has done) to indicate which of several routing schemes should be used for this packet. Alternatively, a future architecture could use different address ranges to trigger different routing (as the current Internet does for multicast), thus giving the receiver control over which schemes can be used to reach it (by controlling which sorts of addresses it gives out for itself) and allowing the sender to pick among them (by picking which destination address to use). By tying the choice of the routing protocol to the address range, third parties in the network cannot override the end-node choices by rewriting a field in the router. The NIRA scheme [Yang, 2003] uses addresses to control routing in this way. This might allow senders and receivers to select between more availability and more protection based on their specific needs.

8.4 Architecture

A key challenge for a future architecture is to resolve the basic conundrum I identified above: if only the end-nodes can detect failures of availability due to attacks, and the end-node cannot be trusted to reconfigure the network lest this be another attack vector, there would seem to be no way to resolve such problems. Working around this conundrum is a challenging design problem that involves creation of control structures that build on trustworthy components (which would have to be specified and implemented) to provide a foundation for these sorts of functions. A component trusted by both the user and the network might be able to intermediate between the two in order to provide a measure of choice and control to the end-node that has detected a failure or attack.

Another potential role of architecture is to facilitate fault localization. As I noted above, this capability is not always in the interest of the receiver, if the receiver is being attacked. Perhaps it is worth exploring a shift in the basic architecture of the Internet. The Internet of today is "deliver by default": a sender can send to any receiver at will. Perhaps there is merit in an approach that is to some extent "deny by default", so that the receiver has to take some action to indicate its willingness to receive traffic, or to receive traffic from which set of senders. Several of the architectures I discuss in this book are to some extent "deny by default".

The discussion of invoking PHBs in Section 4.4 provides one possible way to improve this situation. In that section, I proposed a rule that (with the exception of routing itself) any invocation of a PHB that facilitates communication between willing parties should be intentional–the packets should be delivered to the location of the PHB by being sent there. A more abstract way of saying this is that between senders and receivers that have aligned interests, the end-points should be explicit about what PHBs are being invoked. (This action may be done by an application on behalf of the end-node, in which case it will be the application that has to attempt to localize the point of failure when something goes wrong.)

What this rule would imply is that third parties should not claim to be "helping out" an application by inserting PHBs into the path that neither the sender nor the receiver know about [[[[?]]]]. Once this rule is in place, encryption can be used to limit the failure modes that manifest from unknown PHBs that show up in the path. It may be cleaner (and more secure in practice) to put in place an encryption scheme that lets selected PHBs decrypt a transfer (or parts of it) rather than have an ambiguous relationship between sender, receiver and arbitrary PHBs in the path. [[[Is this clear? Probably not.]]]

The different proposals in Chapter 5 provide a range of mechanisms to deal with these various aspects of availability. To some degree, all those schemes depend on the end-node as the ultimate detector of failures and loss of availability. With respect to localization, Nebula and XIA (both with its basic addressing scheme and in particular the forwarding scheme called SCION) provide a way for the user to pick different routes through the network, potentially making choices to avoid regions that are proving untrustworthy. ChoiceNet provides monitoring elements at region boundaries that are intended to check if the user is getting the promised service.

It is unclear what range of problems they will be able to detect. ICNs raise a slightly different version of the availability challenge. ICNs attempt to exploit all the redundancy in the network, often through some sort of anycast search for a nearby copy of the content, so that a failure may be side-stepped as part of the basic content request function. The malicious attack that can disrupt availability in ICNs is a malicious provider that offers up a malformed version of the content, which can be detected as such but prevents the anycast mechanism from finding another copy that is valid. DONA provides an enhancement to the FIND operation that allows the user to ask for the n-th closest copy rather than the closest copy. NDN allows the receiver to include the public key of the sender in the content request packet, so that nodes along the path can check for themselves the validity of the content and reject malformed copies.

[[[elaborate]]]

8.5 Conclusion

[[[TBD]]]

Chapter 9

Economics

[Note to readers of this version of the book. I view this chapter as preliminary. I think there is more to be said, but I have to figure out what it is. Comments and thoughts welcome.]

9.1 Introduction

The viability and success of a network architecture cannot be divorced from the economics of its deployment and operation. At the same time, there has been very little attention in the literature to understanding the relationship between architectural alternatives and economic viability.

In order to understand the issues, it may be helpful to look at the current Internet as a case study, and then explore the issues that emerge in a more abstract and perhaps fundamental way.

The current Internet is composed of regions (we typically call the larger regions Autonomous Systems), which are deployed and operated by different actors. There are about 45,000 ASes active in the Internet today. Of these, about 5,000 can be classified as service providers; they offer packet carriage service to other parties. The rest are customers of these providers. Most of these service providers (ISPs) are private-sector, profit-seeking actors. The interconnected mesh of these ASes, taken collectively, is what we call the Internet. It is the platform on which higher level services (applications) run, and has become the platform on which society is increasingly dependent. This Internet platform has taken on the status of societal infrastructure, and probably the status of essential or critical infrastructure. The Internet may not be as important as water, sewers or roads, but it is now infrastructure on which society clearly depends.

In comparison to these other infrastructures (roads or water systems), what is distinctive about the Internet is that it has been largely built by these unregulated, profit-seeking actors. Roads and water systems are normally built by governments, and while the telephone system (our previous communications infrastructure) was built by a private sector actor (Bell Telephone/AT&T) it was a highly regulated, government sanctioned monopoly, not at all like the ISPs of today.

Today, we take this situation for granted. We assume we can count on the Internet to be there, and we assume these private-sector actors will continue to provide it. But this assumption should be carefully inspected for flaws. The Internet exists because ISPs chose to enter the market with this product. Nobody forced them to do so. We could ask, looking backwards, why they did so. We should ask, looking forward, whether this situation will continue to be stable. If the Internet is societal infrastructure, can we assume that the private sector will continue to invest in it at a suitable level to meet the needs of society, and can we assume that the private sector will continue to build and operate the Internet that society wants? Perhaps investment will stagnate. Perhaps the ISPs will be motivated to mutate the Internet they offer into a different sort of platform, perhaps more closed or dedicated to specific purposes such as delivery of commercial video.

In chapter 11 I explore in some detail what it means for an Internet to meet the needs of society, but it is necessary to begin that discussion here to identify those issues that strongly relate to economics. Should the future of the Internet be whatever the private sector chooses to deliver, or does society want to have a say in the

future? If so, by what means can society have a say in shaping what the private sector does? How can "society", whatever that term means, even discuss and decide what its Internet should be? Is this a call for the government to step in and define the future of the Internet?

In fact, governments are starting to do so, for better or worse. The most obvious evidence in the U.S. is the sequence of efforts by the FCC to impose network neutrality rules on the Internet.[1] One response (or threat?) by the ISPs covered by these regulations is that the rule will stifle investment. The question of whether ISPs will continue to invest, and whether they will build the Internet society wants, is not abstract. it is being acted out today as we watch. In fact, there are observers who assert that we cannot continue to count on the private sector to be the driver of the future Internet, and that the public sector will have to invest as they do in roads or water systems. [[[find a few cites–perhaps Australia?]]] We see governments today using public sector funds to build Internet access in rural and other low-profit areas the private sector has ignored. But this approach has its own perils–why should we expect governments to have the skills and will to build something as dynamic and evolving as the Internet? So looking to the future, an optimist may see the Internet as so compelling that it will obviously continue to be there, and a pessimist may see several paths to the future, all fraught with perils. Society must pass between Cylla and Charybdis, those eponymous perils between which society always sails, avoiding on the one hand stifling investment and on the other hand getting investment in an Internet that is not suited for its needs. It is in this space that we must consider the economics of the Internet.

A look back

How is it that the Internet (and the investment that brought it to market) actually happened? In the early days, the Internet was built on top of circuits constructed by the telephone company. The option of constructing new capacity dedicated to the Internet was not practical, so the early Internet worked with what could be purchased at the time–the first long distance circuits that made up the ARPAnet were 50 kb/s telephone circuits. ARPA did invest in experiments in alternative technologies such as packet radio, but one of the reasons the Internet protocols were designed to "work over anything" is that using what was to hand was the only way to move forward.

In the mid-1980s', NSF took over the operation of the national backbone, and built (using public sector funds) the NSFnet, upgrading the capacity as it was practical. This public-sector investment demonstrated the viability of the Internet as a information technology, and justified the entry into the ecosystem of profit-seeking, private sector actors. It is not clear if the private sector would have chosen to enter the market if the NSF had not taken this high-risk (from a private-sector perspective) step of "building it to see if they came". But even in the mid-1990s', when NSFnet was being decommissioned in favor of a private-sector offering, the appeal of the Internet as a product was not clear to many of the existing private sector actors. In a conversation about broadband to the home, a telephone company executive said the following to me:

> *If we don't come to your party, you don't have a party. And we don't like your party very much. The only way you will get broadband to the home is if the FCC forces us to provide it.*

Of course, he thought the copper pairs into the house were the only option for broadband access. One could argue that this executive could have been more forward-looking, but given this attitude, why did the telephone companies start to invest in residential broadband? To a considerable extent, it was the emergence of the cable industry as a competitor in the residential market, using a separate physical infrastructure to deliver broadband access. Competition can be a powerful driver.

In the U.S., the government seems to vacillate between regulatory intervention and faith in competition as a tool to shape the future. In the FCC's National Broadand Plan, the FCC, after describing a number of aspirations for the future of the Internet, wrote:

[1] *Discuss the three open orders, citations? etc. How much is needed?*

> *Instead of choosing a specific path for broadband in America, this plan describes actions government should take to encourage more private innovation and investment. The policies and actions recommended in this plan fall into three categories: fostering innovation and competition in networks, devices and applications; redirecting assets that government controls or influences in order to spur investment and inclusion; and optimizing the use of broadband to help achieve national priorities.* [Federal Communications Commission, 2010a, pg. 5]

One can realistically ask if this approach is an act of massive wishful thinking. Competition may be a powerful driver, but in what direction? What evidence is there that competition will drive the private investors in Internet infrastructure to build the Internet that the FCC (or society more broadly) desires? And as well, one can realistically ask if competition based on separate physical facilities (as between the coaxial cable infrastructure of the cable industry and the twisted pair infrastructure of the telephone industry–now both mutating into fiber) can be sustained. Dividing up the market between two or more more providers means (among other things) that the potential market for each provider (the "take-rate") is divided up among the competitors. But each competitor must still build an access network that passes every house. If a market has the character that the more customers a given firm has, the lower its costs (positive economies of scale at all scales), economists describe this situation as a *natural monopoly*. The phone system of the Bell era was taken to be a natural monopoly, and was accepted and regulated for the public good. Perhaps the Internet of the future will turn out to be a natural monopoly, especially the access architecture.

So the particular set of issues that we must consider as we contemplate the economic viability of an Internet architecture are as follows:

- What are the incentives of the private sector to invest in infrastructure? Can it be sustained?
- To the extent that society wants to have a say in the future of the Internet, what are the means to shape the behavior of the private sector to get the outcomes that society desires?
- Can we assume that competition in the access market will continue, or will the future be one more defined by regulation rather than competition?

And, for each of these issues, there is the related question about architecture:

- Can architectural decisions shape (or reshape) the Internet ecosystem so as to better provide incentives for investment? Should we make decisions about architecture based on the assumption that the private sector will continue to build an Internet? Would different decisions be preferable if a future Internet were a public-sector infrastructure?
- Can architectural decisions serve to nudge the outcome of private sector investment in directions that meet the needs of society?
- To the extent that one accepts competition as a desirable discipline in shaping the Internet of the future, can architectural decisions improve the potential of competition (including and specifically competition in providing infrastructure based on different physical facilities) as a long-term option?

Earlier, I used the term *tussle* to describe the contention among actors with different and perhaps mis-aligned interests who seek to shape the Internet. I have named as the *fundamental tussle* the tension between ISPs who assert that they should be able to use the infrastructure they paid for in any way they see fit, and regulators who wish to constrain how that infrastructure is used so as to pursue an Internet suited to the needs of society.

Scarcity

Economics is a discipline that studies the allocation of scarce resources. If there is no scarcity, there is no need to allocate (we can all have as much as we want) and issues of economics fade into the background. So is the Internet a scarce resource? One way to look at the Internet is that once the facilities to carry traffic are in place, the incremental cost of sending additional traffic is essentially zero, so we should think about usage as free. This statement may be true in the short run, but the Internet requires physical facilities in order to exist. These include long distance and metro fibers, residential access networks, wireless base-stations, and so on. These cost money. So to encourage investment in additional capacity, usage has to be seen as generating cost. Providers must recover enough costs that there is a positive return on investment. It is not clear how this fact relates to architecture, but it brings into focus the critical question of who pays for the Internet, and how the money flows among the actors in the ecosystem.

I will return to money flows in a later section. But there is a more fundamental question. Before we can ask which actors are paying for the Internet and how the money flows among them, we must ask why the industry structure of the Internet looks as it does. Why do we have the actors that we do in the Internet? Why do ISPs exist in the form that they do?

9.2 What shapes industry structure?

A useful place to start is fundamentals: can economic theory tell us anything about the relationship of system design, the resulting industry structure and the incentives of the the various actors that make up the system to invest and play their part in making a healthy economic ecosystem. In fact, there is a wonderful framework that helps to explain this space, based on the work of Ronald Coase.

The economist Ronald Coase received a Nobel Prize for his *theory of the firm*, which builds on the concept of *transaction cost*. When firms engage to buy and sell, there are costs aside from the actual cost of the product or service itself: the costs of searching for the providers, the costs of bargaining, the costs that arise due to lack of accurate information about the other firms, and so on. Collectively, these are transaction costs. When transaction costs are low, efficient inter-firm competition can occur, but if transaction costs are high, a firm may incur a lower total cost by realizing the service or function internally. Competition in principle drives down costs, but not in practice if transaction costs are high. One conception of this situation is that inter-firm competition and intra-firm planning and coordination themselves compete to deliver the lowest cost of products and services. Large firms exist when and if the cost savings from internalizing the function exceed the cost savings from competition.

The link between Coase's theory and network architecture is the role of well-defined interfaces between modules. If an interface between modules is well-defined and easy to understand, then exploiting this interface as a basis for competitive interaction among firms may have a sufficiently low transaction cost to be viable. If, on the other hand, there is no clear interface at a particular point, it is hard to "open up" that point to inter-firm action, and that point will normally remain internal to the firm. So the modularity of a system like the Internet, which a technologist might think of in functional terms, is also very likely to end up defining the industry structure of the system.

Defining the relationship between the parts

If architecture defines the shape of the industry, what defines the relationship among the different actors? It is the interfaces defined by the architecture. In the current Internet, we see several key interfaces.

The Internet protocol and the Internet Service Provider The Internet protocol (IP) actually defines a number of interfaces, each of which has helped to define the market structure of the Internet. Most obviously, IP defines the packet carriage service of the Internet–the service that "the network" provides to the higher layers. It is the service defined by the IP specification that becomes the service provided by an Internet Service Provider.

If IP had been specified differently, the business of an ISP would be different. For example, if IP had specified reliable delivery, ISPs would have the responsibility for reliability.

The service specified by the IP spec is minimal. RFC 791 says:

> *The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols.*

Perhaps a different specification of the service, with more attention to the features of the service, would have created different revenue-generating opportunities for the ISPs.

The interface to the communications technology A second interface created by the IP specification is between the service itself and the technology used to deliver it. The IP spec says nothing about the technology, other than it be able to forward sequences of bits. This decoupling, while perhaps implicit, means that the specification allows (and thus encourages) innovation in network transmission technology. Over the decades since IP was specified, there have been any number of network technologies invented, including Local Area Neworks (LANs), WiFI and cellular networks and so on. It is the limited requirement that IP places on these technologies that facilitates these innovations.

The interface between the ISPs There is a third interfaces that is relevant to the IP layer, but it is poorly developed in the original design of the Internet–the interface between ISPs. In the original Internet, the designers downplayed the importance of this interface (sometimes called the network-network interface, or NNI). That decision was perhaps short-sighted. The original designers were not thinking about industry structure–they were designing a network built out of routers. The address in the header of the packet (together with other fields in the header) defines what the router has to do when it gets a packet. Getting that function right was the initial focus of the designers. When the architects started to focus on the fact that different parts of the network were run by different entities, there was some confusion as to how the interconnection would be implemented. One view was that there would be two routers, one belonging to each of the providers, connected to each other at the point of interconnection. Technically, this seemed inefficient–why have two routers next to each other? The alternative view was that there might be one router, jointly operated by the two providers. This idea reduces the number of routers in the system (a serious consideration at the time) but would have required some sort of division of responsibility within this one element. Only after it became clear that this idea was totally unworkable for operational and management reasons did the question arise as to what information two routers belonging to different providers would need to exchange with each other. [2]

RFC 827, published in 1983, provides some insight into the level of understanding about the inter-AS interface at the time that IP was being deployed.

> *In the future, the internet is expected to evolve into a set of separate domains or "autonomous systems", each of which consists of a set of one or more relatively homogeneous gateways. The protocols, and in particular the routing algorithm which these gateways use among themselves, will be a private matter, and need never be implemented in gateways outside the particular domain or system.*
>
> *The Exterior Gateway Protocol enables this information to be passed between exterior neighbors. ... It also enables each system to have an independent routing algorithm whose operation cannot be disrupted by failures of other systems.*

[2] *Today we do see the other configuration, with a common switch between a number of ISPs. This configuration is called an Internet Exchange, and a neutral third party is usually created to operate that shared switch.*

The Expressive Power of the Interface

In Chapter 4 I distinguished different architectures by their *expressive power*. Some packet headers have richer expressive power than others, which can enable a richer set of services in the network (as well as new potential security issues). There is another dimension to the concept of expressive power, which relates to the possible set of protocols that are defined for the *control plane*. Control plane protocols define the messages that can be exchanged among network elements, and if these are of global scope (that is, if all the regions are expected to agree on their meeting) they rise to the level of architecture. The only relevant control plane protocols in the current Internet are the routing protocols, and the interdomain routing protocol (e.g., the one with global scope) is the Border Gateway Protocol (BGP).

BGP is perhaps one of the first examples in the Internet (or at least, a substantially worked out example) of a protocol that was designed to shape industry structure. The predecessor of BGP, the original EGP, assumed a hierarchical pattern of interconnection among the regions (the ASes), with NSFnet as the root of the hierarchy. If EGP had become the routing protocol for the commercial Internet, a single commercial provider would have ended up taking the place of NSFnet, in a position that seems close to an architecturally-created monopoly. BGP was specifically designed to allow for multiple, competing wide-area Internet Service Providers.

At the same time, BGP has limited expressive power, and these limitations have arguably limited the business relationships among the interconnecting ISPs. See [Feamster, 2006] for one discussion of the limited expressive power of BGP.

At the time of the transition to the commercial Internet, industry had a clear understanding of the importance of critical interfaces. As I discussed in Chapter 5, the Cross-Industry Working Team laid out their vision of an architecture for a National Information Infrastructure [Cross-Industry Working Team, 1994], and a central component of their conception was a set of critical interfaces, including the interface between ISPs: the network-network interface, or NII. They fully understood that getting this interface right was critical to the health of the emerging private-sector industry. The designers of Internet protocols may not have fully appreciated the importance of this interface.

Alternative architectures

There are some aspects of industry structure that seem to derive from more basic considerations than architectural variation. The idea that a global network is built out of parts (ASes, regions, and so on, by whatever name they are called), and that these parts have some geographic locality and are operated by distinct providers, seems common to all the architectures that I described.

However, to see that architecture defines industry structure, consider the implications of information-centric networks such as NDN. In NDN, the provider of the forwarding layer has a much richer set of responsibilities, and access to a much richer set of explicit information in the packet headers. These providers can see the names of what is being sought, not just a end-point address, which might open up new opportunities for traffic discrimination. These providers provision caches in their servers, which are critical to the efficient operation of the protocols. They thus have control over what is cached, and on what basis. One must look at a scheme such as NDN not just through the lens of a mesh of interconnected routers, but rather as a mesh of interconnected ASes with independent profit-seeking motivations, to try to understand the economic implications of the architecture.

Architectures such as Nebula and Choicenet make the negotiation over how and whether traffic will be carried an explicit part of the design. They include a control layer (or in Choicenet an *economy* layer) in which negotiation over service and payment can occur. They attempt to bring the economics of the architecture out and make it an explicit part of the design. They include new components, such as monitors at AS boundaries to verify what service is being provided, and stress the power of competition and user choice to drive the network to a desired future.

Incentives to invest

What motivates a private-sector actor to invest, specifically in infrastructure for the Internet? At a high-level, investment is motivated by anticipation of adequate return on investment (RoI), and the greater the risk (the uncertainty about the RoI) the higher the anticipated RoI must be to justify investment.

Looking specifically at the communications sector, there are a number of factors that can influence investment decisions:

Assured RoI: In the era of the highly-regulated Bell System, investment in capital assets was incentivized by Rate of Return regulation, which set a guaranteed return on capital upgrades. This return, which was factored into the regulated rates paid by users of the telephone system, probably resulted in over-investment, and brought the U.S. a highly engineered and reliable phone system which probably cost a bit more than it needed to, and supported a world-class industrial research lab, Bell Labs. Most economists with whom I have spoken say that a return to rate-of-return regulation would be a Bad Idea.

Competition: Competition, in particular when competitors have separate physical access technologies ("facilities") can be a powerful driver of investment. Today we seen the telephone companies and the cable companies, occasionally further goaded by new entrants gleefully pulling fiber to the home, making significant investments in upgrades to their capabilities.

Sometimes fear of competitors will motivate investment even when the RoI is uncertain. When the telephone companies were first thinking about whether to move beyond simple deployment of DSL and contemplating advanced technologies that involve pulling fiber at least part way into their access networks, I asked an executive of one such company whether they had a clear model of what the return would be on this investment. He said more or less the following:

> *We actually don't have a clear business model for doing these upgrades. But we have a clear model for what happens if we don't do these upgrades: in 10 years we are out of the wireline business. Better to bet on uncertain success than certain failure.*

On the other hand, a different pattern of competition, based on a different sort of regulation, can damp the incentive to invest. In other parts of the world, regulation has required that owners of access facilities (in particular the copper pairs of the telephone companies) must lease these facilities to other firms so as to create retail competition over these shared facilities. In general, the regulated rates at which these circuits must be shared has been set low to stimulate entrance into the retail market, but the facilities-based telephone companies have complained, with some justification, that they see no motivate to invest in upgrades if they carry all the risk, and at the same time gain no competitive advantage and a low RoI.

Suitable pricing structures: The current pricing structure for retail (residential) Internet access is probably *not* the best structure to incentivize investment in upgrades. Flat-rate pricing ("all you can eat/send") means that as usage goes up, the providers get no incremental revenues. Indeed, with the massive increases in usage we see today associated with streaming video, we see access ISPs moving to usage caps or tiers, which would either limit the need to upgrade or reward the ISPs for such upgrades. This transition, which has mostly happened in the mobile space, is probably inevitable in the wireline case as well. I once had an operator (from another part of the world) quite explicitly say that in a flat-rate context, they only invest in upgrades where there is a facilities competition; in other locations they see no reason to improve the service.

Architecture? The factors above are understood to influence the incentive to invest. But what about architecture. Again, in general terms, architecture can increase incentive by reducing risk and creating opportunity. It can

reduce risk if it can make the relationship among the actors clear and stable. (Of course, the fundamental goal of architecture is to produce an Internet that is fit for purpose, but that goal is more basic than just improving RoI.) Architecture can potentially improve opportunities by the creative but careful design of features that add to the range of service offerings.

What we see in the market today is that ISPs are attempting to increase the diversity of their product offerings by innovation at a layer that exploits the IP technology, but not just in support of the global Internet. IP technology is being used by ISPs to offer a range of services, including VoIP, IPTV, and so on, as well as Internet access. In the early days of the Internet, the designers did not see this distinction as they were designing protocols. The use of the Internet Protocol (IP) was equated to the public Internet. With the emergence of IP as a converged platform for a range of service offerings, a new architectural question emerges–should the distinction between the packet forwarding mechanisms implied by IP (which can be used in a number of ways) and the creation of the global Internet (which depends, among other things, on the specification of the network-network interface, be made more explicit in the architecture?

To a large extent, this distinction, which is becoming very important in practice today, is not emphasized in any of the architectural proposals that I discuss in Chapter 5. One could take this as a criticism of the network research community, which continues its focus on how the data plane (packet forwarding) works, rather than the sort of issues I discuss in this chapter. At the present time, it would seem that further exploration of this issue, both policy and architecture, is an open research question. For a deeper discussion of the issues, see a paper by my co-author and me [Claffy and Clark, 2014].

9.3 Money flows

If architecture defines the industry structure, and at least some of the features of the relationship among the actors, the next question is what defines how money flows among these actors. Here are two stories:

A while back I had a conversation with a well-known economist that studied the Internet. It went like this:

Economist: "The Internet is about routing money. Routing packets is a side-effect. You screwed up the money-routing protocols."
Me: "I did not design any money-routing protocols!"
Economist: "That's what I said."

And another story–the creation myth of revenue-neutral peering.

It is said that when the first two commercial ISPs met to negotiate their interconnection, one of the engineer-businessmen who was at the meeting was heard to say: "Wait, I thought *you* were going to pay *me* money". They discovered that they did not even have a common basis to agree on which direction the money should flow, let along how to set an amount. Then, as the story goes, these engineers heard the sound of running feet and realized that the lawyers were coming, to start a three year negotiation over the interconnection agreement. They looked at each other and said: "Quick, before the lawyers get here, lets agree that neither of us pays the other; we just interconnect; shake hands." And thus, so the story goes, was revenue neutral peering born.

So should we, as The Economist said, have designed "money-routing" protocols? There have actually been attempts to add "money-routing" to the Internet architecture.

One of the earliest proposal was [MacKie-Mason and Varian, 1996]. This paper, one of the first to explore the economics of the Internet, proposed that in addition to a possible fixed cost for sending data, there should be a *congestion price*, which reflects the cost one user imposes on another by sending when the network is fully loaded. Their approach was a *smart market*, where users specify their willingness to pay, but the price charged to the users that send is the price specified by the marginal user–the user with the lowest willingness to pay that can be accommodated. This idea, which is a form of a Vickery market, provides an incentive for the user to disclose their

true willingness to pay, since they will not be charged that price unless that is the minimum price that gains them admission.

They describe this scheme as preliminary, and there are indeed issues–for example, does the user want to pay for individual packets or for an overall transfer. The authors were clear that there were many details and modifications that would arise were this to be implemented. The high-level point is that they proposed that this price be in the packet header–they were proposing an architectural component that realized a pricing scheme–money routing.

There were a number of other pricing schemes proposed in the1990's. I dabbled in this area. In 1989, I wrote RFC 1102, on Policy Routing, which included a flag in the control-plane routing assertions to control whether the ISP should be paid by the originating ISP, the terminating ISP, or separately by the sender (as was then happening with long distance telephone service.) In 1995 I proposed a more complex scheme, which described a more complex marking scheme in the packet header to refine the direction of money flow as packets are forwarded [Clark, 1997].[3]

Needless to say, none of these proposals went anywhere.

Architecture and money flows

With respect to routing money, as with other objectives, if architecture has any role to play, it is not defining how the system works, but rather making it possible for a desired range of things to happen. Like the TTL field in the IP header, which allows the Internet to exploit routing protocols that induce temporary routing loops, perhaps a "direction of payment" flag in the header might allow a range of billing models that would otherwise not be possible. However, the networking community has so little experience with "money routing" that this area is essentially uncharted territory. In general, architects learn what should or could be built into the core of a system by looking at what application designers have tried to do in earlier generations. So, given perhaps two decades of experience with the commercialization of the Internet, what lessons can be learned about money flows? I find the lessons both uninformative and troubling.

We have seen a number of changes in the pattern of money flows among ISPs since the entrance of commercial ISPs into the ecosystem. The early pattern of routing among ASes was transit: small ISPs paid large, wide area ISPs to carry their traffic to its destination. Payment went from the customer to the provider, independent of direction of the packet flow. Packet counts were sufficient as input to the billing system. The idea of "sender pays" vs. "receiver pays" (by analogy to the 800 numbers in the telephone system), never emerged and did not seem of any interest. One could argue that it might have emerged if the necessary flags in the traffic had been there from the start, but conversations with ISPs suggest that the simplicity of the bulk payment for transit, compared to the complexity of a scheme with more discrimination, was not seen as worth the trouble. In any rate, there was never a way for the idea to be tried in the market.

Then the payment system shifted as a result of the increased use of peering among ISPs. Peering (providing a interconnection between two ISPs so each can have access to the customers of the other) emerged as a revenue-neutral scheme, as I noted above. In parallel, the telephone system was moving in this direction, away from schemes for interconnection "settlement" (where the telephone companies paid each other for traffic they carried) to what was called a "bill and keep" pattern, where the customers paid their local telephone company, and each company retained those payments. Calls were exchanged on a revenue-neutral basis. Given that the telephone system was moving toward this revenue-neutral pattern of interconnection, there was little motivation to bring a more complex scheme into the Internet, given that it would require a complex set of inter-provider agreements, which are hard to negotiate, and can (without care) lead to fears of anti-trust sanctions.

The next shift in Internet payment models has now emerged with the emergence of direct interconnection between content providers and access ISPs for the delivery of high-volume traffic such as video. From a routing perspective, these connections resemble peering connections, but after some very public disputes, the payment

[3] *The various papers in the anthology from which that paper comes give an excellent snapshot of the state of understanding of Internet economics in 1995.*

model that emerged was that the content provided paid the access ISP for the dedicated, high-capacity direct interconnections. The unstated assumption that was embedded deeply in these negotiations was the the value flow matched the packet flow–that is, the sender paid the receiver. One might have argued that when a viewer watches a video, it is the viewer that is extracting the value, rather than the sender, but this idea does not seem to have emerged in any of the negotiations. There was no question in the minds of any of the negotiating parties as to which way the money was flowing–only disagreement about the rate. No packet marking is needed to inform that sort of dispute.

We now see more complex models emerging for payment between content providers and access ISPs, for example the idea of "zero-rating", where a user with a monthly usage quota can receive certain content without it counting against that quota, because the sender has paid the ISP to deliver this content. This concept is most common in mobile access networks, and without pretending that I understand the details of how it is implemented there, the complexity of the cellular networks (with rich control mechanisms, often layers of nested headers and the like) provides lots of tools to implement any required packet marking. The important observation about this context is that it is a local, bi-lateral context between the two actors, there is no requirement that the mechanisms for implementing the accounting for traffic need work across the global Internet.

The next stage in billing may be schemes that attempt to link payment by content providers to the value of the content being delivered, rather than some formula that derives from the cost of delivery. Value pricing represents an attempt by the access providers to extract revenues based on what is delivered, and its perceived value to the sender. For example, a recent paper [Courcoubetis et al., 2016] describes an analysis of the different services provided by Google, and attempts to model the value per byte of the different flows (e.g., search vs. Youtube) so as to set a per-service fee for delivery. Whether one views this as the next stage in the future of access network funding or a throwback to telephone era settlement, the traffic classification is not going to be derived from any simple marking in the header, but by looking at fields such as port numbers, IP source addresses and the like. (In the language of Chapter 4, this is *implicit parameterization.*)

The idea of service-specific billing is not restricted to zero-rating for cellular access to the global Internet. An interconnection architecture called Internet Protocol eXchange (IPX, not to be confused with Internet eXchange Point, or IXP, nor to be confused with Internetwork Packet Exchange) has been deployed for interconnection of private IP networks (not the global, public Internet), networks that support services such as carrier grade VoIP. It is used by the cellular industry to interconnect their VoIP services (which are over IP but *not* over the Internet. IPX contains explicit support for per-service interconnection and cascading payments.[4]

9.4 Bad outcomes in the future

Per-service settlement for Internet interconnection may be an example of the pessimistic fears about the future that I articulated earlier in this chapter. We see a number of forces today that might drive the future of the Internet into a darker place, including pressures from those who carry out surveillance and regulation of access to content, but economics is probably the most potent driver. As long as the Internet is a creature of the private sector, profit-seeking behavior is a natural behavior and what we must expect.

In this respect, it may be that some architects will prefer to push for designs with *less* expressive power, to prevent the implementation of sophisticated toll-booths in the Internet. Architecture is not value-free, and this is an excellent example of a place where values will be close to the surface as we make design decisions.

It may not be possible for the private sector to recover the costs of building expensive infrastructure, if our goal is an open network. In the long run, we may need to think about networks the way we think about roads–as a public sector undertaking. Alternatively, if we do not accept that facilities investment can derive from public sector involvement, then designs that restrict the opportunities to reap returns on facilities investment may limit revenues to the point that facilities buildout does not occur at a suitable rate. Some degree of vertical

[4] *For a good tutorial on IPX, Wikipedia is perhaps good place to start. See https://en.wikipedia.org/wiki/IP_exchange.*

integration or discriminatory treatment of traffic (including billing) may be the price of a healthy infrastructure. Any architecture must think carefully about the extent to which it attempts to take an ex anti position on this point. If an access provider supports both a healthy open Internet and as well other services over the same IP infrastructure, is this a driver of investment or a threat to that Internet service?

9.5 Summary–architecture and economics

Architecture indeed plays a key role in the economics of an Internet ecosystem. Architecture defines the industry structure, and the key interfaces define the potential relationships between the actors. It is less clear what role, if any, architecture should play in enabling different money-routing protocols. It may be necessary to wait another 20 years to see how the economics plays out before we can see what we should have added to the architecture now. Perhaps even with the current Internet, in 20 years we may not have the ISPs that we have today, but a different set of actors trying to work within the existing architecture. If so, it will be economic forces that will make that future happen.

Chapter 10

Network Management and Control

10.1 Introduction

One of the requirements that I listed for a future Internet in Chapter 2 was that it do a better job of dealing with network management and control, which has been a weak aspect of the Internet from the beginning. In this chapter, I will take what is admittedly a somewhat preliminary try at linking these requirements to network architecture.

Using an approach that mirrors what I have done in other chapters, I will pose two key questions that will help to sort out this space:

- What is implied by the term "management"?
- What does architecture have to do with management?

I start with the first question.

10.2 What is management?

Management is in one respect similar to security–the word is ill-defined. I began the discussion of security by asserting that the term "security" was so general that it was aspirational, not operational. I argued that only when we find a substructure for security can we begin to understand the relationships (and potential conflicts) among the sub-goals, and thus understand how to improve security in practice. I believe that this same observation will be true about management. While I will identify some common themes that run through different aspects of management, I will argue that the substructure of management contains distinct objectives that need to be considered separately.

One definition of network management is that management encompasses those aspect of network operation that involve a human. We often talk about the "data plane" of the network, which is that set of mechanisms that actually forward the packets, and the "control plane", which is that set of automatic mechanisms (such as routing and congestion control) that provide the information necessary for the data plane to function. An important part of that definition is that the control plane is automatic–there are no people in the loop. In this framing, management is those set of actions where a person *needs* to be in the loop. However, while this definition may unify the concept from one perspective, there is no reason to think that the totality of issues that require human intervention are homogeneous with respect to network architecture. I will argue that the answer is quite the opposite.

From a design perspective, there are two points of view about management: views that define the ends of a spectrum. At one end are designers who say that a properly designed network should run itself, so the need for management is a signal of failure. At the other end are pragmatists who believe that what they would call "policy decisions" are not worth trying to codify into algorithms, and having a human in the loop for many decisions is the better (and more realistic) way to go. Part of this debate resolves itself when we look at time-constants of

the human intervention. I suspect many designers (and operators) would be skeptical about a network that so thoroughly ran itself that it put purchase orders in for new circuits and routers when it saw itself getting near capacity. Business issues would seem to call for human judgment.[1] On the other hand, a network that requires teams of humans to sit in front of monitors 24 hours a day looking for faults would seem to benefit from some further automation.

Breaking network management into parts

As a starting point to study the sub-structure of network management, it is useful to see what others have already done. The ISO has in fact come up with a standard [CCITT, 1992] that both defines network management and breaks it into parts. Their definition of network management lists the following objectives:

- activities which enable managers to plan, organize, supervise, control and account for the use of interconnection services;
- the ability to respond to changing requirements;
- facilities to ensure predictable communications behaviour; and
- facilities which provide for information protection and for the authentication of sources of, and destinations for, transmitted data.

They then divide the set of management issues into the following categories:

- fault management;
- configuration management;
- accounting management;
- performance management;
- security management.

This set of categories is called the FCAPS framework for management, based on the first letters of the categories.

What is in common among these five categories, as defined by the ISO, is that all of them are based on the reporting of data and events from managed devices in the network to a management system. The information so reported can be fed into a simple display or a complex management system that gives the user a higher-level view of the network. A management system also allows the user to send configuration/control messages to a managed device–again either from a simple interface or a high-level management application.

The ITU specification is based on the assumption that while the different sorts of management categories have different objectives and requirements, the protocols for reporting and control can be the same. This assumption superficially seems to match what we see in the Internet, where a common protocol for reading and writing management variables (Simple Network Management Protocol or SNMP) is used with a variety of Management Information Bases or MIBs for a variety of purposes. (The Internet defines a Management Information Base or MIB to specify the variables associated with each class of managed device.) However, this assumption of a common protocol should be carefully inspected for flaws. In the early days of work on Internet management, there was a

[1] *However, when management decisions become embedded in regulatory orders, the results become almost algorithmic. In the merger between Time Warner and Charter, the new entity, as part of its agreement to provide revenue neutral interconnection to qualifying peers, included the following term: "Either New Charter or the interconnection party can require that the capacity at any interconnection point: (i) be upgraded if aggregate port utilization in either direction at the interconnection point exceeds 70% of the available capacity at the interconnection point for 3 or more consecutive hours per day for 3 or more consecutive days during the preceding 30 day period"*

lot of attention devoted to the problem of communicating with a managed device when the network was failing and communication was impaired. There was a concern that using a protocol like TCP, which insists on reliable, ordered delivery, would not be workable for communication in a situation of failing communications. This fear led to the idea that the communication method used for fault management should be designed for the special case of operation in impaired conditions. This situation is quite different from (say) the protocol to configure a complex router, which may involve the installation of thousands of rules, and would seem to call for a reliable, sequenced update.

However, setting this specific issue aside, the above discussion suggests a possible modularity to the problem of network management. At the bottom layer, at the managed device, there are parameters that can be read and set, and these are specific to the problem at hand. Above this is a (perhaps common) protocol for communication between the managed device and any higher-layer management system. The management system in turn provides an interface to the people responsible for management. The management system provides the human operator with what the military calls "situational awareness": the understanding of what is happening in the network.

Management and control

Both in the Internet and in the ISO framework, the concept of exposing parameters in a device for reading (and perhaps writing) is typically associated with management functions, not the control plane. In the Internet, we have SNMP, the Simple Network *Management* Protocol. The Internet design community has not focused on whether the control plane should have similar interfaces (and perhaps, by analogy to MIBs, should have Control Interface Bases, or CIBs). I believe that this way of thinking has been shaped by the particular approach to the design of the control plane in the early Internet, which is that control protocols run on the devices being controlled, and each device reads and sets its own control variables as a part of executing that protocol. The most obvious example of this is routing protocols, where devices exchange connectivity data, but each device computes and populates its own forwarding table. There is no control interface on a router where it can expose the results of its low-level link measurements, nor an interface to set the forwarding table (except for low-level human interface (CLI) tools to fill in manual entries.)

We now see a trend in the Internet toward a new generation of control algorithm that moves the control computation out of the distributed devices into a more centralized controller. The most obvious example of this is Software Defined Networking (SDN), where network connectivity data is collected in a central controller that computes and downloads forwarding data for the different routers in the network. (I mention another example below of a move to a more explicit control algorithm in the discussion of performance management.) As a part of the development of SDN, it was necessary to define an interface between the router or switch and the route computation function, which both specified the set of variable that could be read and manipulated using this interface and the protocols used to implement this function.

Given this trend, when we consider the relationship between management and network architecture, I will generalize the consideration to include both management and control functions. While the distinction between management (involving people) and control (automated functions) may be helpful in some contexts, it is not material in other respects. If a device exports certain variable for reading and manipulation, it is not material to the role those variables play whether they are manipulated by a person, by an algorithm, or by an algorithm that may sometimes have a person in the loop.

The addition of control to this analysis will add new categories to the five I listed above. Routing and congestion control are the most obvious.

Active probing

The conception of management (and control) as a process driven by data gathered from managed entities is a somewhat limited view. Other methods can be used to assess the state of the network, for example active probing and observation of end-to-end behavior. The most critical control algorithm in the Internet, congestion control, is not driven by data explicitly gathered from each of the routers along the path, but from observing end-to-end

behavior. There have been any number of papers written on how congestion control might be done better than it is now is, given our improved understanding, but while most of these depend on carrying more information in the packet, none that I can think of exploit localizing where congestion is happening to a specific element along the path.

The most basic tools of active probing in today's Internet are *ping* and *traceroute*. They surely qualify as management tools, since they are used by people, often ordinary users frustrated with the state of the network. As well, these tools are regularly used by professional network managers. This sort of probing serves a number of purposes. The two most obvious are understanding performance (often when performance is poor) and fault localization. The original purpose of *traceroute* was configuration–determining the path a packet was taking on its way to the destination. Ping (or more precisely the ICMP Echo option) was designed for the purpose for which it is used, but *traceroute* is a kludge, a hack that uses a packet crafted with a TTL that expires at a point along the path to solicit a response from that element. This tool is indeed used to learn something about the different elements along the path, but implicitly. The ICMP response was never intended for this purpose, and the measurement community has struggled to make sense of the responses, dealing with issues such as de-aliasing the IP address on the return packets, variable processing delay in the control processor of the probed router, and so on. Had the tool been designed for purpose, it might provide information that is easier to analyze, for example a unique ID associated with a router, independent of which port is probed.[2]

Of course, one problem with tools like *ping* and *traceroute* is that the probed element sometimes does not answer. Routers are sometimes configured *not* to answer, for both performance and security reasons. The Internet is composed of regions (ASes) operated by different entities, sometimes not interested in having their insides probed by outsiders. Most operators have come to understand that having their routers respond to a *traceroute* is a useful mutual agreement, but not all operators buy into the agreement at all times. It is important to remember that measurement, especially measurement that reaches beyond the scope of the region that owns the elements in question, is sometimes a adversarial activity, and often an action with political motives.

Very few of the architectures I have discussed in this book give much attention to the question of whether there should be tools in the architecture (fields in packets or additional probing functions) to facilitate active measurement of the network, whether to deal with issue of configuration, performance or fault localization. But the design of any such mechanism would bring into focus an important aspect of tussle, which is that many operators would prefer to keep the details of these issues to themselves. Active probing can be seen as a case of trying to discover from the outside something that is probably already known on the inside, but is not being reported.

If operators are (understandably) sometimes reticent about the status of their region of the network, perhaps a way to balance the interests of all parties would be to define, as part of a management architecture, an abstract view of a region of a network, which hides some details (perhaps the exact topologies of routers, for example) but gives a abstracted view of the current state of the region.[3] If there were a uniform agreement on a set of useful abstract parameters to report, this outcome might represent a resolution of the tussle issues around the desire of all parties to probe their neighbors.

Packet sampling

Another important tool for network management is packet sampling. Tools such as Internet Protocol Flow Information eXport (IPFIX) and its kin sample the packets passing through a router to identify the flow, reporting data such as source and destination IP addresses, number of packets, and so on. The necessity to sample does add

[2] *The IETF, in an attempt to deal with the issue of variable latency in the reply to a probe, has developed the Two Way Active Measurement Protocol or TWAMP mechanism (see RFC 5357). A probing mechanism that combines the features of TWAMP with traceroute might be of great benefit to the measurement community, so long as it cannot be abused.*

[3] *Some operators do provide this sort of abstract view of their network: for example AT&T has a web site where they list the current latency between all their city pairs: see https://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html. It is not clear if this exact abstraction of latency is the most useful, nor is there a uniform agreement among all ISPs to measure and report this parameter, but it illustrates the point.*

uncertainty to the data being reported, but flow data is a rich source of information that can inform performance and configuration management, and in some cases security. It is a good example of an entirely new sort of data reporting in support of management, going beyond the simple counters normally reported using SNMP. It is also an example of a tool that was developed without any support from the architecture. Again, one could ask if some additional information in the packet header could enrich what can be learned from sampling the packets going through a router.

Management of the management system

Management (and control) systems themselves have to be configured and monitored for correct operation. This sounds painfully recursive, and in some cases may be, but the specific cases are usually dealt with in pragmatic ways.

Specifically, how does a management system discover the set of entities that are to be managed? If a managed element can generate alerts (active notification of a change in status of one of its management variables), to which element should alerts be sent? These questions will come up as I look at the categories of management and control. Further, there is a significant security aspect to these questions. What entity should be allowed to manage another entity? Reading of management variables may seem less harmful than malicious modification, but could lead to undesirable revelation of system status, and flooding of a management interface with queries can potentially lead to overload of a managed system (a form of DoS attack). Obviously, malicious modification of management/control variables can lead to a wide range of disruptive outcomes.

The design of any management mechanism must be accompanied by an analysis of how that system is managed and controlled, and the security implications of that system.

10.3 The role of network architecture

The previous discussion suggests that the ISO model of management (the export of management parameters from a device) is not the only aspect of management to consider. However, in that context, are there any components of the system that might rise to the level of architecture? One component might be the bottom layer, where the variables that can be read (and written) to monitor and control the network are defined. These parameters are the foundation for building situational awareness and control of the network–different management systems can be built on top of them, but if the basic data is not available, management will be difficult and flawed. What we should expect to find in the five different sub-classes of management listed above are very different management variables related to the specific tasks. In this respect, the different categories of management may be differentiated. By looking at the previous chapters, we can try to extract insights about how the data and control planes should be designed to provide the most useful information to management. Then we can ask to what extent these considerations relate to architecture, as we have defined it.

In the current Internet, there are no management parameters that come anywhere close to being considered "architecture" by the Internet designers. But if some of those parameters achieve the status of "something on which we all need to agree", then they should properly be considered part of the architecture, even if they earn that status after the fact. In chapter 1, I claimed that a key aspect of architecture is the definition of interfaces. Interfaces modularize the system, and specify what is shared among the entities on the different sides of the interface. The set of parameters exported by a managed device define its management interface, and if there are classes of devices which benefit from an interface that is globally agreed and stable over time, that interface takes on the character of architecture. The exact protocol used to implement the interface may change, but the expectations that the two entities make of each other, defined by the parameters that can be read and written, is a more basic and enduring characteristic.

Instrumenting the data plane

The previous discussion of packet sampling provides a particular illustration of a more general issue: should the data plane of an architecture contain elements that are designed to assist in some aspect of network management?

Are there values that might be put into a packet header (e.g., a flow identifier) that would help with performance analysis? Could the data plane be re-engineered to help with fault isolation?

In the early days of the design of the Internet, colleagues from the telephone company (when they were not telling us that packet switching would not work) strongly advised us that the data plane had to include tools for fault diagnosis. The digital version of the telephone system, which had been engineered only after the telephone system itself was very mature, included features (such as management fields in the data frames that were forwarded in their TDM scheme) that were intended for fault diagnosis. They argued that the ability of a network to diagnose its faults was just as important as its ability to forward its data, and that our failure to appreciate this was just another signal of our inexperience.

If the data plane could somehow be enhanced so that it supported management issues as a part of its normal function (for example supported fault localization or detection of performance impairments) this would shift tussle in a fundamental way. Operators can try to distort what is discovered using active probing (giving *ping* packets priority or refusing to respond to them), but it is harder to distort what is observed by the data plane without distorting how it performs. Of course, if there are fields in the data packet that the router is expected to fill in that only play a role in management (such as the traceback schemes for localizing DDoS attacks) the operator of the distant router can just choose not to implement this function. The ideal tool for instrumenting the data plane will be a feature designed so that it is an inherent part of the forwarding process. This goal, in the presence of tussle, is a significant design challenge that calls for crafty thinking.

State and the dynamics of the control plane

Dynamic control requires some ability to sense the environment–feedback. And for feedback to be effective, it has to act on some state in the system. There is some parameter being controlled, and feedback adjusts that parameter. This is a very general statement of dynamic control, but the essential point is that a stateless element cannot be a controlled element, because it has no state to be controlled. The original design of the Internet strove to minimize the state in the routers, in pursuit of simplicity. Routers keep no track of the packets they forward, other than to count the total number of bytes and packets forwarded in some interval. There is no "per flow" state in routers, which simplified the steps of forwarding of packets. [4] Since packet forwarding must be highly efficient (routers today are required to forward hundreds of millions of packets per second out each port), if the Internet could be made to work without keeping state in routers, so much the better.

What we have learned over the years (but perhaps learned in a fragmentary way) is that even if state in the router is not required for actual forwarding, it may be necessary for control. And adequate control of the network and its resources is a prerequisite for the prime requirement of forwarding.

Congestion control Congestion control makes an excellent case study of different approaches that require different sorts of state in different elements, and different dynamic control mechanisms. When, in the late 1980's, Van Jacobson proposed the congestion control algorithm that is still in use in the Internet today [Jacobson, 1988], one of his central challenges was to find a useful control variable. The software that implements the IP layer (not only in the router but in the sending host) does not have any useful state variables. However, the Transmission Control Protocol (TCP) which many applications use and which runs in the end-points "above" the IP layer, has a control variable (the so-called "window") that is the number of packets that the sender is allowed to have in flight across the Internet at any instant. When an acknowledgement is received at the sender telling it that one of its packets has been received at the destination, then it is allowed to send another packet into the system. [5] This

[4] *In the early days of router design, engineers explored the idea of keeping a lookup table of destination address to which packets had recently been forwarded, so as to potentially reduce the cost of looking up the destination address from each packet in the full forwarding table. The idea seems to have been more complexity than it was worth.*

[5] *To over-simplify how "window-based" flow control works, assume that the round-trip time from sender to receiver is constant. A sender should receive an acknowledgement of a packet one round trip later, so the resulting sending rate is the window size divided by the round trip time. If the window size is 10 packets, and the round trip is .1 seconds, then the sending rate is 10 packets every .1 seconds, or 100 packets/second.*

simple feedback loop was initially designed to keep a sender of packets from overwhelming a receiver, but what Jacobson realized was that it could be modified to deal with overload in the network as well as overload at the receiver.

The scheme that Jacobson devised made minimal assumptions about the functionality in the routers. It worked as follows. When offered traffic at a router exceeds the outgoing capacity, a queue of packets forms, which are stored in the router. When storage is exhausted, the router must discard an incoming packet. TCP keeps track of lost packets and retransmits them, but as part of the congestion control scheme Jacobson proposed, it cuts its sending window (the number of packets it can have in flight) in half, thus reducing it sending rate and the resulting congestion. It then slowly increases its sending window until it again triggers a queue overflow and loss of another packet, when the pattern repeats. This algorithm is a very simple control loop: the sender continuously hunts for the acceptable speed at which to send, slowly increasing it sending rate until it triggers a signal that tells it to slow down.

When Jacobson proposed this scheme, several of us asked why he used the actual dropping of a packet as a congestion signal rather than some explicit control message. His answer was insightful. He said that if he proposed some complex scheme that the router should implement in order to determine when to send a "congestion detected" signal, coders would almost certainly mis-code it. But there is no coding error that can allow a router to avoid dropping a packet if it has actually run out of memory. None the less, the research community set about trying to design a mechanism called *explicit congestion notification*, or ECN, to allow the router to signal to a sender that it should slow down even before it had to drop a packet. The design of ECN was complicated by the fact that the packet lacked a field in which to carry the ECN indication (a lack of what I have called *expressive power*), so most of the design effort went into figuring out how to repurpose an existing field in the packet so it could be used for this purpose. Even today, ECN has not gained wide use in the Internet, and the signal to a sender to slow down is still a dropped packet.

While Jacobson's scheme has worked very well, and has been critical to the success of the Internet, it raises several issues. First, not all applications use TCP as a transport protocol. Other protocols might not react to a dropped packet in the same way. Protocols that stream real-time traffic (audio and video) normally send at a constant rate (the encoding rate of the content) and are designed to mask the consequence of a dropped packet. The idea that they should slow down in response to a lost packet is not consistent with the need to send the encoded content (e.g., the speech) at the rate it is encoded. Further, the TCP congestion adaptation algorithm (the cutting of the window size by two on a lost packet) is implemented in the end-node. There is no way the network can detect if it has actually done so. What if a malicious user just patches his code so it omits this step? He will continue sending faster and faster, other (more obedient) senders will keep slowing down, and the result will be a very unfair allocation of capacity to the different senders.

More generally, this scheme, since it acts on individual TCP flows, makes a single TCP flow the unit of capacity allocation. What if a sender just opens two TCP flows in parallel? He will then be able to go twice as fast. Today we see web servers often opening many TCP flows in parallel, although the goal is not usually to thwart congestion control but to deal with other limits to throughput. The more philosophical (or architectural) question is how should capacity (when it is scarce) be allocated. Per TCP flow? Per sender, independent of how many TCP flows that sender has? Per sender to a given destination? And so on. Early (and persistent) debates around these questions yielded the following understanding of the situation. First, there is a tussle between the user and multiple ISP over control. An ISP might assert that since it owns the resources, and has a service agreement with the user, it should be allowed to determine how resources are allocated among users. But perhaps the congestion is occurring at a distant point in the Internet, where the ISP actually dealing with the congestion has no service agreement with any of the users contributing to the congestion. Users might assert that they should have some control over which of their flows they choose to slow in response to congestion. These debates suggested that the answer as to how scarce capacity is allocated (to flows, to users, to destinations, and so on) might be different in different contexts, and should thus *not* be specified by the architecture. These debates suggested as well that we as designers did not understand (or at least did not agree on) how to build a more general mechanism, and

TCP-base per-flow congestion feedback is still the practice today.

There have been a number of proposals either to improve the Jacobson algorithm (some of which have seen limited deployment) or to replace it (all of which resulted in an academic publication but no transformation of the existing Internet). These schemes are usually defended by evidence that they increase performance, but it is interesting to look at them through the lens of the control state they require, as well as expressive power in the packet.

Jacobson's scheme uses the queue of packets as a crude state variable–when the queue overflows, the dropped packet is the control signal. Several schemes have been proposed to use the length of the queue (the number of packets held there at the moment) as a more sophisticated basis to generate a congestion indication. The scheme called RED (which stands for Random Early Detection or Random Early Drop) [Floyd and Jacobson, 1993], picked packets from the queue to drop even before the queue was full, selecting them at random as they arrived, with increasing probability as the queue grew. With proper setting of the parameters that controlled the probability of dropping, this scheme had many benefits compared to waiting until the queue was actually full, but it proved difficult to set those parameters correctly in an automatic manner depending on the operating conditions, and the need for manual configuration (e.g, congestion *management*) somewhat limited the deployment of RED.

Later schemes have tried to improve this approach, and eliminate the need for any manual configuration. These include CoDel [Nichols and Jacobson, 2012] and Pi (proportional-integral controller) schemes. There is a vast literature (and I use that term without exaggeration) on congestion and its control, which I do not attempt to even start to review here. From the perspective of architecture, what is interesting is the assumptions that these schemes make about state and expressive power. When CoDel was being developed and evaluated, it became clear that for best operation, it was useful to have per-flow state in the routers–in fact to keep a per-flow queue of packets that is serviced according to some proportional scheme. This idea was put forward as a minor extension to CoDel, but it might instead have been seen as a fundamental change in our architectural assumptions about state. One could ask, looking generally at mechanism and function, what might all the benefits of per-flow state be, in the various control functions of the Internet, if one were to accept the cost and complexity of implementing it.[6] For example, once routers support per-flow state, could this be used as part of a mitigation scheme for DDoS attacks, as I discuss in Chapter 7?

As I hinted above, there have been more "academic" proposals to improve congestion control–"academic" in the sense that they require more modification to the architecture (more *expressive power* in the header) and thus were not intended as schemes to be deployed as described in the current Internet. The eXplicit Control Protocol (XCP) [Katabi et al., 2002] puts per-flow congestion state in the packet, to avoid having this state in the router. To over-simplify (as I have done in many places) the sender puts its send window value in the packet, so that the routers, using a suitable algorithm), can directly modify that window. The Rate Control Protocol [Dukkipati, 2008] takes a similar approach, putting a rate variable into each packet so that there is no per-flow control state in the router. Schemes like these require that a router estimate the round trip of traffic flowing out over each of its links–for a control loop to be stable there needs to be some measure of the delay in the control loop. As well, these schemes do not define what a flow is–they operate on individual packets based on the assumption that a flow is that set of packets that share a common sending window parameter.

Another framework to consider congestion control is re-feedback or re-ecn, proposed by Briscoe. [7] Re-ecn is a scheme that tries to take into account the incentives of the different actors (both senders and ISPs) to deal with congestion, and allows for the inclusion in the scheme of a *policer* that the ISP can use to detect if the user is responding correctly to the congestion signals received from the network. Using the framing of this book, re-ecn adds a new sort of PHB to the architecture to control congestion in ways that reflect the potential tussle between

[6] *In fact, routers today support this sort of function–it has crept into the implementations without being considered from a more fundamental or architectural perspective.*

[7] *For the reader interested in the re-ecn work, which is not just a protocol but a proposal to reframe the point of view used to think about congestion, a good place to start is the web page maintained by Briscoe at http://www.bobbriscoe.net/projects/refb/.*

sender and ISP.

State and network management It is interesting to note that essentially none of the architectural proposals that I reviewed in Chapter 5 discuss congestion control, or more specifically what expressive power in the header might be useful for this purpose. They do not discuss the balance of what needs to be architected as opposed to what needs to be changed in different contexts. NDN is an interesting alternative in this respect. NDN is based on an architectural decision to have in each router not just per-flow state but per-packet state. Whenever an interest packet is forwarded by a router, it makes a record of that event. If and when a data packet returns, it is matched to that state. It logs the time the interest packet arrives, so that it can measure the time between the interest and data packet, thus providing a potential control loop with an estimate of the delay time in the loop. This mechanism, once in place, can be used for a variety of control functions, giving NDN a very powerful capability that other architectures with less state in the router cannot implement. NDN can implement per-hop forms of congestion control by limiting the number of pending interest packets. It can perform routing experiments by learning which paths over which an interest is forwarded actually triggers a data packet in return. While NDN is often described in terms of its forwarding function, its ability to implement a range of novel control algorithms is an equally important aspect of its architectural fundamentals.

Some of the Active Network proposals discussed in Section 5.3 allow the active packets (packets with code that is executed as the packet arrives at a node) to create transient state that can be used for more sophisticated network control. The PLANet scheme [Hicks et al., 1999] discusses the use of active code to create what they call *scout packets* that fan out to seek good routes to a destination, creating transient per-packet state to keep track of the scouting function, somewhat reminiscent of the transient state created by interest packets in NDN. There is an interesting contrast between NDN and PLAN. NDN is most emphatically not an Active Network approach–any algorithm that implements (for example) exploration of good routes would be a part of the software installed in the router. In PLAN, the source node implements the scouting code, and sends it to be evaluated by nodes along the path. The latter scheme does raise the question of what happens if different scouting programs from different source nodes are simultaneously exploring the network and taking perhaps independent decisions that end up having common consequences. But the approach illustrates a form of "end-to-end" thinking–in principle only the source knows what sort of service it is seeking, and thus what sort of scouting algorithm will best meet its needs.

Layering and control

In the earlier chapters, I have described the Internet as having a layered structure. The most simple layered model is a network technology layer at the bottom, the application layer at the top, and the Internet Protocol layer in the middle providing the service specification that links the technology to the application. This layering emerged in the context of the data forwarding function, and it is described in that way by most network designers. There is less consideration of what this layering means for critical control functions. In particular, there is seldom any discussion of what interfaces (definitions of how information is to be exchanged between the layers) is needed in the context of control. Fault control (or management) provides a good illustration of the issue. When a physical component fails, the consequence manifests at every layer. If there is no interface to allow coordination among the layers, every layer may independently initiate some sort of corrective action, which may conceivably conflict with each other. In today's Internet, we see complex technology layers below the IP layer. We see multi-hop sub-networks hooking routers together, where those networks themselves have forwarding elements in them, which are usually called *switches* to distinguish them from routers. These switches will have their own routing protocols, and when a fault occurs, both the technology layer protocols and the Internet routing protocols may undertake a re-routing effort, while the application might attempt to initiate a connection to a different end-point. I know of little discussion in the research community of how information exchange between the layers might improve this sort of situation. One possible answer might be a specification of the time within which any layer will undertake to fix a problem. A technology layer might be specified (or indicate through an interface to the

layer above) that if it can repair a problem it will do so within 100 ms, so the layer above should not undertake any reconfiguration until this period has elapsed.

Congestion control provides another example of the complexity that can arise in control mechanisms from a layered architecture. Consider again a complex technology layer with multiple switches routing packets among routers. What happens when congestion occurs at a switch rather than a router? How does the switch participate in the congestion control algorithm? In Jacobson's original and very insightful design, the switch just drops a packet. As he said, any element knows how to drop a packet. But if the scheme involves some use of the expressive power in the packet, then the switch has to have knowledge of that packet format, which means it has to be designed knowing how the Internet is specified–it has to function at the IP layer, not just a lower technology layer. This sort of example suggests that from the point of view of control, the idea of modularity based on layering may be less useful than a modularity based on the scope of a mechanism –is the mechanism operating within a region of the network or globally. We designed Internet routing this way–the Internet allows alternative routing protocols inside individual regions of the Internet, and uses the global BGP to hook these routing protocols together. The designers do not think of these as running at different layers, but at different scopes. While there can be alternative routing protocols used within different regions, the relationship (the interfaces, speaking abstractly) between these two protocols are well-understood if implicitly specified. The routing mechanisms we now find below this level (inside specific network technologies) are less-well linked to what happens at the Internet level. I think we will begin to see (and in places are starting to see) that these lower-layer mechanisms are becoming more explicitly "Internet aware"–they will be distinguished by the scope over which they function, not by being at a lower layer that is independent of the specification of the Internet.

I think the design of layer interfaces (and modularity generally) in support of control functions is a much understudied aspect of Internet architecture, and in the FIA projects it continued to receive less attention than the data forwarding functions.

10.4 Categories of management and control

In this section, I look at categories of management, starting with the ISO FCAPS list, and seek to identify architectural issues, and further explore what aspects of the management interface might rise to the level of architecture.

Fault management

The challenge of fault management has come up in various places earlier in this book, most directly in the discussion of security and availability.

In my discussion of availability, I proposed a high-level framework for understanding availability:

- It must be possible to detect the failure.
- it must be possible to localize the failed parts of the system.
- It must be possible to reconfigure the system to avoid depending on these parts.
- It must be possible to signal to some responsible party that a failure has occurred.

I noted at the time that putting these steps into the passive voice papered over huge issues: which entity was to carry out each of these tasks. Resolving these issues falls within the scope of fault management.

There are a variety of ways that a failure can be detected, involving different actors. In some cases, an element may be able to tell that it is failing and raise an alert. In this case, the question is where that alert should be directed. Some utterly feeble mechanisms have been devised for an element to indicate that it is failing, such as

turning on a small red light in the hope that a person will notice it (a wonderful example of a horrible management interface).[8]

Sometimes machines interacting on the network can detect that one of their number has failed. Most Internet routing protocols embed some assessment of correct function into the protocol itself (perhaps a simple "keep-alive" or "handshake" probe). The failure of this sort of handshake can be taken as a signal of failure, but then the question is whether one machine can be trusted to tell another machine that it is failing. In fact, the first machine may be failing, not the second machine, or the first machine may just be malicious.[9] Any machine that is told by another machine that it is failing must take that input with considerable caution. This is a case where it may make sense to have a person in the loop, but if rapid recovery is required, there is a tension between a quick response and a considered response. One of the benefit of a system with redundancy is that service can be restored quickly using redundant elements, while the failed element can be recovered more slowly.

The protocol for forwarding Internet email has a built-in redundancy/resilience mechanism. The DNS can list more than one IP address for a Mail Transfer Agent, so that if the first one is unresponsive the sender can try another one. However, there is no means for the sender detecting that the first receiving agent has failed to report that problem. The underlying problem might be repaired more quickly if the failure could be reported when it is detected, but again, there are a number of issues to be resolved for such a scheme to work. The first is to provide the address of the location where an error report should be sent. The second issue is to prevent this mechanism from being abused. The third issue is to deal with a possible flood of legitimate error reports when lots of senders detect at the same time that a receiver has failed.

A network could be equipped with several mechanisms to deal with these issues, which (since they seem to be global in scope) might rise to the level of architecture. One would be to include in the DNS a new kind of record that gives the name of the machine to which a failure of the intended service can be reported. The second would be some sort of "incast" mechanism to aggregate multiple error reports together as they flow across the network toward that reporting point. An incast scheme also limits the range of DoS attacks on the error reporting service.

In the most simple cases (consider a home network), I would propose that a standard method of logging errors within that context be a part of the basic configuration process. For example, Dynamic Host Configuration Protocol (see below) could be extended so that when a machine first connects to the network, it is given the address of a place to send fault reports. The home router could be the aggregation point for these messages, and such a framework could be part of a new service that allows for diagnosis of problems in the home network.

In the case of email, the two-party nature of the forwarding steps makes localization somewhat straightforward. However, in other cases (most obviously the failure of a packet to reach its destination in a timely manner), localization is much harder. Without the ability to localize the problem, it is much harder to resolve the problem by avoiding the failing component (one is reduced to trying other options more or less at random) and there is no possibility of reporting the fault. The tools the Internet has today to localize faults along the forwarding path are minimal: usually the only option is *traceroute*, with its many limitations. But as I noted above, it may not be in the best interest of a particular region of the network to let outsiders successfully localize faults within that region, and when the "fault" is due to the successful blocking of an attack, it is absolutely not in the best interest of the target of the attack that the attacker be able to diagnose the reason the attack has failed. I believe that fault localization is a much understudied and poorly understood but critical aspect of network design, which may have implications for architecture were it better understood.

[8] *In the era of early time-sharing, when I was coding the Multics system, the I/O controller had a management alerting channel, but if this failed, it reported the failure of its management interface by ringing a loud alarm bell. One's programming errors took on a somewhat public character. The management of the management system implies a sort of recursion that has to be resolved somehow.*

[9] *There is a famous rolling outage of the AT&T phone system which is similar in character to this pattern. One machine self-detected a fault and reset itself, and in recovering from this reset sent a sequence to its neighbor machines which (due to a bug) then reset themselves, and so on. It went on for nine hours [Neumann, 1990].*

Configuration management

Configuration is the process of setting up the elements of a system so that they can interwork properly. As a simple example, the Dynamic Host Configuration Protocol (DHCP) allows for the automatic configuration of a host when it is first attached to the Internet. DHCP changed initial host configuration from a manual and somewhat mysterious management task to an invisible control function hidden from the user. DHCP provides three critical pieces of information: an IP address for the new machine to use, the address of a router that can provide a path to the Internet, and the address of a DNS server that provides access to Domain Name resolution.

More complicated devices, such as production routers, have much more complex configuration requirements. To a variable degree, configuration of complex devices like routers is automated, but in some cases people end up doing device configuration from a command line.

It is not too hard to conceive a configuration interface that allows a managed device to be configured over the network. But there is a bootstrapping problem: how does the new device know what existing device on the network is allowed to configure it? There may be some necessary first step taken by a person, such as typing some validation information into the new machine. In simple cases, the process by which a new machine finds a service to configure it is itself automated. For example, in DHCP, the new machine broadcasts to find a DHCP server. But lurking inside these automatic discovery schemes that are part of many configuration protocols is a potential security problem. With DHCP, for example, the newly attached host requests configuration information by broadcasting its request and believing whatever machine answers. This mechanism is usually not a major vulnerability, but should serve as a reminder that the initial phase of configuration is a moment of vulnerability in system setup, whether the mechanism is DHCP, bluetooth peering or configuring devices in a smart home.

Accounting management

In Chapter 9 I discussed a range of schemes for "money-routing", which depended on new fields in packets, and presumably depended as well on new sorts of tools in routers to track and report usage of different sorts.

Operators today use fairly simple tools to gather data to inform accounting functions: packet and byte counts, data from samples of the packet stream (such as IPFIX) and so on. In 1991, as the first commercial ISPs were happening, the IETF looked at accounting, and published an RFC [Mills et al., 1991] that frames the problem. It discusses methods of reporting based on packet capture, and in many respects the state of the art does not seem to have advanced all that much. The RFC is cautionary with respect to inventing complex tools for accounting, lest they be used.

Performance management

Performance, as it relates to architecture, is not a simple matter of throughput between two end-points. Various of the proposals I have discussed in this book have implications for performance, but in very different ways that illustrate that performance is a multi-dimensional issue that will have different manifestations in different architectures. ALF was intended to improve host processing performance. NDN uses caching to improve the delivery performance of popular content. MobilityFirst improves the performance of mobile devices as they move from network to network.

For each of these proposals, part of the analysis must be whether the mechanisms related to performance need management, need a control protocol, or function as a natural consequence of the design of the data plane.

NDN In NDN, the performance is a function of how the routing protocol finds the closest copy and the cache replacement algorithm in the various routers in the system. It is possible that the cache replacement algorithm needs to be tuned based on the dominant class of content being retrieved, and this tuning may be a management function. If so, what parameters should a router report about its cache to facilitate management? If the cache uses an LRU scheme, it might make available some measure of the time that is elapsing between last use and removal.

MobilityFirst Does the Global Name Resolution Service in MobiliyFirst require management? Should the latency of the GNRS be tracked and reported?

[[[Add others?]]]

With the advent of massive content delivery over the Internet, and the use of Content Delivery Networks with complex caching schemes to improve the delivery of content, new issues related to performance have arisen that seem to call for new interfaces for the management (or control) of these schemes. CDN providers may have many copies of the same content cached at different locations around the Internet, and can select a specific source for any transfer in order to optimize the delivery. By careful management, CDN providers can operate their interconnection links essentially fully loaded without triggering actual congestion and its consequences. However, to do this they have to detect what the actual instantaneous load on the link is. Today, there is no way to extract that information through a management/control interface; they must estimate whether the link is fully loaded by looking for transient evidence of congestion. In this case, there is no business barrier to revealing the information–with directly interconnected caches the router and the CDN belong to the same firm. But the desired parameter is not defined or exported.

Like SDN, where the transition from a decentralized route computation to a centralized one triggers the need for new interfaces between the router and its control/management function, the transition from a end-to-end congestion scheme based on indirect feedback to an explicit scheme running on the CDN infrastructure will benefit from the development of new performance parameters on routers.

Security management

In Chapter 7, I broke the problem of security into four sub-objectives. Each of them will raise its own requirements for management, some of which I discussed in that chapter.

Attacks on communication With the exception of availability, I argued that this requirement should be addressed using end-to-end encryption. The major issue here is key management, which is not strictly an issue for the *network* but for the attached service nodes. However, systems such as the Certificate Authority system, while not a part of "the network", have risen to a level of importance that they are (perhaps like the DNS) trending toward being architecture. The CA system has massive issues of management, with organizations such as the CA/Browser Forum[10] meeting to discuss which root authorities are trustworthy, and so on. This situation, on the one hand, may serve to support the view that a system that requires this much management is a mis-designed system. On the other hand, key management is a known, tricky problem. However, while the problems are critical to the overall security of the Internet, they seem out of scope for network architecture as I have been defining it.

With respect to availability, the issue here are those I discussed in the context of fault management.

The process of configuring a web server to support TLS has been a manual and complex management task, which has prevented many web site operators from implementing the security protocols. A recent effort, the Let's Encrypt initiative,[11] has attempted to change to process of configuring TLS from a manual management process to an essentially automated task, requiring only a minimum of user intervention. While again, this effort seems a bit removed from *network* architecture, it illustrates that for many problems there are a range of solutions, ranging from the more manual (management) to more automatic (control) solutions.

Attacks on the host When a host is attacked by the network or by another host, the mitigation of this problem (as I conceive it) requires both end-node and network action. Proper design of applications is critical.

Some architectures, such as I3 and DOA, allow end-nodes to use the expressive power of the packet header to invoke in-network services to provide services such as protection from attack. The management issues in such a

[10] *See https://cabforum.org/*

[11] *See https://letsencrypt.org/.*

scheme remain to be fleshed out, but the complexity of having services distributed across several nodes in the network seem to suggest the potential of complex management requirements.

Attacks on the network itself The most obvious attacks on the network today (aside from DDoS, discussed below) are attacks on the interdomain routing system. Other architectures with different feature sets will, of course, manifest different opportunities for attack. The security of BGP, as it is being done today, requires a great deal of manual configuration (installation of public-private key pairs, registration of address blocks, and so on). As with the Let's Encrypt effort, there is a open question as to how automatic the configuration of secure BGP might be. However, a general point is that much of security management is the*configuration* of security parameters such as keys.

Denial of Service attacks As I discussed in Chapter 7, DoS attacks (and DDoS attacks in particular) are a problem that arises at the network level and must be managed at least to some extent at the network level. I described a range of approaches, each of which has its own requirements for new management and control interfaces. Routers that participate in traceback logging must make available that function through some interface, and the resulting security issues must be analyzed. The approach in FII involving the Shut Up Message (SUM) requires that every sending host be associated with a trusted third party that vouches for its identity, which seems to imply a significant management task. Again, different design approaches may result in schemes with very different degrees of manual intervention.

Routing

The routing protocols in the current Internet are in some respects self-configuring. When two routers each discover that there is an active node on the other end of a connected link, they begin to exchange information with the goal of discovering what is reachable through that other router. The ports on each router have to be assigned an IP address (manual configuration management), and a name (for reverse lookup) is sometimes assigned to that address, but little more is needed in general.

The emergence of new, centralized route computation schemes such as SDN require new management/control interfaces on routers and switches, as I noted above.

Quality of Experience

Quality of Experience, or QoE, is the subjective measure of the degree to which a user is satisfied with the application being used. Many factors can influence how QoE is perceived by the user: the expectation against which the experience is being assessed, whether the user paid for the application, the overall mood of the user, and so on. However, in this context, I want to focus on those aspects of QoE that are related to the character of the network being used to implement the application. In this context, QoE might fit into performance, or perhaps fault isolation. As well, it has aspects of security, if I include availability in that category. When the user encounters an impairment to QoE that is due to some phenomenon in the network, the steps to resolve the problem very much resemble those I identified to deal with issues of availability:

- It must be possible to determine that the impairment is arising in the network.
- it must be possible to localize the failed parts of the system.
- It must be possible to reconfigure the system to avoid depending on these parts.
- It must be possible to signal to some responsible party that a failure has occurred.

The issue of localization is thus central of allowing impairments to QoE to be remedied. Lacking localization, the user is reduced to waiting until some other person (presumably the person who manages the relevant entity)

notices that something is wrong and fixes it. And, as I noted above, localization of a problem to a distant region of the network may be seen as an adversarial act.

I believe that in the future, there will be an increasing focus on measurement of QoE and diagnosis of QoE impairments, which will create a generalized requirement for localization that is not restricted to "faults", but as well to performance issues, flaws in higher-level services in the network, and so on. As such, if there is a generalized approach to localization of issues in a "dumb network", the invention of such a scheme would be a major advance in network design.

10.5 Conclusions

This chapter is more speculative than some of the earlier chapters. Research on network architecture and design has provided many fewer examples of candidate mechanisms to consider, and our operational experience with the current Internet is based on a set of ad hoc mechanisms that are often based on using features in ways for which they were not intended. While I believe that I have identified a few potential network features that rise to the level of architecture, and have posed some important research challenges, it is not clear how the research community should proceed to learn more about this area. What we need is operational experience with networks at scale, but we cannot easily use the production Internet for this purpose. I fear that this area may remain underdeveloped and feeble.

Chapter 11

Meeting the needs of society

by David Clark and kc claffy

11.1 What do we want our future Internet to be?

The goal of this chapter is to identify some desirable properties of a future Internet, looking through the lens of societal concerns, and consider what (if anything) network architecture has to do with these goals.

Several years ago, my co-author kc claffy and I were moved to try to collect in one paper a list of all the societal aspirations for the future of the Internet that we could find, and organize them into categories[Clark and claffy, 2015]. For example, we collected statements from governments and public interest groups. The resulting list of aspirations was not original to us, nor did we agree with all of them. We cataloged these aspirations in order to subject them to critical analysis, and motivate a debate over which of them are desirable, well-specified, realistic and achievable.

This exercise led us to three high-level conclusions, perhaps obvious but often neglected. First, not only are many of the aspirations hard to achieve, but some are incompatible with others. Second, many are under-specified and resist operational definition; it is unclear how to translate the aspiration to concrete goals against which to measure progress. Third, most of the current tools society has to shape the future of the Internet seem unequal to the task.

These conclusions, while potentially pessimistic, raise the question of whether a different Internet might be a better vehicle for the pursuit of these goals. For this reason, we have taken our list from that paper as a starting point through which to look at this final architectural requirement: a future Internet should be designed to meet the needs of society.

In the pursuit of these goals, we encounter again what I called the *fundamental tussle*. Governments or advocacy groups express many aspirations on this list as societal goals – desirable outcomes for the citizenry, thus "in the public interest". And yet the Internet's architecture and infrastructure are now primarily under the stewardship of the private sector, driven by profitability and commercial viability, constrained by technological and economic circumstances, and sustained by interconnecting and interoperating with competitors in a multistakeholder ecosystem. Navigating the inherent tension between private sector objectives and societal aspirations is essential to shaping the future of the Internet.

11.2 Catalog of aspirations

Here is our catalog of aspirations for the future of the Internet:

1. The Internet should reach to every person by some means. (Reach)
2. The Internet should be available to us everywhere. (Ubiquity)
3. The Internet should continue to evolve to match the pace and direction of the larger IT sector. (Evolution)
4. The Internet should be used by more of the population. (Uptake)

5. Cost should not be a barrier to the use of the Internet. (Affordable)
6. The Internet should provide experiences that are sufficiently free of frustration, fears and unpleasant experiences that people are not deterred from using it. (Trustworthy)
7. The Internet should not be an effective space for law-breakers. (Lawful)
8. The Internet should not raise concerns about national security (National security)
9. The Internet should be a platform for vigorous innovation, and thus a driver of the economy. (Innovation)
10. The Internet should support a wide range of services and applications. (Generality)
11. Internet content should be accessible to all without blocking or censorship. (Unblocked)
12. The consumer should have choices in their Internet experience. (Choice)
13. The Internet should serve as a mechanism for the distribution of wealth among different sectors and countries (Redistribution)
14. The Internet (and Internet technology, whether in the public net or not) should become a united technology platform for communication. (Unification)
15. For any region of the globe, the behavior of the Internet should be consistent with and reflect its core cultural/political values. (Local values)
16. The Internet should be a tool to promote social, cultural, and political values, especially universal ones. (Universal values)
17. The Internet should be a means of communication between citizens of the world. (Global)

As we organized these aspirations, we found that many of them could be clustered into four more general categories:

- Utility
- Economics
- Security
- Openness

11.3 The utility cluster

The Internet should support a wide range of services and applications. (Generality) The original Internet architecture embedded this aspiration, since it was designed to support a cooperative network of time-shared general-purpose computers. Benefits that follow from this aspiration include *Innovation* and *Uptake*, since the more value the Internet can deliver, the more users it will attract.

Although there is no obvious way to quantify progress toward *Generality*, the range of Internet applications demonstrates its success at this aspiration. But not all applications work well on the public Internet today – most problematic are those that require very high reliability and availability, e.g., remote surgery, remote control of autonomous vehicles. Does *Generality* imply the need to evolve to support such ambitious services, or should they be segregated to more controlled private networks?

The Internet should be used by more of the population. (Uptake) *Uptake* is about getting more people to use the Internet services available to them. As more essential social services migrate to the Internet to increase the efficiency of delivering them, non-users may be increasingly disadvantaged.

This goal seems generally laudable, but invites the question as to whether policy intervention is appropriate to convert the non-users. There is less consensus on *Uptake* as a societal aspiration, compared to others, e.g., *Reach*. Respondents to Pew's 2010 survey on U.S. home broadband usage [Smith, 2010] split on the question of whether non-users were disadvantaged; the most significant concern for non-users related to finding job opportunities.

The consumer should have choices in their Internet experience. (Choice) There are many possible sorts of *Choice* in the Internet ecosystem, e.g., of broadband access providers, or of software in an app store.

Freedom of choice seems central to U.S. policy thinking, but the word "choice" is ill-defined; it is often used as a proxy for some other aspiration, for which choice is either a means or a consequence. Choice is described as a positive consequence of a competitive market. The logic is that competition leads to choice, and consumers will choose wisely, so competition disciplines providers toward offering products and services that consumers prefer.

But choice presents tension with other aspirations. Given choice, consumers might pick a network that was more regulated, curated, and/or more stable than today's Internet (e.g., Apple's app ecosystem), an outcome aligned with the *Trustworthy* aspiration, but less aligned with *Innovation* and *Generality*. Or a consumer might prefer a network that is totally free of accountability and thus rampant with piracy, which governments and rights-holders would find unacceptable and constrain by other means. Or a consumer might prefer a network that is zero cost but limits the selection of applications, e.g., Facebook Zero.

Overall, we found that this aspiration was ambiguous and subject to multiple interpretations as we attempted to reduce it to operational terms.

Architectural relevance It would seem that any architecture that defines a general-purpose platform for the creation of services would support this basket of aspirations. The more detailed questions have to do with the degree of generality (e.g., QoS features) and the range of applications. Choice at the ISP level (as opposed to the higher-level service and application layer) seems to relate to the next cluster: economics.

11.4 The economics cluster

The Internet should reach every person by some means. (Reach) The **Reach** aspiration is generally uncontentious; almost every country has some form of it. The differences relate to granularity (household or kiosk?), bandwidth (how much?), and methods to achieve it. Developed countries focus on reaching the yet unserved population, usually rural areas. In developing countries, where most of the population may not have access, the focus may be on the wireless form of *Reach*, (next on the list) i.e., *Ubiquity*. To achieve *Reach* in rural areas that lack sufficient revenue to justify private investment in infrastructure deployment, some nations have provided subsidy or tax incentives to build or maintain networks. In some cases the public sector has directly funded construction. In the United States, direct public investment has happened at multiple levels, from federal stimulus money to municipal construction of residential broadband networks.

The Internet should be available to us everywhere. (Ubiquity) The *Reach* aspiration has a corollary in the age of mobile communications – every person should have access to the Internet approximately everywhere they go, implying the integration of high-performance wireless technology into the Internet.

Cost should not be a barrier to the use of the Internet. (Affordable) This goal is a component of *Uptake*, since cost is a major barrier cited by non-users today. The phrase "cost should not be a barrier…" could be mapped to the simpler phrase "the Internet should be low-cost". However, we don't expect wine to cost as little as tap

water. Low cost might map to lower value, which might be counter-productive. Perhaps an emphasis on value would be more productive as a means to uptake.

The Internet should evolve to match the pace and direction of the larger IT sector. (Evolution) The Internet was designed to connect computers together, and this aspiration captures the idea that as computing evolves, so should the Internet. In particular, as computing gets faster and cheaper (e.g., sensors), the net should get faster, and access to it cheaper. For decades Moore's law has characterized how (IT-based) demands on broadband infrastructure change much more rapidly than other sorts of infrastructure, such as the power grid. In 2013, the forecast growth of U.S. power consumption was .9% per year [U.S. Energy Information Administration, 2013], while the forecast of Internet traffic growth was 23% per year [Cisco Systems, 2013].

National Policy statements have often had a dual character [Yochai Benkler, et al., 2012]: getting some level of broadband to everyone (*Reach*) and pushing for deployment of a next generation or broadband (*Evolution*). The U.S. FCC National Broadband Plan published in 2010 aspired to a 10-year milestone for *Reach* and *Evolution*: *"100 million U.S. homes should have affordable access to actual download speeds of at least 100 Mbps and actual upload speeds of at least 50 Mbps by 2020." [Federal Communications Commission, 2010b, p. 9]* (which admittedly now looks less impressive compared to Google Fiber's gigabit-to-the-home deployments around the country since 2011).

Architectural relevance This set of aspirations relate directly to the discussion in Chapter 9 on the incentives of the private sector to invest. Investment can improve *Reach* and *Ubiquity* and *Evolution*, but perhaps not in the proportion that society might want. All are capital-intensive activities, and thus would seem to drive up cost, which would put them in conflict with the aspiration that the Internet be *affordable*. The possible influence of architecture over these factors was discussed in Chapter 9.

The Internet should be a platform for innovation, and thus a driver of the economy. (Innovation) As a key component of the IT space, the Internet has contributed to economic growth by promoting innovation and creativity, technology development, revolutionizing logistics and service industries, among other ecosystem disruptions. One interpretation of the *Innovation* goal is that the Internet must be "open", a term used to capture many other aspirations. We believe this word is a red flag for muddy (or at least unfinished) thinking. *Open* is a word with strong positive connotations, useful as a rallying cry, but dangerously vague. We prefer to refer to more specific objectives in the ill-defined basket called "open": stability, specification (open standards), freedom from discrimination or from intellectual property restrictions. But even these aspirations are not absolute. For example, some forms of discrimination among uses of a platform can promote innovation, assuming clear and consistent rules [David Clark and kc claffy, 2014]. In fact, many traffic discrimination scenarios may benefit users, the most obvious being protecting latency-sensitive traffic from the consequences of co-mingling with other traffic.

The deeper and more vexing policy question that is poorly informed by theory or fact relates to causality: what underlying properties (e.g., *Generality*, *Uptake*, *Ubiquity*, *Evolution*, *Unblocked* or access to capital) are key drivers of Innovation?

The Internet should serve as a mechanism for the distribution of wealth among sectors and countries. (Redistribution) Thousands of independent firms combine to provide the Internet ecosystem, each typically striving to be profitable and competitive, and the flow of money is integral to its structure. Contentious arguments about redistribution of capital, either to cross-subsidize from more profitable to less profitable sectors of the ecosystem (e.g., commercial to residential, urban to rural), or from more developed to less developed countries, have long characterized telecommunication policy debates and legislation.

A recent vivid example is the ongoing tension as to whether high-volume (video) content providers (and transit providers who serve them) should contribute to the costs of the infrastructure. This tension has led to debates on

whether access providers should be able to charge content and/or transit providers for access to their customers, and more generally whether interconnection arrangements should be left to industry or regulated to more fairly allocate money flows according to who induces versus carries load on the infrastructure [Rob Frieden, 2011].

In addition to cross-subsidizing across industry sectors within one country, governments also aspire to tap into international revenue flows in the Internet ecosystem. The developing world used to bring in substantial hard-currency payments from settlement fees associated with telephone calls into their countries, a revenue source that is diminishing as communication moves onto the Internet. The global controversy about the role of the ITU in regulating international Internet interconnection reflects a motivation by many parties, including governments, to change the current norms for payment for the global flow of Internet traffic to be closer to historical telephony-based norms [Hamadoun I. Toure, 2012, Geoff Huston, 2012].

Architectural relevance: These two aspirations relate directly to the discussion in Chapter 9 on "money-routing" across the Internet. The *Innovation* aspiration is almost directly an expression of hope that the infrastructure providers will spend money so that the innovators on top of that platform can make some. Put thusly, it is not obvious why such a hope would come true. The aspiration of *Redistribution* is in some direct sense a response to the pursuit of *Innovation*; it is a call for the innovators to give some of their profits to the infrastructure providers. It is interesting that one can find this aspiration expressed in pretty direct terms by some of the actors.

Again, to the extent there are architectural implications of this set of aspirations, I have tried to address them in Chapter 9. They seem to relate to architectural modularity and what interactions among the different actors are facilitated by the expressive power of the module interfaces.

The Internet (and Internet technology) should become a unified technology platform for communication. (Unification) This aspiration is not directly relevant to society; IP network operators tend to share this aspiration as a source of cost savings, or more generally to maximize return on capital investment. As such, it may facilitate the pursuit of other aspirations discussed here. The *Unification* aspiration differs from *Generality*; the latter is about supporting a wide range of services, while *Unification* reflects the economic efficiency of discontinuing other platforms and associated investments.

Historically, telephone calls, cable television, and industrial control networks each used independent specialized legacy communications infrastructure. Today, Internet technology can provide a unified platform for any important communications application. Many ISPs today run a "fully converged" IP backbone for economic efficiency, and would resist any regulatory intervention that would cause them to separate infrastructures they have unified or plan to unify.

Note that although *Unification* reduces overall costs in some areas, it also may increase costs in others, since the unified platform must support the performance of the most demanding application in each quality of service. For example, a unified IP-based platform must be reliable enough to support critical phone service, have the capacity to carry large bundles of television channels, etc. Unification may also increase risks to *National Security*, since a less diverse infrastructure has higher potential for systemic failure [Schneier, 2010, Geer, 2007], although this fear is debated [Felton, 2004].

Architectural relevance Today, we see a two-level IP platform emerging in practice, in which ISPs build an IP platform and then run their part of the IP-based global Internet on top of this platform. Most of the architectural proposals I have discussed in this book related to the creation of a new global Internet, not the creation of a new form of unified platform. Given the current trends in industry, it would seem beneficial to have an architectural exploration of this two-level structure.

The argument above about diversity vs. monoculture in the context of systemic failure (and national security) seems a valid issue to explore from an architectural perspective.

11.5 The security cluster

Just as in my chapter on security (Chapter 7), the overarching concept of security (or an aspiration for "better security") proved too general as a starting point. The aspirations grouped here all do relate to aspects of security, but they break the problem up in slightly different ways than Chapter 7 since (in the language of that chapter), they focus on harms and not system components. This focus on harms seems to make sense in the context of aspirations.

The Internet should provide experiences that are sufficiently free of frustration, fears and unpleasant experiences that people are not deterred from using it. (Trustworthy) Most users hope, expect, or assume that their use of the Internet does not lead to their behavior and data being used against them. Users also need to be able to (but often cannot) assess the safety of a given aspect of their Internet. Today, users fear side effects of Internet use, i.e., their activities being monitored, personal information used in unwelcome ways, e.g. behavioral profiling. Users fear identity theft, loss of passwords and credentials, malware corrupting their computer, losing digital or financial assets through compromised accounts. The threats are real [Madden et al., 2012, Ehrenstein, 2012, Sullivan, 2013], and include not just crimes but violations of norms of behavior, e.g., spam or offensive postings.

The Internet should not be an effective space for law-breakers. (Lawful) An Internet ecosystem that cannot regulate illegal activities will make it less *Trustworthy* and hinder *Innovation*, impeding the role of the Internet as a *General* and *Unified* platform. Generally, crime is a drag on the economy, and a symptom of erosion of civic character. But much of today's cybercrime is international, and there is significant variation in what different countries consider illegal, as well as inconsistent and in some jurisdictions poor tools to pursue lawless behavior internationally.

The Internet should not raise concerns about national security (National security) While small-scale intrusions, crimes and attacks may alarm and deter users, a large scale attack might disable large portions of the Internet, or critical systems that run over it. There are legitimate fears that the Internet could be a vector for an attack on other critical infrastructure, such as our power or water supply.

The Center for Strategic and International Studies maintains a public list of "cyber" events with national security implications [Lewis, 2014]. A few attacks have risen to the level of national security concerns, but they are hard to categorize.

Finally, of course, specific approaches to improving security may be in conflict, such as the tension with surveillance and privacy.

Architectural relevance: I refer the reader to Chapter 7 for a discussion of the issues relating architecture and security.

The user-centered framing of the *trustworthy* aspiration brings into focus the issue of privacy, which relates to the confidentiality component of the CIA triad in communications security, but is not emphasized in Chapter 7. Privacy can either be consistent with or at odds with security, depending on the aspect of security under consideration. It is consistent with the prevention of attacks on communication, makes dealing with attacks by one host on another harder, and may be at odds with some aspects of national security. Decisions as to whether (and to what extent) an architecture should favor privacy over accountability are potentially architectural, and certainly not value free. There are proposals to re-engineer the Internet in a more *Trustworthy* direction, e.g., to ensure that every user's identity is robustly known at all times [Landwehr, 2009, Mike McConnell, 2010]; these are highly controversial [Clark and Landau, 2011].

11.6 The openness cluster

Internet content should be accessible to all without blocking or censorship. (Unblocked) This aspiration implies that ISPs and other network operators must not block access to content. It also implies that those with power to compel the blocking or removal of content (e.g. governments) should refrain from doing so. Of course, many blocking and censorship actions taken by governments and private sector actors are legally justified.

This aspiration is not equivalent to the ideal that all information be free – some commercial content may require payment for access, and some content may be illegal to transmit. Rather than describing the relationship between content producers and users, this aspiration describes the role of the Internet in connecting them.

For any region of the globe, the behavior of the Internet should be consistent with and reflect region's core cultural/political values. (Local values) Because values differ so much across the global, this aspiration arguably implies some partitioning of the global Internet, at least in terms of user experience. In the U.S., the relevant values would include First Amendment freedoms (speech, association/ assembly, religion, press, petition), but with limitations on certain types of speech and expression. Other regions prefer an Internet that safeguards social structure or regime stability. Debate about the desirability of this aspiration is a critical aspect of international policy development.

The Internet should promote universal social and political values. (Universal values) This aspiration implies the existence of universal values, such as those articulated in the United Nations' charter or the Universal Declaration of Human Rights (UDHR) [United Nations, 1948], namely peace, freedom, social progress, equal rights and human dignity [Annan, 2013]. Although such values are by no means universally accepted, we can imagine translating these values into the Internet (as Barlow passionately did back in 1996 [John Perry Barlow, 1996]) to yield aspirations such as:

- Governments should not restrict their citizens' ability to interact with people outside their borders, as long as there is no harm to others. The physical world analogue is universal human right of freedom of movement, either within the state, or outside the state with right of return, or to leave permanently [United Nations, 1948].
- People should be allowed to communicate directly with citizens of other states and they should be able to hear our message without interference from their government; this is a functional implementation of the global right to free (virtual) assembly and speech.
- The Internet should enable and enhance global interactions (as long as they are not criminal) to foster the exchange of ideas. (But since "criminal" has nation-specific definitions, this aspiration would require a liberal interpretation of acceptable interaction across the globe.)
- The Internet should serve as a forum for an international "marketplace of ideas".

Perhaps as a cyber-manifestation of American exceptionalism, the U.S. has expressed the view that the technology of cyberspace can be a means to export rather U.S.-centric values we hold as universal, i.e., to change other societies to be more like us. [1] Other nations take a more inward-facing view of what they want the Internet to do for them.

[1] Two billion people are now online, nearly a third of humankind. We hail from every corner of the world, live under every form of government, and subscribe to every system of beliefs. And increasingly, we are turning to the Internet to conduct important aspects of our lives… the freedoms of expression, assembly, and association online comprise what I've called the freedom to connect. The United States supports this freedom for people everywhere, and we have called on other nations to do the same. Because we want people to have the chance to exercise this freedom." – [Hillary Clinton, 2011]

Architectural relevance: The aspiration that citizens be able to communicate globally does not imply that *all* of the Internet experience be globally available in a consistent form, only that there is an effective basis for global communication among people, i.e., some tools for discourse and exchange. This aspiration would seem to benefit from *Generality*. The alignment of the Internet with local values has a positive and a negative aspect. The positive is the development of applications that are localized to the language and expectations of different parts of the world. Even if the Internet is potentially a platform for global communication, we should realistically expect that for most users, most of their experience will be domestic. The negative side of shaping the Internet to local values is censorship. In technical terms, censorship is an attack on a communication between willing parties, but those who carry out censorship do not describe what they do as a security violation, since they claim the right of law. However, the tools we design to protect communication from attack will blunt the tools of the censor, whether or not we have sympathy with the motives of a censor.

In the current Internet, this tussle over censorship has played out in a particular way. Rather than try to examine packet flows and block content in flight, countries have been going to the major content providers and pressuring them to block delivery at the source based on the jurisdiction of the recipient. Large providers have in many cases yielded to this pressure and are providing country-specific filtering of content and search results.

The desire for jurisdiction-specific blocking is not restricted to governments. Providers of commercial content such as music and video usually license such content for consumption on a country-specific basis. They are as anxious as any government to regulate access based on the country of the recipient.

This current state of affairs raises a specific value-laden decision for an Internet–should the design make it easy or hard to determine the country (legal jurisdiction) of a particular recipient? The Internet today supports this capability in an approximate way, since most IP addresses are assigned in a way that maps to a country. In this context, IP addresses cannot be forged, since the sender needs to get packets back.[2] Most of the actors concerned with access control have accepted this approximation as adequate. But if a new Internet were proposed, one option would be that addresses are *always* assigned on a per-country basis, which would make this approach more robust.

An alternative would be to require that some sort of "credential of citizenship" be included in requests for content. This approach seems highly problematic for a number of reasons, including the obvious evasion, which would be to borrow a credential from a person in a different country. Additionally, a country could revoke the right of a citizen to retrieve content by revoking his credential (sort of like revoking a passport, perhaps). This seems like a risky allocation of power to the state. However, architectures such as Nebula, with the requirement for a distributed control plane negotiation before initiating a data transfer, might be able to embed a certificate of jurisdiction into the Proof of Consent in a non-forgeable way.

Another alternative would be to design an architecture that escalates the tussle by making it *harder* to determine the jurisdiction of origin for a query, and see how the adversaries respond. This is what NDN does, where the interest packet carries the name of the content being sought, but not the address of the requester, thus making it impossible for the content source to determine the jurisdiction of the sender from the received interest packet.

To this date, countries have been willing to take rather drastic action, including the blocking of a whole web site as a consequence of one unacceptable piece of content hosted there. This is a space where any architectural decision will be heavily value-driven. I argued above, in the context of individual accountability, that identity at the individual level should *not* be a part of the architecture. I am less clear about an architectural binding of an internet end-point to a jurisdiction. One consideration is that there will be other sorts of credential that service providers will want from clients (such as their age group) and there is no enforceable way this can be embedded into the architecture.

Political scientists will note that avoidance of escalation is an important topic of study for those concerned with international relations. The sort of arms races we see today (with encryption, blocking of VPNs, tunnels

[2] *Of course, informed clients today are defeating this jurisdictional binding by using VPNs and other sorts of tunnels, which is causing censors to block those tools.*

and whole sites) signals that designers today are in an escalatory frame of mind when they design mechanism. Perhaps, in meeting the needs of society, we need to think about political compromise and not confrontation and escalation when we make value-laden architectural decisions.

Bibliography

[Alexander et al., 1997] Alexander, D. S., Shaw, M., Nettles, S. M., and Smith, J. M. (1997). Active bridging. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '97, pages 101–111, New York, NY, USA. ACM. 5.3

[Andersen, 2003] Andersen, D. G. (2003). Mayday: Distributed filtering for internet services. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 3–3, Berkeley, CA, USA. USENIX Association. 7.9, 7.9

[Anderson and Needham, 2004] Anderson, R. and Needham, R. (2004). Programming satan's computer. In *Computer Science Today*, pages 426–440. Springer Verlag. 4

[Anderson et al., 2005] Anderson, T., Peterson, L., Shenker, S., and Turner, J. (2005). Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41. 5.3

[Anderson et al., 2004] Anderson, T., Roscoe, T., and Wetherall, D. (2004). Preventing internet denial-of-service with capabilities. *SIGCOMM Comput. Commun. Rev.*, 34(1):39–44. 7.9

[Annan, 2013] Annan, K. (2013). Universal values - peace, freedom, social progress, equal rights, human dignity fi?! acutely needed, Secretary-General says at Tubingen University, Germany. `http://www.un.org/press/en/2003/sgsm9076.doc.htm`. 11.6

[Argyraki and Cheriton, 2004] Argyraki, K. and Cheriton, D. R. (2004). Loose source routing as a mechanism for traffic policies. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, FDNA '04, pages 57–64, New York, NY, USA. ACM. 11.6

[Balakrishnan et al., 2004] Balakrishnan, H., Lakshminarayanan, K., Ratnasamy, S., Shenker, S., Stoica, I., and Walfish, M. (2004). A layered naming architecture for the internet. *SIGCOMM Comput. Commun. Rev.*, 34(4):343–352. 5.5

[Belady and Lehman, 1976] Belady, L. A. and Lehman, M. M. (1976). A model of large program development. *IBM Systems Journal*, 15(3):225–252. 6.3

[Braden et al., 2003] Braden, R., Faber, T., and Handley, M. (2003). From protocol stack to protocol heap: Role-based architecture. *SIGCOMM Comput. Commun. Rev.*, 33(1):17–22. 4.8

[Caesar et al., 2006] Caesar, M., Condie, T., Kannan, J., Lakshminarayanan, K., and Stoica, I. (2006). Rofl: routing on flat labels. *SIGCOMM Comput. Commun. Rev.*, 36(4):363–374. 1, 5.5

[CCITT, 1992] CCITT (1992). *Management framework for Open Systems Interconnection (OSI) for CCITT applications: X.700*. International Telecommunications Union. https://www.itu.int/rec/T-REC-X.700-199209-I/en. 10.2

[Cerf and Kahn, 1974] Cerf, V. and Kahn, R. (1974). A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637–648. 11.6

[Chen et al., 2010] Chen, S., Wang, R., Wang, X., and Zhang, K. (2010). Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 191–206, Washington, DC, USA. IEEE Computer Society. 7.5

[Cheriton, 2000] Cheriton, D. (2000). Triad. *SIGOPS Oper. Syst. Rev.*, 34(2):34–. 5.1

[Cheriton, 1989] Cheriton, D. R. (1989). Sirpent: A high-performance internetworking approach. *SIGCOMM Comput. Commun. Rev.*, 19(4):158–169. 11.6

[Cheriton and Deering, 1985] Cheriton, D. R. and Deering, S. E. (1985). Host groups: A multicast extension for datagram internetworks. In *Proceedings of the Ninth Symposium on Data Communications*, SIGCOMM '85, pages 172–179, New York, NY, USA. ACM. 11.6

[Chirgwin, 2015] Chirgwin, R. (2015). Spud ? the ietf's anti-snooping protocol that will never be used. *The Register*. 7

[Cisco Systems, 2013] Cisco Systems, I. (2013). Cisco visual networking index: Forecast and methodology, 2012-2017. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`. 11.4

[Claffy and Clark, 2014] Claffy, k. and Clark, D. (2014). Platform Models for Sustainable Internet Regulation. *Journal of Information Policy*, 4:463–488. 6.7, 9.2

[Clark et al., 2003] Clark, D., Braden, R., Falk, A., and Pingali, V. (2003). Fara: Reorganizing the addressing architecture. *SIGCOMM Comput. Commun. Rev.*, 33(4):313–321. 11.6

[Clark and claffy, 2015] Clark, D. and claffy, k. (2015). An Inventory of Aspirations for the Internet's future. Technical report, Center for Applied Internet Data Analysis (CAIDA). 11.1

[Clark and Landau, 2011] Clark, D. and Landau, S. (2011). Untangling attribution. *Harvard National Security Journal*, 2. 7.6, 11.5

[Clark et al., 2004] Clark, D., Sollins, K., and Wroclawski, J. (2004). New arch: Future generation internet architecture. Available at `http://www.isi.edu/newarch/iDOCS/final.finalreport.pdf`. 5.1

[Clark, 1997] Clark, D. D. (1997). Internet economics. In McKnight, L. and Bailey, J., editors, *Internet Economics*, chapter Internet Cost Allocation and Pricing. MIT Press, Cambridge, MA. 9.3

[Clark and Blumenthal, 2011] Clark, D. D. and Blumenthal, M. S. (2011). The end-to-end argument and application design: The role of trust. *Federal Communications Law Review*, 32(2). 7.3

[Clark and Tennenhouse, 1990] Clark, D. D. and Tennenhouse, D. L. (1990). Architectural considerations for a new generation of protocols. In *Proceedings of the ACM Symposium on Communications Architectures &Amp; Protocols*, SIGCOMM '90, pages 200–208, New York, NY, USA. ACM. 5.1

[Clark and Wilson, 1987] Clark, D. D. and Wilson, D. R. (1987). A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy (SP'87)*, page 184?193. IEEE Press. 5

[Clark et al., 2005a] Clark, D. D., Wroclawski, J., Sollins, K. R., and Braden, R. (2005a). Tussle in cyberspace: Defining tomorrow's internet. *IEEE/ACM Trans. Netw.*, 13(3):462–475. 4.3

[Clark et al., 2005b] Clark, D. D., Wroclawski, J., Sollins, K. R., and Braden, R. (2005b). Tussle in cyberspace: defining tomorrow's internet. *IEEE/ACM Trans. Netw.*, 13(3):462–475. 1074049. 6.5

[Computer Systems Policy Project, 1994] Computer Systems Policy Project (1994). Perspectives on the national information infrastructure: Ensuring interoperability. 5.1

[Courcoubetis et al., 2016] Courcoubetis, C., Gyarmati, L., Laoutaris, N., Rodriguez, P., and Sdrolias, K. (2016). Negotiating premium peering prices: A quantitative model with applications. *ACM Trans. Internet Technol.*, 16(2):14:1–14:22. 9.3

[Cross-Industry Working Team, 1994] Cross-Industry Working Team (1994). An architectural framework for the national information infrastructure. Available at `http://www.xiwt.org/documents/ArchFrame.pdf`. 5.1, 9.2

[Crowcroft et al., 2003] Crowcroft, J., Hand, S., Mortier, R., Roscoe, T., and Warfield, A. (2003). Plutarch: An argument for network pluralism. *SIGCOMM Comput. Commun. Rev.*, 33(4):258–266. 5.1

[Dannewitz et al., 2013] Dannewitz, C., Kutscher, D., Ohlman, B., Farrell, S., Ahlgren, B., and Karl, H. (2013). Network of information (netinf) - an information-centric networking architecture. *Comput. Commun.*, 36(7):721–735. 5.1

[David Clark and kc claffy, 2014] David Clark and kc claffy (2014). Approaches to transparency aimed at minimizing harm and maximizing investment. `http://www.caida.org/publications/papers/2014/approaches_to_transparency_aimed/`. 11.4

[Decasper et al., 1998] Decasper, D., Dittia, Z., Parulkar, G., and Plattner, B. (1998). Router plugins: A software architecture for next generation routers. In *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '98, pages 229–240, New York, NY, USA. ACM. 5.3

[Deering, 1992] Deering, S. E. (1992). *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, Stanford, CA, USA. UMI Order No. GAX92-21608. 11.6

[Deering, 1993] Deering, S. E. (1993). Sip: Simple internet protocol. *IEEE Network*, 7(3):16–28. 11.6

[Deering and Cheriton, 1990] Deering, S. E. and Cheriton, D. R. (1990). Multicast routing in datagram internetworks and extended lans. *ACM Trans. Comput. Syst.*, 8(2):85–110. 11.6

[Dukkipati, 2008] Dukkipati, N. (2008). *Rate Control Protocol (RCP): Congestion Control to Make Flows Complete Quickly*. PhD thesis, Stanford University, Stanford, CA, USA. `http://yuba.stanford.edu/~nanditad/thesis-NanditaD.pdf`. 3, 10.3

[Ehrenstein, 2012] Ehrenstein, C. (2012). New study in germany finds fears of the internet are much higher than expected. `http://www.worldcrunch.com/tech-science/new-study-in-germany-finds-fears-of-the-internet-are-much-higher-than-ec4s4780/`, visited July 1, 2013. 11.5

[Fall, 2003] Fall, K. (2003). A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 27–34, New York, NY, USA. ACM. 5.1, 5.2

[Farber and Vittal, 1973] Farber, D. and Vittal, J. J. (1973). Extendability considerations in the design of the distributed computer system (dcs). In *Proc. Nat. Telecomm. Conf.*, Atlanta, Georgia. 11.6

[Feamster, 2006] Feamster, N. G. (2006). *Proactive Techniques for Correct and Predictable Internet Routing*. PhD thesis, MIT, Cambridge, MA, USA. 9.2

[Federal Communications Commission, 2010a] Federal Communications Commission (2010a). Connecting america: the national broadband plan. https://www.fcc.gov/general/national-broadband-plan. 9.1

[Federal Communications Commission, 2010b] Federal Communications Commission (2010b). The National Broadband Plan: Connecting America. http://download.broadband.gov/plan/national-broadband-plan.pdf. 11.4

[Felton, 2004] Felton, E. (2004). Monoculture debate: Geer vs. charney. https://freedom-to-tinker.com/blog/felten/monoculture-debate-geer-vs-charney/. 11.4

[Floyd and Jacobson, 1993] Floyd, S. and Jacobson, V. (1993). Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413. 10.3

[Ford, 2004] Ford, B. (2004). Unmanaged internet protocol: Taming the edge network management crisis. *SIGCOMM Comput. Commun. Rev.*, 34(1):93–98. 11.6

[Forgie, 1979] Forgie, J. (1979). St - a proposed internet stream protocol: Ien 119. `https://www.rfc-editor.org/ien/ien119.txt`. 11.6, 11.6

[Fortz and Thorup, 2000] Fortz, B. and Thorup, M. (2000). Internet traffic engineering by optimizing ospf weights. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 519–528 vol.2. 11.6

[Francis, 1994a] Francis, P. (1994a). *Addressing in Internet Protocols*. PhD thesis, University College London. `http://www.cs.cornell.edu/people/francis/thesis.pdf`. 11.6, 11.6, 11.6, 11.6, 11.6

[Francis, 1994b] Francis, P. (1994b). Pip near-term architecture: Rfc 1621. `https://tools.ietf.org/html/rfc1621`. 11.6, 11.6

[Fraser, 1980] Fraser, A. G. (1980). Datakit - a modular network for synchronous and asynchronous traffic. In *Proceedings of the International Conference on Communications*, Boston, MA. 11.6

[Geer, 2007] Geer, D. E. (2007). The evolution of security. *Queue*, 5(3):30–35. 11.4

[Geoff Huston, 2012] Geoff Huston (2012). It's just not Cricket: Number Misuse, WCIT and ITRs. `http://www.circleid.com/posts/number_misuse_telecommunications_regulations_and_wcit/`. 11.4

[Godfrey et al., 2009] Godfrey, P. B., Ganichev, I., Shenker, S., and Stoica, I. (2009). Pathlet routing. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 111–122, New York, NY, USA. ACM. 5.2, 7.9

[Gold et al., 2004] Gold, R., Gunningberg, P., and Tschudin, C. (2004). A virtualized link layer with support for indirection. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, FDNA '04, pages 28–34, New York, NY, USA. ACM. 11.6

[Guha et al., 2004] Guha, S., Takeda, Y., and Francis, P. (2004). Nutss: A sip-based approach to udp and tcp network connectivity. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, FDNA '04, pages 43–48, New York, NY, USA. ACM. 11.6

[Hamadoun I. Toure, 2012] Hamadoun I. Toure (2012). Remarks to ITU Staff on World Conference on International Telecommunications (WCIT-12). http://www.itu.int/en/osg/speeches/Pages/2012-06-06-2.aspx. 11.4

[Hicks et al., 1999] Hicks, M., Moore, J. T., Alexander, D. S., Gunter, C. A., and Nettles, S. M. (1999). Planet: an active internetwork. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1124–1133 vol.3. 5.2, 5.3, 10.3

[Hillary Clinton, 2011] Hillary Clinton, S. o. S. (2011). Remarks: Internet Rights and Wrongs: Choices & Challenges in a Networked World. http://www.state.gov/secretary/rm/2011/02/156619.htm. 1

[Hinden, 1994] Hinden, R. (1994). Rfc 1710: Simple internet protocol plus white paper. https://tools.ietf.org/html/rfc1710. 11.6, 11.6

[j. Wang and l. Xiao, 2009] j. Wang, X. and l. Xiao, Y. (2009). Ip traceback based on deterministic packet marking and logging. In *Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09. International Conference on*, pages 178–182. 7.9

[Jacobson, 1988] Jacobson, V. (1988). Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA. ACM. 4.9, 10.3

[John Perry Barlow, 1996] John Perry Barlow (1996). A Declaration of the Independence of Cyberspace. https://projects.eff.org/~barlow/Declaration-Final.html. 11.6

[Jon Postel, 1981] Jon Postel (1981). Service Mappings. http://www.ietf.org/rfc/rfc795.txt". 3

[Jonsson et al., 2003] Jonsson, A., Folke, M., and Ahlgren, B. (2003). The split naming/forwarding network architecture. In *First Swedish National Computer Networking Workshop (SNCNW 2003)*, Arlandastad, Sweden. 11.6

[Katabi et al., 2002] Katabi, D., Handley, M., and Rohrs, C. (2002). Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 89–102, New York, NY, USA. ACM. 3, 10.3

[Kaur et al., 2003] Kaur, H. T., Kalyanaraman, S., Weiss, A., Kanwar, S., and Gandhi, A. (2003). Bananas: An evolutionary framework for explicit and multipath routing in the internet. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, FDNA '03, pages 277–288, New York, NY, USA. ACM. 11.6

[Keromytis et al., 2002] Keromytis, A. D., Misra, V., and Rubenstein, D. (2002). Sos: Secure overlay services. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 61–72, New York, NY, USA. ACM. 7.9, 11.6

[Kim et al., 2008] Kim, C., Caesar, M., and Rexford, J. (2008). Floodless in seattle: A scalable ethernet architecture for large enterprises. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 3–14, New York, NY, USA. ACM. 5.5

[Kirschner and Gerhart, 1998] Kirschner, M. and Gerhart, J. (1998). Evolvability. *Proceedings of the National Academy of Science*, 95:8420–8427. 1.2

[Koponen et al., 2007] Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., and Stoica, I. (2007). A data-oriented (and beyond) network architecture. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, pages 181–192, New York, NY, USA. ACM. 5.1

[Koponen et al., 2011] Koponen, T., Shenker, S., Balakrishnan, H., Feamster, N., Ganichev, I., Ghodsi, A., Godfrey, P. B., McKeown, N., Parulkar, G., Raghavan, B., Rexford, J., Arianfar, S., and Kuptsov, D. (2011). Architecting for innovation. *SIGCOMM Comput. Commun. Rev.*, 41(3):24–36. 5.1

[Kushman et al., 2007] Kushman, N., Kandula, S., and Katabi, D. (2007). Can you hear me now?!: It must be bgp. *SIGCOMM Comput. Commun. Rev.*, 37(2):75–84. 8.1

[Lampson, 1973] Lampson, B. W. (1973). A note on the confinement problem. *Commun. ACM*, 16(10):613–615. 7.6

[Landwehr, 2009] Landwehr, C. E. (2009). A national goal for cyberspace: Create an open, accountable internet. *Security Privacy, IEEE*, 7(3):3 –4. http://owens.mit.edu:8888/sfx_local?__char_set=utf8&id=doi:10.1109/MSP.2009.58%7D,&sid=libx%3Amit&genre=article. 11.5

[Lewis, 2014] Lewis, J. (2014). Significant cyber events. http://csis.org/program/significant-cyber-events. 11.5

[Luderer et al., 1981] Luderer, G. W., Che, H., and Marshall, W. T. (1981). A virtual circuit switch as the basis for distributed systems. In *Proceedings of the Seventh Symposium on Data Communications*, SIGCOMM '81, pages 164–179, New York, NY, USA. ACM. 11.6

[MacKie-Mason and Varian, 1996] MacKie-Mason, J. K. and Varian, H. R. (1996). Some economics of the internet. *Networks, Infrastructure and the New Task for Regulation.* Available at http://deepblue.lib.umich.edu/handle/2027.42/50461. 9.3

[Madden et al., 2012] Madden, M., Cortesi, S., Gasser, U., Lenhart, A., and Duggan, M. (2012). Parents, teens and online privacy. http://www.pewinternet.org/Reports/2012/Teens-and-Privacy.aspx. 11.5

[Mike McConnell, 2010] Mike McConnell (2010). Mike McConnell on how to win the cyber-war we're losing. *The Washington Post.* 11.5

[Mills et al., 1991] Mills, C., Hirsh, D., and Ruth, G. (1991). Internet accounting: Background. 10.4

[Monge and Contractor, 2003] Monge, P. R. and Contractor, N. S. (2003). *Theories of Communication Networks.* Oxford University Press. 6.1

[Naous et al., 2011] Naous, J., Walfish, M., Nicolosi, A., Mazières, D., Miller, M., and Seehra, A. (2011). Verifying and enforcing network paths with icing. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 30:1–30:12, New York, NY, USA. ACM. 5.4, 11.6

[National Telecommunications and Information Administration, 1993] National Telecommunications and Information Administration (1993). The national information infrastructure: Agenda for action. http://clinton6.nara.gov/1993/09/1993-09-15-the-national-information-infrastructure-agenda-for-action.html. 5.1

[Needham, 1979] Needham, R. M. (1979). Systems aspects of the cambridge ring. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, SOSP '79, pages 82–85, New York, NY, USA. ACM. 11.6

[Neumann, 1990] Neumann, P. G. (1990). Cause of at&t network failure. *RISKS-FORUM Digest*, 9(62). 9

[Nguyen et al., 2011] Nguyen, G. T., Agarwal, R., Liu, J., Caesar, M., Godfrey, P. B., and Shenker, S. (2011). Slick packets. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '11, pages 245–256, New York, NY, USA. ACM. 11.6

[Nichols and Carpenter, 1998] Nichols, K. and Carpenter, B. (1998). Definition of differentiated services per domain behaviors and rules for their specification. 2

[Nichols and Jacobson, 2012] Nichols, K. and Jacobson, V. (2012). Controlling queue delay. *ACM Queue*, 10(5). http://dl.acm.org.libproxy.mit.edu/citation.cfm?id=1368746&CFID=673468559&CFTOKEN=90975803. 10.3

[Nygren et al., 1999] Nygren, E. L., Garland, S. J., and Kaashoek, M. F. (1999). Pan: a high-performance active network node supporting multiple mobile code systems. In *Open Architectures and Network Programming Proceedings, 1999. OPENARCH '99. 1999 IEEE Second Conference on*, pages 78–89. 5.2

[Open Interconnect Consortium, 2010] Open Interconnect Consortium (2010). Internet gateway device (igd) v 2.0. 5

[Parno et al., 2007] Parno, B., Wendlandt, D., Shi, E., Perrig, A., Maggs, B., and Hu, Y.-C. (2007). Portcullis: Protecting connection setup from denial-of-capability attacks. *SIGCOMM Comput. Commun. Rev.*, 37(4):289–300. 7.9, 11.6

[Postel, 1981] Postel, J. (1981). Internet protocol, network working group request for comments 791. http://www.ietf.org/rfc/rfc791.txt. 3

[Pujol et al., 2005] Pujol, J., Schmid, S., Eggert, L., Brunner, M., and Quittek, J. (2005). Scalability analysis of the turfnet naming and routing architecture. In *Proceedings of the 1st ACM Workshop on Dynamic Interconnection of Networks*, DIN '05, pages 28–32, New York, NY, USA. ACM. 11.6

[Raghavan et al., 2009] Raghavan, B., Verkaik, P., and Snoeren, A. C. (2009). Secure and policy-compliant source routing. *IEEE/ACM Transactions on Networking*, 17(3):764–777. 11.6

[Rob Frieden, 2011] Rob Frieden (2011). Rationales For and Against FCC Involvement in Resolving Internet Service Provider Interconnection Disputes. Telecommunications Policy Research Conference, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1838655. 11.4

[Rosen, 1982] Rosen, E. (1982). Exterior gateway protocol (egp). 4

[Saltzer, 1982] Saltzer, J. (1982). On the naming and binding of network destinations. In et al., P. R., editor, *Local Computer Networks*, pages 311–317. North-Holland Publishing Company. Reprinted as RFC 1498. 11.6

[Saltzer et al., 1980] Saltzer, J. H., Reed, D. P., and Clark, D. D. (1980). Source routing for campus-wide internet transport. In West, A. and Janson, P., editors, *Local Networks for Computer Communications*. http://groups.csail.mit.edu/ana/Publications/PubPDFs/SourceRouting.html. 11.6

[Saltzer et al., 1984] Saltzer, J. H., Reed, D. P., and Clark, D. D. (1984). End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288. 357402. 6.8

[Savage et al., 2000] Savage, S., Wetherall, D., Karlin, A., and Anderson, T. (2000). Practical network support for ip traceback. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, pages 295–306, New York, NY, USA. ACM. 7.9

[Schneier, 2010] Schneier, B. (2010). The dangers of a software monoculture. https://www.schneier.com/essays/archives/2010/11/the_dangers_of_a_sof.html. 11.4

[Schwartz et al., 1999] Schwartz, B., Jackson, A. W., Strayer, W. T., Zhou, W., Rockwell, R. D., and Partridge, C. (1999). Smart packets for active networks. In *Open Architectures and Network Programming Proceedings, 1999. OPENARCH '99. 1999 IEEE Second Conference on*, pages 90–97. 5.3

[Shoch, 1978] Shoch, J. F. (1978). Inter-network naming, addressing, and routing. In *IEEE Proc. COMPCON Fall 1978.* Also in Thurber, K. (ed.), Tutorial: Distributed Processor Communication Architecture, IEEE Publ. EHO 152-9, 1979, pp. 280-287. 11.6

[Smith, 2010] Smith, A. (2010). Home Broadband 2010. http://www.pewinternet.org/Reports/2010/Home-Broadband-2010.aspx. 11.3

[Snoeren et al., 2002] Snoeren, A. C., Partridge, C., Sanchez, L. A., Jones, C. E., Tchakountio, F., Schwartz, B., Kent, S. T., and Strayer, W. T. (2002). Single-packet ip traceback. *IEEE/ACM Transactions on Networking*, 10(6):721–734. 7.9

[Song and Perrig, 2001] Song, D. X. and Perrig, A. (2001). Advanced and authenticated marking schemes for ip traceback. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 878–886 vol.2. 7.9

[Stoica et al., 2004] Stoica, I., Adkins, D., Zhuang, S., Shenker, S., and Surana, S. (2004). Internet indirection infrastructure. *IEEE/ACM Trans. Netw.*, 12(2):205–218. 5.1, 11.6

[Sullivan, 2013] Sullivan, B. (2013). Online privacy fears are real. http://www.nbcnews.com/id/3078835/t/online-privacy-fears-are-real, visited July 1, 2013. 11.5

[Sunshine, 1977] Sunshine, C. A. (1977). Source routing in computer networks. *SIGCOMM Comput. Commun. Rev.*, 7(1):29–33. 11.6

[Tennenhouse and Wetherall, 1996] Tennenhouse, D. L. and Wetherall, D. J. (1996). Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, 26(2):5–17. 4.8, 5.2, 5.3

[Trossen and Parisis, 2012] Trossen, D. and Parisis, G. (2012). Designing and realizing an information-centric internet. *IEEE Communications Magazine*, 50(7):60–67. 5.1

[Trossen et al., 2008] Trossen, D., Tuononen, J., Xylomenos, G., Sarela, M., Zahemszky, A., Nikander, P., and Rinta-aho, T. (2008). From design for tussle to tussle networking: Psirp vision and use cases. http://www.psirp.org/files/Deliverables/PSIRP-TR08-0001_Vision.pdf. 5.1

[Turányi et al., 2003] Turányi, Z., Valkó, A., and Campbell, A. T. (2003). 4+4: An architecture for evolving the internet address space back toward transparency. *SIGCOMM Comput. Commun. Rev.*, 33(5):43–54. 11.6

[United Nations, 1948] United Nations (1948). The Universal Declaration of Human Rights. http://www.un.org/en/documents/udhr/index.shtml. 11.6

[U.S. Energy Information Administration, 2013] U.S. Energy Information Administration (2013). Annual energy outlook 2013. http://www.eia.gov/forecasts/aeo/MT_electric.cfm. 11.4

[van der Merwe et al., 1998] van der Merwe, J. E., Rooney, S., Leslie, L., and Crosby, S. (1998). The tempest-a practical framework for network programmability. *IEEE Network*, 12(3):20–28. 5.3

[Walfish et al., 2004] Walfish, M., Stribling, J., Krohn, M., Balakrishnan, H., Morris, R., and Shenker, S. (2004). Middleboxes no longer considered harmful. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, pages 15–15, Berkeley, CA, USA. USENIX Association. 5.1

[Wetherall, 1999] Wetherall, D. (1999). Active network vision and reality: Lessons from a capsule-based system. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, SOSP '99, pages 64–79, New York, NY, USA. ACM. 5.1

[Wilkes and Wheeler, 1979] Wilkes, M. V. and Wheeler, D. J. (1979). The cambridge digital communication ring. In *Local area communication networks symposium*, Boston, MA. Mitre Corporation and the National Bureau of Standards. 11.6

[Wing et al., 2013] Wing, D., Cheshire, S., Boucadair, M., Penno, R., and Selkirk, P. (2013). Port control protocol (pcp), rfc 6887. 5

[Wright et al., 2008] Wright, C. V., Ballard, L., Coull, S. E., Monrose, F., and Masson, G. M. (2008). Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 35–49, Washington, DC, USA. IEEE Computer Society. 7.5

[Wroclawski, 1997] Wroclawski, J. (1997). The Metanet. In *Workshop on Research Directions for the Next Generation Internet*, Vienna, VA, US. Computing Research Association, Computing Research Association. 5.1

[Yaar et al., 2003] Yaar, A., Perrig, A., and Song, D. (2003). Pi: a path identification mechanism to defend against ddos attacks. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 93–107. 7.9

[Yaar et al., 2004] Yaar, A., Perrig, A., and Song, D. (2004). Siff: a stateless internet flow filter to mitigate ddos flooding attacks. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 130–143. 7.9, 11.6

[Yang, 2003] Yang, X. (2003). Nira: A new internet routing architecture. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, FDNA '03, pages 301–312, New York, NY, USA. ACM. 8.3, 11.6

[Yang et al., 2005] Yang, X., Wetherall, D., and Anderson, T. (2005). A dos-limiting network architecture. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, pages 241–252, New York, NY, USA. ACM. 7.9, 7.9, 11.6

[Yemini and da Silva, 1996] Yemini, Y. and da Silva, S. (1996). Towards programmablenetworks. 5.3

[Yochai Benkler, et al., 2012] Yochai Benkler, et al. (2012). Next Generation Connectivity: A review of broadband Internet transitions and policy from around the world. `http://www.fcc.gov/stage/pdf/Berkman_Center_Broadband_Study_13Oct09.pdf`. 11.4

A history of Internet addressing

Introduction

In Chapter 5 I discussed a number of alternative proposals for an Internet architecture. However, that chapter by no means discussed all the alternative schemes that have been published. Since the Internet was first designed, there have been any number of proposals to improve or redesign it, and one of the most popular topics of study has been addressing. Since the core function of a network is to forward traffic, and the forwarding is driven by addressing (something that is in the packet), most proposals for an alternative architecture center on addressing. (I did note a few proposals in Chapter 5, including FII and Plutarch, which left addressing as a regional problem with conversion at the region boundary.) This appendix offers a review of the literature on alternative addressing schemes, and attempts to put these proposals into an organizing framework.

Defining some terms–mechanisms for forwarding

The first question is: what is an address? A operational starting point is that an address is that piece of data in the packet that allows the packet to be delivered. Packets flow across the network from source to destination, passing through a series of routers. The address is the part of the packet that the router takes as input to the forwarding decision.

This very operational definition masks a history of philosophical discussion on the difference between a name, which provides some form of identity, an address, which provides some sort of guidance to location, and a route or path, which describes the series of routers through which the packet should pass to reach that location. The interested reader is referred to the classic papers in this area: [Shoch, 1978, Saltzer, 1982], and a slightly later discussion by[Francis, 1994a]. In this appendix, I will continue with an operational definition of address.

This appendix discusses addressing and forwarding, and only peripherally routing. While most architectural proposals define an addressing and forwarding scheme, most leave the development of a routing scheme to a later stage where the architecture is fleshed out to become a complete system. Routing (as opposed to forwarding) is the process of computing the path to destinations. The routing computation produces forwarding information, which is then used to direct each packet on its way to the destination. In the case of the Internet, the routing computation runs in each of the routers and produces a forwarding table; when the packet arrives, the destination address from the packet is looked up in the forwarding table to determine what the proper action is.

With this information as background, here is a quick review of some general classes of addressing and forwarding schemes.

Destination-based forwarding: In the basic Internet architecture, there is a (sort of) global, (usually) distributed routing algorithm that runs in the background and computes, for each router, the correct next hop for every set of destination addresses. When a packet arrives at a router, the forwarding algorithm searches in the forwarding table for an entry that matches that destination address, and extracts from that entry the stored information about the next hop the packet must take. Within this general framework, different addressing schemes may lead to more or less efficient routing and forwarding algorithms. By way of example, a "flat" address space,

as in DONA, requires that the routing algorithm keep separate track of every address, while an address scheme where the structure of the address matches the topology of the network (provider-based addressing), routing need only keep track of groups of addresses that map to different parts of the network. For a very detailed analysis of different forms of address, including flat, hierarchical, and many others, see citefrancis.

Source routing: This alternative has the forwarding information in the packet, rather than in the router. In general terms, the packet lists the desired sequence of routers through which the packet should flow. Each router in turn removes its address from the list and then sends the packet onward to the router at the next address. Of course, this over-simplified description begs many questions, such as how the sender gets the source route in the first place. Source routing was proposed and discussed by [Farber and Vittal, 1973, Sunshine, 1977, Saltzer et al., 1980], and the definition of IP includes source route options, but the idea did not survive as an operational capability. None the less, variants on source routing form the basis of many alternative proposals for forwarding. In Chapter 5 I mentioned Nebula and Scion as using source addresses, and I will discuss more in this appendix. The reasons for the practical failure of source routing (so far) make an interesting case study of the set of requirements that addressing must meet.

Label switching: Destination-based forwarding, as I described it above, has a potentially costly step: the search of the forwarding table to find the entry that matches the destination address in the packet. An alternative that avoids the searching is to configure each router along the path to a destination so that the forwarding table contains an indication of the outgoing link on which the packet should be sent, and the index (usually called the label) into the forwarding table in the next switch. At each switch, the incoming label is used to look up the correct forwarding entry, and is then rewritten with the correct outgoing label. This process repeats at each switch along the pre-established path. This is the concept of label rewriting or label switching or label swapping.

Given that the data in the packet is sometimes a destination address, sometimes a sequence of addresses, and sometimes a label (and sometimes other things), the term "address" may not be the best term for that information. The term *locator* has been used as an alternative, to capture a somewhat more general idea–that thing that is in the packet. NewArch used the term *forwarding directive* in this way.

The organization of this chapter

This appendix starts with a quick review of addressing in the Internet, since this is a system that many people understand. It then looks at an alternative tradition, which is addressing in virtual circuit networks. With this background, it then catalogs a list of objectives that an addressing and forwarding scheme should meet, and then describes a number of alternative proposals using this set of objectives.

A history of Internet addressing

Before the Internet, there was the ARPAnet, of course. The ARPAnet was the first packet technology deployed by DARPA, and the first technology used to carry Internet packets when they were invented. The addressing/forwarding mechanism was a very simple form of destination-based address. In the first version each packet carried an 8 bit switch number (the so-called Interface Message Processor, or IMP number) and a 2 bit host number (so there could be 4 hosts per IMP). This was seen as too limiting and was changed to 16 bits to select the IMP, and 8 bits to select the host. Near the end of life for the ARPAnet, it was augmented with logical addressing (see RFC 878),[3] a flat, 16 bit field (two of which are flags), so 2**14 hosts could be addressed, without the sender having to know where the receiving host is. (When a host attached to the ARPAnet, it notified the IMP about what logical names it was going to use, and this information was propagated around the net, an early form of forwarding on a flat address space.)

[3] *In this appendix, I refer to a rather large number of Internet RFCs. I have not bothered to list them all in the citations. It is easy enough to look them up for those readers that want to learn more.*

The Internet's current 32 bit address was not the first option considered in the early design. The initial paper on TCP [Cerf and Kahn, 1974] proposed an 8 bit network field and a 16 bit host field (called the TCP field), with the comment: "The choice for network identification (8 bits) allows up to 256 distinct networks. This size seems sufficient for the foreseeable future. Similarly, the TCP identifier field permits up to 65 536 distinct TCP's to be addressed, which seems more than sufficient for any given network." However, in the early design discussions, the option of a variable length address field was considered, in order to allow for growth. Regrettably, this idea was rejected because of the overhead of processing variable length header fields, and the initial design settled on a 32 bit fixed field. Paul Francis, in an Epilogue to his PhD thesis [Francis, 1994a] provided a very thoughtful review of the early Internet design literature, and points out that we almost had a source-routing scheme and variable length addresses.

During the 1980's, there were a series of proposals to deal with growth. The simple model of "8 bits for the network number, and 24 bits for the rest" [RFC 760, in 1980], was replaced by the class structure, with class A addresses having 8 bits for the network, class B having 16 bits and Class c having 24 bits [RFC 791, in 1981] . Even this was seen as inadequate, and in January, 1991, the Internet Activities Board (the IAB) held a retreat, the results of which are documented in RFC 1287, which contains the following statements:

> *This [addressing and routing] is the most urgent architectural problem, as it is directly involved in the ability of the Internet to continue to grow successfully.*
>
> *The Internet architecture needs to be able to scale to 10**9 networks.*
>
> *We should not plan a series of "small" changes to the architecture. We should embark now on a plan that will take us past the exhaustion of the address space. This is a more long-range act of planning than the Internet community has undertaken recently, but the problems of migration will require a long lead time, and it is hard to see an effective way of dealing with some of the more immediate problems, such as class B exhaustion, in a way that does not by itself take a long time. So, once we embark on a plan of change, it should take us all the way to replacing the current 32-bit global address space.*
>
> *There will be a need for more than one route from a source to a destination, to permit variation in TOS and policy conformance. This need will be driven both by new applications and by diverse transit services. The source, or an agent acting for the source, must control the selection of the route options.*

The workshop that produced this report might be seen as the starting point for the search for IPng, which eventually led to IPv6.

Two short-term ideas emerged at that time to "bridge the gap" to IPng . One was CIDR, or classless addressing, which is described in a series of RFCs published around 1993. The other approach to preservation of IP addresses was to let large enterprises with many "private" hosts use private address spaces to address these machines, rather than globally routed addresses. The IESG commissioned a subgroup, called the ROAD group, whose deliberations are documented in RFC 1380. In November 1992. They wrote:

> *The following general approaches have been suggested for dealing with the possible exhaustion of the IP address space:*
>
> *...*
>
> *Addresses which are not globally unique. Several proposed schemes have emerged whereby a host's domain name is globally unique, but its IP address would be unique only within it's local routing domain. These schemes usually involve address translating.*

This idea is the starting point for the idea of Network Address Translation (NAT) devices, introduced by Paul Francis in 1993 . Private address spaces are further documented in RFC 1597, in March 1994.

Multicast

The major intellectual advance in addressing during the 1980's was the specification of IP multicast by Steve Deering. Deering's PhD thesis did not appear until 1991 [Deering, 1992], but the initial proposal emerged earlier, in [Cheriton and Deering, 1985, Deering and Cheriton, 1990]. In multicast, the destination address is not the location of the destination, but a handle or pointer, called the multicast ID. At each router, this handle is used to look up a list of next hops that the packet should take, as it fans out to all the destinations. This means that every router on the path from the source to (all of) the destinations must have the proper state information to map the handle to the destinations. This led to a large research agenda in the design of multicast routing protocols.

Multicast is important for two reasons. First, of course, it is a proposal for a new class of delivery semantics. But it also demonstrated three more general points: the locator need not be a literal address of a destination but an arbitrary bit-sequence, different regions of an address space can be associated with different forwarding mechanisms, and a router can run more than one routing algorithm at the same time. All of these generalizations are important.

While Deering is generally given credit for establishing the concept of multicast in the Internet architecture, there are much earlier mentions of the idea. As a part of the Stream protocol ST [Forgie, 1979], the concepts of conference connections, multi-addressing and special forwarders called "replicators" are introduced. So the idea of delivery to multiple destinations has been present in the Internet research community even before the standardization of IP.

So in the early part of the 1990's, there is one significant proposal (multicast) to enhance the delivery semantics of the original Internet, there is the launch of a long term effort to replace IP with a new protocol with a larger address field, there is the quick deployment of CIDR as a stopgap, and there is a major deviation being proposed to the original addressing architecture, address translation, which is also put forward as a short-term fix, but which has profound long-term implications.

The quest for IPng

The call for proposals to replace IPv4 produced two initial contributions. One was PIP [Francis, 1994b], which represented a significant shift from IP in that it used source routing as its basic forwarding mechanism. The names for the nodes that were the intermediate points in the source route were not globally known and routed addresses, but more compact locators that were only unique within a region of the network. To realize this scheme, the nodes of the Internet were organized into a hierarchy, and node names were positioned within a place in the hierarchy. The other was SIP [Deering, 1993],[4] a more conservative design that included the concept of source routes, but used globally routed names for the intermediate nodes. After some discussion within the IPng community, a compromise was devised called SIPP (SIP plus), which used the syntax of SIP but incorporated some of the advanced semantics of PIP. SIPP is described in [Hinden, 1994], and a very detailed comparison of these alternatives can be found in [Francis, 1994a].

The size of the address field was, of course, a major motivation for the replacement of IPv4. SIP and SIPP used an address field of 64 bits, twice IPv4. The final version of IPv6 further increased this to 128 bits, not because 2*64 was too small to address all the nodes we might someday see, but to facilitate address space management.

A parallel universe–virtual circuit networks

One of the key design principles of the ARPAnet was that it was "connection oriented": the ARPAnet established preset paths between all pairs of hosts, so that every IMP had per-connection state for every possible path from source host to destination host. This state was uses to manage resource allocation and congestion. This connection-oriented design philosophy contrasted with the connectionless,or "datagram" approach of the Internet. This split in approach defined two different lines of evolution for packet switched data networks.

X.25 is a connection-oriented outgrowth of a number of early projects, including ARPAnet and the early work

[4] *This acronym has nothing to do with Session Initiation Protocol, which came along later and reused the TLA.*

in England on packet networks. The design of X.25 included the concept of virtual circuits between sender and receiver. Before sending data, a user of an X.25 network used a signaling protocol to request a circuit, and received back (if the setup was successful) a short identifier for this circuit that could be used to tag each packet. The full address used in an X.25 packet thus only showed up in the setup protocol. The full X.25 address somewhat resembled a phone number: it was a sequence of ASCII digits: 3 for country, 1 for network within the country, and 10 for the end-point within the country. The country-based assignment reflects the telephony roots of X.25. It is also important that X.25 is an interface protocol that describes how a host (a piece of Data Terminal Equipment or DTE) talked to its attachment point in the network (the Data Communication Equipment, or DCE). The X.25 specification did not describe how the switches inside the network talked to each other. There were some interesting demonstrations, such as the use of X.25 as an interface to a datagram network, but there was also a need for vendor standards to design interoperable switches. In most X.25 networks, the representation of an address inside the network was the same as the representation across the interface–a short identifier for the circuit.

X.25 contained some very rich mechanisms for error detection and recovery. Much of the state established inside the switches had to do with the detection and retransmission of corrupted packets. This function seemed important in the beginning when circuits were very noisy. However, as reliability and performance improved, there was a movement toward a design that traded the occasional loss for a simplification of the switch, and this produced Frame Relay. Frame Relay forwarding and addressing was similar in design to X.25. Virtual circuits in Frame Relay networks were identified by a ten bit Data Link Connection Identifier (DLCI). This very compact coding of the virtual circuit cannot work if DLCIs had global meeting–there are not enough values. The DLCI had only local meaning, between each switch and the next. So the DLCI functions as a label, and Frame Relay is an example of label-switching as the forwarding mechanism.

Label switching and cells

The Internet uses the packet as the unit of statistical multiplexing, which is a much finer-grained unit of sharing than the phone system, where the statistical nature of the sharing shows up at call-setup time. But it is important to remember that the digital telephone system can make a separate forwarding action for each byte, which is a much finer-grained unit than the packet. This is a fixed, time-domain sharing, not a statistical sharing, and there is no "header processing" at this level–there is no header. Bytes from a given call are always in the same location in the same frame, and the forwarding action consists of reading the byte from a known position in a receive frame and writing it into a known location in a transmit frame. One of the major motivations for this design is reduction of latency and jitter, since the quality of a phone call is known to deteriorate as the end-to-end latency goes up. Given this history and this requirement for minimizing total latency, multiplexing at the packet level was thought to introduce an unacceptable level of statistical uncertainty about jitter, because of the variable size of packets that have to be interleaved. So as the designers of the telephone system considered replacing their circuit-switched architecture with a scheme more like packet switching, an alternative multiplexing model was put forth, in which the unit of multiplexing was not a variable size packet but a small, fixed size cell.

There is no fundamental reason why a cell could not carry a globally routed destination address, but considerations of header size and resulting overhead on the link suggest that the header has to be very compact if the cell size is small. This leads to the preference for pre-establishment of virtual circuit state in the switch and the "address" in each cell becomes a simple index into a table rather than requiring a search for a matching destination address.

Perhaps the most extreme form of cell switching is found the Cambridge Ring, developed at the Computer Laboratory at the University of Cambridge in the late 1970'fis, about the same time that Ethernet was being developed at Xerox Parc. The unit of transfer across the Cambridge Ring (which they initially called a packet in [Wilkes and Wheeler, 1979], but then more suggestively call a mini-packet in[Needham, 1979]), has a two byte payload, and a source and destination address of one byte each. Needless to say, the forwarding decision was of necessity rather simple, but it is also clear that one could not contemplate the overhead of putting an IP header on

a two byte payload. This system did not have label rewriting, however, since the number of hosts was so small that it was acceptable to have a global id for each.

A less extreme and more relevant example of cell switching is the Datakit architecture, developed at ATT Bell Labs [Fraser, 1980, Luderer et al., 1981]. The Datakit cell had a 16 byte payload, and a two byte header, with one byte to identify the link on the switch and one byte to identify the virtual circuit on the link. These bytes were rewritten at each switch.

The most mature form of cell switching, which to a considerable extent descends from Datakit, is the idea of Asynchronous Transfer Mode, or ATM. There is an extensive literature on ATM, which I do not even attempt to catalog here. But the core idea is similar to Datakit. The ATM cell contains a 48 byte payload, and a somewhat complex Virtual Circuit Identifier (VCI) in the header. The scheme depends on virtual circuit setup, and the VCI is a label that is rewritten at each hop.

There are actually two motivations tangled up in the design of cell switching (just as there are multiple motivations behind the design of the Internet packet). One is the switching efficiency and jitter control of fixed size cells. The other is a preference for virtual circuit setup and per-flow state in the switch. The Internet design took an extreme view that there was never any per-flow setup, and no per-flow state in the packet. This was seen as reducing the overhead and complexity of sending data–there is no required setup phase; to send a packet you "just send it". But this meant that the network was making no service commitment to the sender. There is no idea of a "call", no commitment of resources to the call, and no effort to provide different quality of service to different flows. Indeed, the Internet community spent much of the 1990s figuring out how to add QoS to the Internet, while this was always a central tenet of the virtual circuit community. So the per-flow state in the router should not just be seen as a necessary consequence of the small cell size and the need for a small header, but as a virtue in its own right. In this respect, the Internet and the "cell switching/virtual circuit" worlds are distinguished as much by their views on circuit setup and per-flow state as they are about fixed vs. variable size multiplexing units.

Label switching meets Internet 1: the Stream protocol ST

From essentially the beginning of the Internet, there was an effort to define an alternate forwarding scheme that set up a flow and had per-flow forwarding state in the router, resulting in the Stream protocol ST, first documented in IEN 119 [Forgie, 1979]. ST is concerned with providing explicit QoS for speech packets, and discusses in some detail the concept of a Flow Spec, and the need to set up state in the routers using an explicit setup protocol. ST is also concerned with multicast. ST takes advantage of the state to use a small packet header, in which the destination address has been replaced by a connection identifier, or CID, which is only locally meaningful between routers and is rewritten on each hop. This mechanism is thus an example of label switching, perhaps the first in the Internet. The ST protocol evolved over the next decade, and a new protocol called ST-2 was described in RFC 1190 in 1990. This version of ST was still based on a local label, now called the hop identifier, or HID, a 16 bit field. The specification in RFC 1190 contains details on the process of setting up a sequence of HIDs that are unique across each link. Interestingly, in the final specification of ST-2, in RFC 1819 in 1996, the HID is replaced with a stream ID, or SID, which is globally unique (a 16 bit nonce combined with the 32 bit source address), which implies a slightly more complex lookup process. RFC 1819 says: "HIDs added much complexity to the protocol and was found to be a major impediment to interoperability". So the final version of ST abandons the idea of label switching, but still depends on a full, per-flow connection setup and state in the packet.

Label switching meets Internet 2: remove the cells

As the previous discussion suggested, there are actually a bundle of motivations behind the label switching approach–the need for low-overhead headers on cells, and the desire for flow setup. There were both packet-based (Frame Relay) and cell-based (ATM) connection-oriented networks, using label switching as the basis of forwarding. In the early networks, the idea of flow setup was that a virtual circuit was equivalent to an end-to-end flow, but it became clear that another use for a virtual circuit was to set up state (and perhaps to allocate resources) to a path that carries aggregated traffic from a set of sources to a set of destinations, for example city-pairs in a large

national network. This sort of undertaking is often called traffic engineering: the allocation of traffic aggregates to physical circuits in such a way that the overall link loads are balanced, there is spare capacity for outages, and so on. The goal of traffic engineering, together with the goal of simplifying the forwarding process, led to the proposal to use label switching on variable size packets, rather than cells. Cisco Systems put forward a proposal called Tag Switching, building to some extent both on the ideas of Frame Relay and on ATM. This proposal was turned over to the IETF for standardization, where it was first called Label Switching, and then, in its full glory, Multi-Protocol Label Switching. Again, there is a wealth of literature on MPLS, including complete books. Since MPLS is a recent innovation of great current interest to the community, a brief tutorial can be found in that new compendium of all knowledge, Wikipedia.

MPLS, like many mature ideas, has come down with complexity. It supports the idea of nested flows–that is, a set of virtual circuits carried inside another. So the header of a packet can have a series of labels, not just one. When a packet reaches the beginning of a MPLS path, a label is added, when it passes along the path, the label is rewritten at each node, and then it reaches the end of the path, the label is "popped", which may reveal yet another label, or may leave no further labels to process, in which case the packet is processed using the native packet header, for example the traditional IP header.

A MPLS header is 32 bits, which is a very efficient representation of forwarding state. It consists of a 20 bit label along with some other control information.

Label switching meets Internet 3: the loss of the global address space

The essence of the original Internet forwarding scheme was the existence of global addresses, which could be used as a basis of a search in a forwarding table of any router anywhere in the Internet. But in the early 1990's, the idea of Network Address Translation boxes was introduced, which was on the one hand a very clever way to conserve scarce Internet addresses, and on the other hand, a total violation of the global addressing assumption. The "trick" that makes NAT boxes work is label switching, except that in this case the "labels" that are being rewritten are the IP addresses themselves. The IP address field in the header, which was previously a static and immutable field in the packet, is now rewritten inside the NAT box using "state in the router". This begs the question of where that state comes from, and what limitations this scheme implies. There has been a great deal of work to try to patch up the rift in the Internet architecture created by NAT, and the necessity of establishing and maintaining the right state in the NAT box, given that the Internet lacks any sort of signaling protocol. (Of course, as we will see, many of the solutions somewhat resemble a signaling protocol, though most would not be so bold as to call them "circuit setup protocols".)

The first idea for NAT was simple. When a host behind a NAT box sends a packet out, the NAT box rewrites the source address of the packet with its own source address, and remembers the internal IP address and port number of the packet. If an incoming packet arrives for that port number, it uses that remembered state to rewrite the destination address of this incoming packet with the correct local address. (Some NAT boxes also do port remapping.) In other words, the outgoing packet triggers the setup of state for the subsequent incoming packet.

This idea is fine as far as it goes, but what about an incoming packet without a prior outgoing packet–what about a server behind a NAT box? The current solution for most "consumer grade" NAT boxes is primitive. The user manually configures static state in the NAT box to allow the remapping of the incoming packet. But there are a variety of more complex solutions to set this state up dynamically.

One approach is to set the state up using a message from the machine behind the NAT box–the machine that is acting as the server. There has been a lot of work in the IETF on this: for example see RFC 3303 and the related work on middleboxes.

A more complex scheme is found in IPNL (Francis and Gummadi 2001), which provide a discussion of the pros and cons of multiple address spaces, and (in Internet terms) NAT. They list expansion of the address space, changing providers without renumbering, and multi-homing, They propose an architecture that attempts to reproduce the existing Internet functionality: all hosts have long-lived globally routed addresses if they choose, routers (including the elements that link the address spaces) are as stateless as today, and only a router on the

path of the packet can disrupt the forwarding process. Further, the scheme allows a region with a private address space to have its public address reassigned at will without disrupting the flow.

Their approach involves a source-routing *shim* header, a new layer put between IP and the transport protocol. The scheme uses fully-qualified Internet Domain name (FQDN) in the first packet of a flow as a way of deriving addressing information at various points along the path to the destination. This addressing information is gathered in the packet (not the router–this is a stateless source-routing scheme). On subsequent packets, these lower level "IP-style" addresses are used in the packet, in a tailored three-stage source route. The scheme supports failure and rerouting to alternative entry points into the private address spaces.

In IPNL, the FQDN is managed so that it is globally meaningful. The scheme does depend on having some global space of names, even if those names do not directly express location and required interaction with servers to resolve. The extreme point in this space would be a system in which there are no shared names of any sort, either locators or higher-level names. Examples of such a systems include Sirpent, discussed below, and Plutarch,discussed in Chapter‘5.

A more recent scheme for dealing with NAT is NUTSS [Guha et al., 2004], which uses Session Initiation Protocol (SIP) to set up rewriting state (in NAT boxes), so the setup is associated with a per-flow/per application signaling phase. So this scheme uses a application-level signaling protocol (SIP) to set up forwarding state in NAT routers. It is stateful, as opposed to the stateless character of IPNL.

A simpler form of source routing to deal with NAT is 4+4 [Turányi et al., 2003], a scheme that uses a two-stage source route again made up of traditional IP addresses. The 4+4 scheme uses the DNS in a different manner than IPNL. In IPNL different parts of the address are obtained as the packet crosses different addressing regions (that it, the DNS returns different values in different regions), whereas in 4+4, the DNS stores the whole two-part address. The sender looks it up and puts it on the packet at the source. This has implications for what information is visible where, and what information can change dynamically. (In IPNL, public addresses have no meaning in private address regions, in 4+4, they are meaningful and routed.)

Comparing mechanisms

Earlier, forwarding schemes were divided into two broad camps: *state in the router*, where the packet carries a simple globally meaningful locator and the router has the forwarding information, and *state in the packet*, where the packet carries a series of forwarding instructions that are carried out at each node, using forwarding information that may be very simple and locally defined. This latter scheme, as I have discussed, is called source routing. As we have seen, there are two relevant forms of *state in the router*: forwarding based on globally known locators and label rewriting. There are two relevant forms of *state in the packet*, source routing and encapsulation, which is discussed below.

A different way to divide up the schemes has to do with the relative power of expression in the various schemes. Traditional IP forwarding based on a globally known destination address allows the sender to name a destination. Source routing and label switching have in common that they allow the naming of a path to a destination. Naming a path is equivalent to naming a destination if there is only one path to the destination; this is the way Internet routing works. But if there is a need to deal with more than one path to a destination, then the ability to name paths is more expressive. This is one of the motivations behind most "Internet-centric" source routing proposals, so there is recognition in the Internet community that this expressive power is of some value. Naming a path, rather than a destination, allows for more control over multi-path routing, support quality of service routing, and other actions.

So to oversimplify, there is a two-by-two matrix.

	Destination-based	Path-based
State in packet	Encapsulation(sort of…)	Source route
State in router	"Classic" Internet	Label switching

Source routing

There are two high-level motivations for source routing. One is to simplify what the router does, both by removing the routing computation from the router, and by simplifying the forwarding process. The other motivation is to give the end-point control over the path of the packet, perhaps to allow the end-points to pick their providers, or to implement more general sorts of policy routing. Different schemes can be positioned in different parts of this landscape.

In SIP and SIPP, source routes do not offer any simplification to the routing or the forwarding. If a source address is included in a SIPP packet, each such address is a globally routed address. So the lookup in each router requires a search of a forwarding table of the same complexity as in the case of a simple locator. PIP had a slightly more compact representation of a source route, in which the intermediate elements in the source route (called Route Sequence Elements) are not full global addresses, but are only unique and meaningful within a hierarchically organized region of the network.

An even simpler version of source routing makes the individual elements of the source route only locally meaningful to each router. For example, a router could label its ports using a local index (1,2,3,...) and a source route could just be a sequence of the small numbers. This idea leads to very compact source routes (though they are still variable length), but means that the sender has to have a lot of router-specific information to construct the source route. So this idea is most often proposed in sub-nets, where the issues of scale are not so daunting. Examples of this idea include Paris, an early network designed at IBM (Cidon and Gopal 1988), and the link-id option in the Bananas scheme (Kaur, Kalyanaraman et al. 2003), which is otherwise a label-rewriting scheme (see below).

The other use of source routing is to give the sender control over the path the packet takes. In terms of policy and end-node control, SIPP contains a special form of an anycast address, called a cluster address, which can be used to identify a region (e.g. an AS or provider), which allows a source address to select a sequence of providers without picking the specific entry point into that router. This feature was called source selection policies, and while the use of source routing in SIPP was not restricted to this purpose, this was the only special form of locator provided to be used in the source route. [Hinden, 1994], describing SIPP, lists these other examples of the use of a source route: host mobility (route to current location), auto-readdressing (route to new address), and extended addressing (route to "sub-cloud").

An example of an early proposal to exploit the advantages of source routing is Sirpent [Cheriton, 1989], a scheme that the authors claim will support accounting, congestion control, user-controlled policy based routing and better security. source routing scheme to hook together disjoint addressing regions. In 1989, it was not clear that Internet would establish the ubiquity that it has, and there was a real concern that end-to-end connectivity would require bridging different architectures, such as OSI, or X.25. Sirpent was a source routing scheme to allow packets to traverse multiple heterogeneous addressing architectures. The paper notes that source routing can also address issues such as access control (the Sirpent source route contains an authorization token), congestion avoidance, and multicasting. In order for the sender to obtain a source route to the recipient, the Sirpent architecture assumes that there is a global name space (somewhat like the DNS) that can generate and return the source route from the sender to the receiver.

[Argyraki and Cheriton, 2004] propose WRAP, a loose source routing scheme that differs in detail from the IP option. The source route is carried in a shim layer between the IP and next layer, so it does not cause the IP processing to deal with a non-standard (slow path) header. They note that this scheme has some detail advantages– the source address in the actual packet is the from address of the last relay point, so a router can filter on this if it wants. But this seems a detail. The scheme is claimed to be useful in DoS filtering and QoS routing. The rewriting is done by a "non-router" element, so it could conceivably have functions beyond rewriting, but the authors do not discuss this.

Source routing and fault tolerance One of the issues with source routing is that if an element along the specified path has failed, there is no way for the network to remedy this problem and send the packet by an alternate path–the path has been specified by the user. In time, the user may be able to detect that the path has failed, and construct a new source route, but the process of discovery, localization and constructing a new source route may take much longer than the recomputation of routes done inside the network. A scheme called Slick Packets [Nguyen et al., 2011] proposes a solution to this–the source route is actually a directed acyclic graph, which gives each router along the specified source route a set of options for forwarding the packet. This scheme faces a number of challenges, of course, including constructing the graph and encoding it in the packet header is a sufficiently efficient manner. Compensating for these issues, the option of alternative routes to deal with failures means that information about short-term failures need not be propagated across the network to sources constructing source routes, since the alternative routes in the header will deal with these failures.

Label switching

As the previous discussion suggests, there are also a number of motivations for label switching, and divided schools of thought about different mechanisms and their merits. The debate between connection-oriented and connectionless (datagram) networks is as old as the Internet. Two important distinctions in the design philosophies are the unit of multiplexing and the value (and cost) of per-flow state. The mechanism of label switching is often a consequence of these other considerations, but it takes on a life of its own.

An argument used in favor of label switching over destination-based forwarding is that label switching (such as MPLS) can provide a more precise allocation of aggregates to circuits (for traffic engineering purposes) than using link weights in OSPF. But [Fortz and Thorup, 2000] argue that a global computation of OSPF weights can reproduce any desired pattern of traffic allocation. So it is claimed that destination-based forwarding and label switching are equally effective in this case (again, so long as there is one path).

One of the objections to label switching is that it seems to imply the overhead of circuit state setup. If the paths are in the "middle" of the net and used for carrying stable aggregates of traffic (as is the usual case with MPLS), then the overhead of path setup is not serious, since the paths are set up fairly statically as part of network management. But if label switching were used end-to-end per source-destination flow, it seems as if the use of label switching would imply a connection-oriented design with per-flow setup.

These two ideas can be separated. It is possible to use label switching with "Internet-style" route computation, datagram forwarding and no per-flow state in the routers. Bananas [Kaur et al., 2003] provides a clever solution to this problem: how to use label-switching and a constant size packet without doing per-flow state setup in the network.. The described purpose of Bananas is to allow multi-path routing , but it could be used in any context in which a set of *global*, or *well-known* routes can be computed. Assume that every node has some sort of global address. Conceptually, traverse each path backwards from the destination toward the source, computing at each stage a hash that represents the path (the sequence of addresses) to the destination. These paths and the associated hashes can be precomputed. For multi-path, use any well-known multi-path computation algorithm. At the source, compute the hash for the first node of the desired path. Put that and the destination into the packet. At each router, look up the destination, using some sort of prefix match together with an exact match with the hash. Rewrite the hash with the value (stored locally) of the hash of the sub-path starting at the next router. In this way, the hash values are a specialized form of label, the label rewriting is done based on the stored information in the forwarding table, and no flow setup is required. All that is required is that all parties agree on what subset of valid paths have been encoded. Some scheme is required for this (they suggest one), but this depends on the goal of the paths being computed. The paper discusses the operation of Bananas in a number of contexts, such as BGP. One might ask why this scheme is useful. The answer is that label switching might be more efficient, and different path setup schemes might be used at the same time; the source could select among them by selecting which label to put on the packet when it is launched.

Since both label switching and source routing can be used to specify a path, and both can be used with virtual circuit or datagram networks, one might ask whether they are fundamentally different in some way. The distinction

here is not one of expressivity but of control (and packet size). When source routes are used, it is the source that determines the path. When label switching is used, different parties can install the path, and the source only has the choice of selecting which path to use. In some cases, the source may not know the details of the path, but only the starting label. So label switching gives more options for which parties can configure paths. On the other hand, source routing allows a path to be established without any sort of distributed path setup. In a label switching network, every path must either be set up on demand or pre-computed. With source routing, a path can be specified by the source at the time it is needed.

New requirements

The previous discussion cataloged schemes based on a set of criteria, which include expressivity, efficiency (both forwarding and header size), and control over path determination. In the early days of the Internet, these were the primary considerations. [Francis, 1994a] offered the following criteria for selecting among candidates for IPng:

- Cost: hardware processing cost, address assignment complexity, control protocol (routing) complexity, header size.
- Functional capability–necessary: big enough hierarchical unicast address, multicast/shared-tree group address, multicast/source-tree groups address, scoped multicast group address, well-known multicast group address, mobility, multicast two-phase group address, domain-level policy route, host auto-address assignment.
- Functional capability–useful: ToS field, embedded link-layer address, node-level source route, anycast group addressing, anycast/two-phase group addresses.

All of these have to do with the expressive power of the forwarding scheme and its cost. But in more recent times, there has been a recognition that addressing and forwarding must be responsive to a broader set of requirements.

Addressing and security

The relationship between addressing and security is rather complex. It is hard to attack a machine if the attacker cannot find the machine's address, so some addressing schemes allow machines to hide their addresses to some extent. NAT is viewed as a useful, if incomplete, tool in securing a host, since it is hard to attack a machine one cannot address. If addressing is not a complete solution, it can be part of a solution that requires that an attack totally mimic normal behavior.

The i3 scheme [Stoica et al., 2004], described in Section 5.2, is a tool to protect a node from attack by keeping its address secret. In this scheme, a receiver controls who can send to it by installing a trigger, which is a form of a label, into a forwarding overlay. The sender is given the trigger, but not the actual destination. When the packet reaches the right forwarding point, the trigger is rewritten with the final overlay, and then destination-based addressing is used for the remainder of the path. However, hiding addresses and similar tactics cannot fully protect a "public" site from attack, since a machine must reveal itself to some way to be used. Once it reveals itself, a DDoS attack using zombies can mimic normal behavior in all respects and still attempt to overrun a server.

One of the design challenges in designing an indirection scheme such as i3 is whether the source and destination are equally trying to protect themselves from attack. Schemes such as i3 attempt to hide or protect the destination. If the destination is truly hidden, then when a packet goes in the reverse direction, the source of that packet must be hidden. This implies that the original destination (perhaps a server) has some measure of anonymity in the distant region. If i3 is used in a symmetric way to protect both ends from each other, then the identity of each end is not easily known by the other. This raises question of when a machine can hide its address (to protect itself) and when it must reveal its address (for reasons of accountability.)

The TVA scheme, described in [Yang et al., 2005], is an alternative way of protecting the destination from attack. Instead of hiding the destination address, packets must carry a specific authorization, a capability, to be allowed

to pass through the network. The forwarding scheme is the basic destination-based Internet mechanism, but the router (in particular the router at trust boundaries between ISPs) is responsible for checking the capabilities.(This design approach is what I called *intentional* delivery in Chapter 4.) The scheme actually uses a rather complex mix of mechanisms to implement its function. The packet carries a variable-length set of capabilities (which as something in common with a source address, in that it does not require state in the router to validate), but also uses soft state in the router and a nonce in the packet to avoid the variable length packet in most cases. It computes a incremental hash of the source path to assist in tracking sources and allocating capacity among different sources. It uses fair queueing to limit the congestion that one flow can cause to another.

Yet another indirection scheme is SOS [Keromytis et al., 2002], which is again designed to protect servers from being attacked. In this case, they restrict the problem to protecting servers that have a known and pre-determined set of clients–they do not offer SOS as a means to protect public servers. They use a three-tier system of defense. The server is protected by a filter that is topologically placed so that all packets to the server must go through it. They assume the filter can run at line speed, and cannot be flooded except as part of a general link flooding scheme. This means that the filtering must be simple, so they only filter on source address. To allow for more complex filtering, they require that all legitimate traffic to the filter first pass through an overlay mesh, where one of the overlay nodes has the knowledge of the location of the filter. The address of this node is secret, and the overlay uses DHT routing to get the packet to the right overlay node. To protect this layer, they have a set of secure access overlay access points (SOAPs), which perform the first line of checking and perform a hash on the destination address to get the identifier which is used to drive the DHT. The paper contains a discussion of the justification for this rather complex set of mechanisms, and an analysis of the various attacks that can be mounted against it.

SIFF [Yaar et al., 2004] allows a receiver to give a capability (permit to send) to a sender; the routers check these capabilities and reject them if forged, but otherwise give them priority over unmarked traffic. In this way traffic without a permit to send (including malicious traffic) is disadvantaged relative to favored traffic and presumably preferentially dropped as the network becomes fully loaded with attack traffic. Portcullis [Parno et al., 2007] is concerned with preventing attacks on the blocking system itself. Systems using capabilities to provide preferential service to selected flows offer strong protection for established network flows, the Denial-of-Capability (DoC) attack, which prevents new capability-setup packets from reaching the destination, limits the value of these systems. Portcullis mitigates DoC attacks by allocating scarce link bandwidth for connection establishment, and they argue that their approach is optimal, in that no algorithm of this sort can improve on their assurance.

All of these schemes, and both TVA and SOS in particular, have rather complex and rich sets of mechanisms, which arise when the full range of attacks are contemplated, defenses are selected for these attacks, and then these defenses in turn must be defended. This does beg the question of whether there is a different way, perhaps more simple, of factoring the security problem.

Tussle and economics:

the simple model of the Internet was that the network computed the routing, and everyone used the result. But both senders and receivers may want to have some control over where the traffic goes. Senders and receivers may want to pick a path through the network as part of picking a service provider, obtaining a specific QoS, avoiding a particular part of the network, and so on. Different third parties may also want to have some control over routing, which may cause them to invent a separate address space. This set of considerations have also been understood for some time. [Francis, 1994a] says:

> *There are several advantages to using a source routing approach for policy routing. First, every source may have its own policy constraints (for instance, certain acceptable use or billing policies). It is most efficient to limit distribution of this policy information to the sources themselves. Second, it may not be feasible to globally distribute policy information about transit networks. Further, some sources may have less need for detailed transit policy information*

than others. With a source routing approach, it is possible for sources to cache only the information they need, and from that information calculate the appropriate routes.

NIRA [Yang, 2003] (Yang 2003) is primarily about routing, and providing the ability for users to select routes. This objective is proposed in order to create a competitive market for routing, and impose the discipline of competition on ISPs. As a part of this, it proposes an efficient scheme to encode explicit routes in a packet. The method is to use addresses extravagantly, and to assign a separate address to each valid route in a region of the network. To control the cross-product explosion of sources and destinations, NIRA breaks the routes into three parts, a *source part*, a *middle part*, and a *destination part*. A packet carries (as usual) a source address and the destination address. For the first part of the path, the source address guides the packet. Across the middle (large global ISPs) traditional routing is used. For the final part, the destination address is used. So any node only has a separate address for each path to/from it and the middle part of the network, not all the way to the destination.

It is interesting to contrast NIRA and Bananas in this context. Bananas computes routes all the way from the source and the destination. As a result, there are a huge number of routes, and there is no plausible way to assign a distinct global id to each such route. Instead, it uses a clever trick to rewrite the pathID at each node. NIRA computes IDs for "route halves", and asserts that each one of these can have a unique id (an address) valid within that region of the net. So no rewriting is needed. In exchange for this simplicity, paths that do not follow a simple "up, across, and down" pattern require explicit source routing. Bananas can use, with equal efficiency, any route that has been precomputed.

Relation of addressing to identity and naming

There has been a long-standing set of arguments in favor of separating the notion of location (address) and identity. The Internet uses the IP address for both, which hinders mobility. But using the IP address for identity in the packet provides a weak form of security, and separating the two requires an analysis of the resulting issues. There is a hypothesis that if identity is separated from location, there is no form of identity weaker than strong encryption that is of any real value.

Proposals for separating identity from addressing include FARA [Clark et al., 2003], SNF [Jonsson et al., 2003] and PIP [Francis, 1994b].

[Jonsson et al., 2003] propose a split between names and locators, called SNF, for Split Naming Forwarding. They suggest that locators need not be global, and that source routing or state in translation gateways can be used to bridge addressing regimes. They offer little detail. They propose that naming is a least common denominator, so naming schemes must be well-specified and global. But there can be more than one, and this is good, in that different schemes can compete. Names map to things at locations, so it appears that they name machines, not higher level entities. They describe naming as an overlay that can route, but at low performance, somewhat like IPNL. They also propose an ephemeral correspondent identifier (ECI) that is used by the transport layer. This is visible in the packet, and becomes a short-term identifier that does not change if the locator changes.

PIP proposed a separate identifier (called the Endpoint identifier, or EID), 64 bits in size. The EID is used by the router at the last forwarding step to forward the packet to the end node, so it is used both for forwarding and identity. But it is not globally known or the basis for global routing. There is no structure in the EID that is useful for routing, and it is not known globally.

Fara explores the implications of separating the locator from the route, and in particular the implications of removing any EID from the "IP level" header all together and moving it to a header that is visible only to the endpoint entity. The question explored by FARA is whether there is any need to agree on the EID scheme, and whether it is necessary to be able to look up a location using the EID. The hypothesis in FARA is that the EID scheme can be private among the end-nodes, that higher level naming schemes like a DNS can be used to find the location of entities, and entities can manage their locations (e.g. they can move) without having to provide a means to "look them up" using the EID.

Most schemes that separate identities from location do use the identity as a way to "look up" the location. IPNL uses the DNS to look up the addresses to deal with NAT, as does 4+4. The Unmanaged Network Protocol [Ford, 2004] used "flat" identifiers that are public keys. That is, any node can make its own identifier, and later can prove that it is the entity with which this identifier goes by using the private key associated with the identifier. The scheme uses a DHT to allow nodes to incrementally find each other an establish paths across the DHT overlay among all the nodes. Turfnet [Pujol et al., 2005] , another scheme for tying independent addressing regions together by using a common naming scheme, uses [names of a kind I donfit understand.] These flat identifiers are flooded up the routing tree, which raises interesting performance issues with finding an entity.

Label switching–again

[Gold et al., 2004] propose a scheme called SelNet, a Virtualized Link Layer. It is a label-based forwarding scheme (somewhat like MPLS), with the feature that each label includes a next-hop destination and a selector, which is a generalization of a label that can trigger not just a re-writing but a range of services and actions. Actions might include forwarding, local delivery, multicast, etc. Further, actions can include removing a label, replacing a label, and so on. So a packet can be sent with a series of labels, which produces a variant of source-routing, or the labels can trigger rewriting, which is more stateful and more resembles MPLS. In this respect, SelNet is an interesting generalization.

The SelNet design does not constrain how actions (the things that selectors refer to) are established. They can be static and long-lasting, or they can be set up. SelNet includes a protocol somewhat like ARP, called XRP, extensible resolution protocol, which allows a sender to broadcast to find a receiver, and get back a address/selector pair in a reply. They observe that validation or verification can/should be done before returning this information, (in contrast to ARP, which always answers), which gives a measure of protection somewhat like a dynamic NAT. This idea of a security check before answering is a clever idea that allows for a range of checks, including application level checks. But it begs the question of what info should be in the request packet, which they do not elaborate.

The form this would take in a new architecture is not clear. They describe it as a link layer, or layer 2.5 scheme, but this seems to derive from the desire to interwork with IP. In a new scheme, this might be the way that IP worked. The novelty seems to be the idea of a selector with generalized and unspecified semantics, the separation of the forwarding from the (multiple) ways that selector state is set up, and the idea of a security check at label setup time. I believe that by defining a few selectors with global meeting (well known selectorsneeds a security analysis) this system can emulate several prior indirection schemes, such as MPLS.

Completely independent regions

Some of the schemes above try to deal with independent addressing regions (in particular NAT schemes, but more generally) by using a global identity scheme. These include IPNG, 4+4, SNF, FARA, Sirpent and Unmanaged Network Protocol. A more extreme challenge is to hook together regions that do not share any common naming, as well as no common addressing.

(Crowcroft, Hand et al. 2003) describes Plutarch. Plutarch is not just an indirection scheme, but is designed to cross-connect heterogeneous contexts, within which we find homogeneous addressing, etc. At the edges, there are interstitial functions (IF) that deal with addressing, naming, routing and transport. The core argument is that the context-crossing points should be explicit in the architecture. The paper concentrates on naming and addressing.

Their scheme is an example of a class of network where all contexts are equals (there is no distinct global context in which rendezvous can be facilitated. I propose the word concatinets to describe this class of world. The hard problems are well known: how to find a context, (and a route to it), and how to set up the necessary state. One approach is to have a central registry of contexts. In contrast to schemes such as Sirpent, Plutarch avoids any shared naming context, and proposes instead a gossip scheme that allows one context to search for another by name. Because of name conflict, multiple replies might show up. Each reply is a list of chained contexts, and Plutarch assumes that enough info comes back to disambiguate the different contexts that reply. The returned information is sort of like a source route across the series of contexts, and is sufficient to allow a setup of IF state

at the crossing points.

This particular architecture is defined by the extreme reluctance to have any layer of naming that is global. It does not emphasize any sort of precomputed routes, which raises many issues of scale. (In contrast, schemes like SelNet allow for and assume that some bits of forwarding state will be long-lasting, which implies some higher-level name space in which they can be described.

Mobility

Work on mobility seems to be particularly constrained by the current architecture, in particular the overloading of address with identity information. I do not attempt to discuss the large number of schemes for mobility, since this is as much about routing as addressing.

In general, schemes can be divided into end-to-end and network aware. In the end to end schemes, a host that moves gets a new address that reflects its new locationpart of an address block that is already routed across the network. So the routers see nothing special about a mobile host. In the network-aware, there is some sort of indirection, either in the routers or in a special node (e.g. a home server), so that the sending host does not have to be told about the move. There are issues of complexity, scale, speed of response.

(Mysore and Bharghavan 1997) make the point that multicast and mobility have many similarities. They explore the option of using current multicast as a way to track a mobile host. They note the major problemshow does the mobile host find the multicast tree to join, since flooding will be very expensive. They summarize the other problems, all of which arise from current details. This paper might be a nice input to a fresh study of mobility in a new architecture.

Lilith (Untz, Heusse et al. 2004) is an addressing/routing scheme for ad hoc nets of limited scope, where broadcast and flooding can be used. They use flooding to set up flows, and MPLS to utilize the flows. They note the interesting point that if you discover topology and routes at the same time, e.g. by using flooding, then you need a lower level set of addresses that scope the flooding. So they donfit use IP addresses for the labels, because IP broadcast only works within a subnet, and they are trying to build a subnet at the IP level. Because of the state in the routers, they call this a connection oriented approach, but this is a particular use of the term. They say that they prefer connections to allowing each forwarder to decide what to do, but it is not clear exactly what the dynamics of their route setup scheme is. It is not clear how this scheme would differ if the path setup message from the destination back toward the source set up an IP forwarding entry, rather than an MPLS label rewrite entry. (It would eliminate their ability to do multipath setup, and to have different paths to the same destination from different sources. But is there more?)

Relation of addressing to forwarding and routing Once we step back from the constraints of a particular architecture (such as the Internet), there are not that many fundamentals of address representation. There are global addresses, encapsulated addresses and rewritten addresses. Global is a special simple case where the address survives intact. Encapsulation represents "state in the packet", rewriting represents "state in the router". And, of course, there are hybrids.

More revealing and perhaps more interesting is where the state comes from. SelNet is a rewriting scheme where one way of setting up the rewriting state is by broadcast across the lower layer. So this feels like a link level mechanism, and is describes as such. Lilith has the same feel–it uses MPLS as the rewriting scheme but sets up state using a flooding protocol across an ad hoc network.

Making source routing robust

As I discuss above, there are a number of problems raised by source routing. One is that source routing seems to take control of resources away from the network operator and give it to the user. There is no reason to believe that an ISP will want to carry a packet for a user unless the ISP is going to be compensated, or at least is party to an agreement to carry that class of traffic. As well, perhaps giving control of the routing to the user creates a new and massive attack mechanisms, where the routes can be used to implement a Denial of Service attack against some part of the network. Another problem is that in a simple source routing scheme, there is no guarantee that

the packet will actually follow the specified path. Schemes have been proposed to try to address some of these issues.

Platypus [Raghavan et al., 2009] is an authenticated source routing system built around the concept of network capabilities. Platypus defines source routes at the level of the ISP–it defines a route as a series of "waypoints" that link ISPs. Inside an ISP default routing is used. A Platypus header is thus a sequence of capabilities, each specifying a waypoint. The process of obtaining a capability allows an ISP to maintain control over which traffic it agrees to carry.

Another scheme for creating robust and policy-compliant source routes is ICING [Naous et al., 2011]. ICING is described in Chapter 5; it is essentially the forwarding scheme for the Nebula proposal. And we have now followed the history of addressing and forwarding up to the era of the NSF Future Internet Architecture Program.