Toward the design of a Future Internet

David D. Clark

Version 5.0 of May 5, 2009

Version notes:

Version 1.0 : More or less a complete draft up through section 7. Section 8 is fragments that will be incorporated into next version. Whole document should be viewed as very rough and potentially very incomplete.

Version 2.0: First initial draft of sections on naming and addressing.

Version 2.1: Small fixes to tunnels and anycast discussion.

Version 3.0: section on application design patterns added.

Version 4.0: Preface and restructure of architectural design principles

Version 4.1, 4.2, 4.3: slight revisions. Section 8 still incomplete.

Version 5.0: Added section on longevity.

Notes for revision:

Acknowledgement:

The research reported here and the preparation of this document was supported by the Office of Naval Research under contract N00014-08-1-0898, and by the National Science Foundation under agreement 0836555. The opinions contained are those of the author, and do not reflect the opinions of the supporting agencies.

Preface

The origin of this document and where it is going.

This document is a very preliminary proposal for the design of a Future Internet—an outline of requirements and architecture. This document should only be seen as a first step in such a proposal; there are many parts that remain to be considered and elaborated. But it does try to offer a rationale for making key design systems.

Intellectual roots

This document draws on a number of sources for its insights. First, it draws on our collective experience with the current Internet—what works, what has survived, and what has eroded or broken down under the pressures of evolving requirements. Second, it draws on the reasoning to be found in many of the projects in the NSF FIND program, and the overall philosophy of that program. The FIND program encourages research that is not afraid to propose new ideas when they are justified, but which does not ignore the lessons of history. FIND researchers are expected to justify their ideas based on requirements, theories of design, and experience—ideas that will prove right in the long term, whether or not the idea fits well into the current Internet. This document tries to follow that design philosophy. Third, and more broadly, the document draws on the wide range of architectural research that has been done in the networking community, including some prior project with long-range architectural objectives, such as the DARPA NewArch project. I acknowledge the wide range of argued reasoning on which I have drawn.

Putting requirements first

The various sections of this document are organized around recognized clusters of requirements, such as security and management. They start with a discussion of what is known about how to deconstruct the requirement into component parts, and then a summary of what is generally accepted as the right way to address the requirement. The sections then lists some Points of View (POVs), perhaps conflicting, about paths to the future. Each section then tries to argue in favor of particular architectural preferences in order to meet these requirements.

This document represents a serious attempt to work from requirements to mechanism. In cases where requirements do not seem to imply any need for architectural consistency, the document tries to recognize that fact and "de-architect" the issue. In other words, this document tries to derive architecture from requirements, rather than from examination of mechanism.

The discussion covers the range of traditional layers from technology to application. The traditional view of Internet architecture has a focus on the packet layer and addressing. Perhaps more interesting is the section at the end on application design—a topic that received relatively little consideration in the design of the original Internet. It is my claim that proper application design is at least as important to a successful Future Internet as the mechanisms at the packet level. There is some discussion of the traditional packet or "pipe" level, but in many cases the discussion is not to put forward an architectural proposal, but to argue that architecture at this level is not as important as we might have thought. For example, I assert that some of the traditional arguments about addressing, such as whether there should be a global address space, is the wrong question. Based on the requirements and the design principles I put forward, I argue that a discussion about the scope of addressing is important, but the question of global addressing is misplaced.

Next steps

There are several ways to go forward from this document, all appropriate. One, there are many places in the document where the reasoning and conclusions are incomplete. Additions and amendments are welcome. Second, there are many forks in the road—points where there are diverging points of view, where designers could have taken a different path. In the context of the FIND program, NSF and the FIND leadership have always hoped that multiple ideas might emerge for the design of a Future Internet. Perhaps the initial discussion here will inspire alternative proposals. Finally, researchers may find herein suggestions for specific research project they might want to undertake. All of these paths forward are welcome.

Table of contents

1. Introduction

This document is a proposal for a Future Internet (FI). It offers a motivation as to why it is worth contemplating a Future Internet that is different from the Internet of today, and it makes some specific recommendations about overall design, or *architecture* for a FI.

The Internet of today is an amazing success. The issues that motivate a fresh look at the architecture of a FI are not any inability of the current Internet to support applications, or provide the basic service of forwarding packets, but rather a set of requirements that are contextual and future-looking. This section tries to set the state by reviewing some critical requirement for a FI.

1.1. The important requirements.

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. A FI must have a coherent security architecture, which makes clear what role the network, the application, the end node, etc. each has in improving security. The goal of security will have a primary role in shaping the architecture of a successful FI.

Availability and resilience: These goals are sometimes lumped into security, but have been be identified separately because of their importance, and because they are issues that arise independent of security issues. 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 an *architecture* for availability. While the Internet of today deals with specific sorts of failures (lost packets, links and routers that fail), it does not have an overall architecture.

Better 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 manaement problem was not clear in the early days of the design.

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*, only come into existence if some actor chooses to invest in them. As I will argue, 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. Any proposal for a FI must of necessity take a stance in this space, which I will call the *fundamental tussle*. 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.

Suited to the needs of society: The Internet is not just a technical artifact connecting computers, but a social artifact connecting people, deeply embedded in society. The success of a FI will depend on how it deals with important social issues (for example *identity*), some of which will come to the front as we address issues above such as security.

Long-lived: The proposed design must remain useful over time. One view is that a long=lived network must be *evolvable*; it must have the adaptablity 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.

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 PDA, but most probably the small, embedded processor acting as a sensor or actuator. At the same time, high-end computing will continue to grow, with huge server farms, cloud computing and the like. Any FI must somehow take this wide spectrum of computation into account.

Exploit tomorrow's networking: At least two technologies will be basic to tomorrow's networks, wireless and optical. Wireless (and mobility) imply new sorts of routing, the need for intermittent connectivity, and dealing with losses. Advanced optical networks can offer rapid reconfiguration of the network connectivity graph, which again has large implications for routing and traffic engineering.

Support tomorrow's applications: Today's Internet has proved versatile and flexible in supporting a range of applications. There is not some killer application that is blocked from emerging because of the current Internet. None the less, applications of today and tomorrow present requirements that a FI should take into account. These include a range of security requirements, support for highly available applications, new sorts of naming, and the like.

1.2. The different perspectives:

The traditional view of the Internet is that it is a platform for data carriage. In this respect, it would seem that it addresses the needs of the user, or perhaps the needs of the application designer. However, these are not the only actors in the Internet, and design choices should recognize the full range of these actors:

The network manager: As noted above, network management is not a well-developed part of the current Internet architecture. This adds cost and detracts from availability.

The investor: As noted above, the Internet cannot exist unless someone invests to deploy it. As a now-forgotten economist put it: "The Internet is about routing money; routing packets is a side-effect." Like it or not, a designer cannot simply ignore this point of view.

The attacker: While we do not need to be sympathetic to the goals of the attacker, we cannot pretend he does not exist. There will always be "bad guys" in the network, and we should catalog the range of motivations that drive them.

The user: If we think of the user as simply the actor who invokes applications, then we might conclude that the user is well-served. However, if we think about the user as the target of attacks, as an amateur network manager, or as an amateur investor (think about the developing world) the user is not well served at all. We should recognize all of the issues with which users must cope, if we want a FI to be as widely accepted as possible.

1.3. The format of this report

The first chapters of this report deal with external constraints that limit the design of a Future Internet. These chapters have roughly the same organization. They introduce a requirement such as security, and then discuss the implications of this requirement. The chapters begin with a summary of what might be taken as common wisdom—what most of us might agree about. It then lists some points of view (POVs) about how to deal with the issue, sometime conflicting. Following this background each chapter will put forward a proposal for the design of a FI. Since most mechanisms (such as addressing, for example) will be driven by many requirements, this discussion of a specific approach may have to contain some "forward references", but as much as possible the documents tries to offer concrete alternatives for the

architecture of a FI. Considerations and implications of these design proposals will be flagged as appropriate.

After the chapters on security, availability, economics and management, there is a chapter that provides a (very partial) discussion of design principles for architecture.

The final collection of chapters deal with mechanism design: naming, addressing, routing, transport protocols, packet headers and application design.

2. The fundamentals: external constraints.

This section discusses "givens": external considerations that will shape our work, but over which we do not have control. Certain points of view (POV) are cataloged with respect to these givens.

2.1. Infrastructure

Some of the facilities that make up the network (e.g. long distance fibers, towers, etc.) will be expensive to construct. The economics of networking (e.g. capital investment) cannot be ignored. (This fact raises the fundamental tussle mentioned above and elaborated below.)

Network technology will be heterogeneous, with different features and implications. A FI will have to include a solution for integrating these. (The present Internet had as a core goal interoperation among heterogeneous technologies. This goal must continue.)

- POV: Since Internet (and by extension a successful FI) will define the shape of the desired technology, technology can be expected to follow the architecture, rather than the architecture having to accept technology designed for other purposes. The architecture need not bend itself to try to incorporate technology built for different purposes. Additionally, careful design of the architecture can either facilitate or hinder the emergence of useful sorts of technology heterogeneity.

2.2. Industry structure

Different actors (providers) will run different parts of the FI. As today, some of these will be commercial, some enterprise and organizational, some governmental, etc.

- POV: 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.

These providers will need to interconnect, but may not have interests that are perfectly aligned.

- POV: The attention paid to the issues of interconnection will have a profound influence on the shape that emerges for the industry that supports an FI. In the original Internet, the designers downplayed the importance of this interface (sometimes called the network-network interface). That decision was perhaps unjustified.

We will continue to see the trend toward specialization in market sector and technology base, e.g. broadband consumer service provider vs. enterprise backbone provider vs. content/hosting provider. And, as discussed later, there will be an expanding range of higher-level services and distributed applications in the FI. This implies a richer mix of service providers.

- POV: By the principle of "Tussle Isolation", and taking into account the fundamental tussle, these mid- and higher-level services should not be designed so that facilities-based infrastructure providers have a privileged role in providing them.
- POV: If we do not accept the POV that facilities investment will require public sector involvement, then the above position may restrict the opportunities to reap

returns on facilities investment to the point that facilities buildout does not occur at a suitable rate. Some degree of vertical integration may be the price of a healthy infrastructure. Any FI design must think carefully about the extent to which it attempts to take an ex anti position on this point.

2.3. Computing technology

The FI we are considering is intended to connect to computing devices at its edges. So its nature will be shaped by the future trajectory of computing. We are moving to an era of more diversity in edge-devices, from high-performance servers and computing platforms to very small, inexpensive ubiquitous devices such as sensors and embedded computers.

There will always be general purpose computing devices, so the FI must be designed, as the Internet was designed, to support a wide range of applications, not just a fixed and known set.

- POV: This wide range of requirements for performance and for low-cost ubiquitous connectivity cannot be met by one set of standards for transport and interconnection. We will see the emergence of (at least) two sorts of FI, targeted toward each of these domains and only interconnected at higher levels.
- POV: Wrong. One set of standards will span this range of requirements just fine.
- POV: The requirement for generality does not mean that the FI has to be based on the same design everywhere. Interoperation and generality are distinct problems.

2.4. Edge networking characteristics

The diversity of network technology at the edge will continue to increase. Different modes of wireless will proliferate. A FI must deal with a much broader range of edge diversity than the model today where almost everything looks like an ethernet.

Mobility of edge devices and edge networks, both physically and in terms of changing points of attachment, will be a dominant pattern.

2.5. Social considerations

The network is not just a technology that hooks computers together, but is deeply embedded in the larger social, political and cultural context. Assuming that we aspire to build a global network, we must accept that different parts of the world will present a very different context into which the network must fit. We must balance this consideration with the fact that technology, especially as embedded in standards, tends to work the same everywhere. It will be necessary, as part of the design process, to think about how to avoid "baking in" un-necessary cultural norms. The network will be expected to work differently in different contexts.

There are always "bad guys", and they cannot be excluded from the network.

3. A starting point: some initial design considerations

This section lays our some assumptions about the design of a FI, and some initial proposals for architecture. This initial section should be seen as stage-setting, since many design proposals cannot be fully justified until all of the requirements have been considered.

3.1. Basic connectivity

Universal connectivity by mutual agreement: By default, any set of nodes that want to communicate should be able to do so. The above does not mean that all edges of the network should be able to send traffic to any other at will. Some action by the receiver to declare its willingness to receive traffic may be appropriate in order to reduce the range of possible malicious attacks. "Off by default" need not preclude willing end-points from communicating.

- POV: Any tools put in to the network to limit the ability to communicate will ease the difficulty that third parties, including unwelcome ones, have in blocking communication that the end points consider legitimate. The balance of control over which parties have control over the possible patterns of communication will be a key point of tussle, and must be designed with great care.

The core of the FI will continue to be built out of high-speed trunks (mostly optical). It should provide interactive connectivity (analogous to the service provided by the Internet today) between its edges. In other words, continuous connectivity should be assumed as the definition of "available" in the core of the network. (I will argue below that packet carriage between endpoints is the right design point for a FI, but this assertion is intended to be more general than that.)

However, at the edge of the network, both edge networks and edge devices will display a wide range of connection models, from continuous connectivity to very intermittent connectivity. In particular, sensor devices and networks and portable wireless devices will encounter intermittent connectivity, due to limits in network coverage and to conservation of power. Mobile networks (e.g. in cars, planes, etc.) will be common.

- POV: To deal with end-nodes and edge networks that are intermittently connected, an important mode of connectivity in a FI will be a staged mode, in which bundles of content are relayed from source to destination. This mode is sometimes called Delay Tolerant Networking, or Disruption Tolerant Networking (DTN).

DTN architecture: The DTN concept to support applications should be a core part of the architecture in a FI.

This proposal raises an important design decision, which is whether the staged delivery model should be the fundamental transport model of a FI, or whether staged delivery should be "on top of" a more classic interactive packet delivery model. This choice is an aspect of the fundamental tussle: should the facilities providers that own the routers also have unique control of the staged delivery service, or should that service be a competitive one running on top of the basic forwarding service. My preferred outcome is that staged delivery should not "replace" transport of packets, but should sit "on top" of it. This structure is very important in order to support the open and general nature of the future edge platform, since it will permit the emergence of a number of competing delivery services, just as we have a number of competing CDNs and email delivery services today. Thus, while edge devices may sometimes be disconnected, when they are

connected, the network should be able to provide them the same sort of interactive packet delivery that they would expect if they were always connected.

Some applications will be designed to adapt to variation in connectivity, other applications will not work properly between every sort of edge node/edge network. Supporting a a general communication mode that is as close to the one found in the "core" of the network will allow the widest range of applications to "sometimes work" at the intermittently connected edge.

Service management interface: Because there will be much more diversity in the connectivity service that applications will encounter in different circumstances, the application should be able to interrogate the network to ask what the nature of the available service is, as opposed to the "try it and see" model of today's Internet.

3.2. Semantics of the basic connectivity service

One of the advantages/implications of today's simple packet carriage is that since "what comes out is exactly what goes in", we do not need to define what aspects of the semantics are held invariant by the network. However, a more complex semantic model may have some compensating advantages.

- POV: The ability to communicate does not imply the strict requirement that packets be able to flow directly from source to destination with only router-like devices in the path. An alternative mode might be that the data is passed through other sorts of devices that transform the packets in some way that is well-understood, and transparent in the sense that it preserves key semantic properties.

Continued weak semantics: A FI should preserve the basic simplicity of the current forwarding model of the Internet—no transforms at the forwarding layer that require the end-nodes to have a complex model of which aspects of what is sent are preserved and which may be changed. Any transformations performed between the sender and the receiver should be invoked by the end-nodes, and should be consistent with the application being used.

[This point may be obscure at this point. ALF was a great idea until we tried to sort out such issues as packet loss. Then it fell apart. If the node where the ADU was transformed knew something of the semantics required by the application, however, ALF could be made to work well. So the proposal is that transformations between the sender and receiver may make sense in some cases, but should be seen as “above” the basic forwarding layer, and in service to the specific application being used.]

3.3. Multiplexing

Cost-effective operation will continue to depend, as it does today, on statistical sharing of capacity among the highly variable set of applications in use.

Packets: At the "edges" of the network, where aggregation is low and traffic mix is unpredictable, packets seem like a proven and effective technology.

- POV: The need to exploit statistical sharing in order to provide service at reasonable cost does not preclude features for QoS.
- POV: The need for forwarding of packets does not imply the need for a traditional routing protocol, or indeed any sort of dynamic routing algorithm at all.

Aggregates of packets: In (at least parts of) the "core" of the network, with large volumes of traffic with stable statistics, the network will deal with aggregates of packets, in a mode more like circuit switching.

- POV: The management of these "aggregate" multiplexing schemes should be integrated into the architecture, along with the packet level scheme. The reasons for this include the integration of the control and management schemes. This POV is in contrast to the implementation of these schemes at "level 2" today, as in MPLS.

We will continue to see a tension between the need for strict isolation (for reasons of security and predictability), and statistical sharing (for reasons of cost.) We should expect to see a growing number of approaches for sharing physical facilities in order to build systems with various degrees of isolation and sharing, and various sorts of service diversity. Examples include division of a fiber into lambdas, and the construction of separate networks over separate lambdas, or the use of time-division schedulers to produce isolation of different traffic groups.

- POV: An emerging view of how to share resources is virtualization. Virtualization implies that physical resources (such as links and packet forwarders) are virtualized, so that different higher-level networks can run at the same time on the same physical resources. Link virtualization is not a new idea--there are lots of ways that a link can be divided up, both in the frequency (lambda or channel), time (MPLS cells, etc), or code division. Virtualization of the processing elements is a newer concept in the network context, especially if it is seen as allowing different service providers to co-exist on the same hardware platforms.

3.4. Interconnection

The core of the net will be constructed by many providers, so there must be an effective solution to interconnection, which meets the rich set of needs. The problem of interconnection is much more than exchange of routing information and packet flows. There are issues of economics, security, and so on that must be considered as part of designing interconnection mechanisms.

Interconnection of aggregates: If the core of the network is based on aggregates of packets, then the interconnection paradigm must address the issue of aggregates. This fact is another reason why the concept of aggregates must be a part of the FI architecture, and not "below" the architecture.

3.5. Performance

The range of infrastructure performance (throughput, reliability, dynamic variability) found at the edge will continue to expand. The fast technology will get faster, and there will be new sorts of slower devices and networks that will be supported.

- POV: Absolute speed will continue to be an issue, and the design of a FI must balance complexity of packet processing with the desire to manufacture low-cost high-speed line cards.
- POV: On the contrary, the ability to do deep packet inspection suggests that the limit in line-card performance is not packet-processing overhead.
- POV: It is not clear what the relationship is between the design of a FI and the resulting implications for raw performance.

[Conclusion?]

3.6. Congestion and its control

While today the core of the Internet is engineered with the goal that congestion not normally occur, there will always be some parts of the network (e.g. wireless regions with constrained bandwidth) where congestion will occur, and for any flow, unless the rate is limited by the sender or receiver, it will be limited by some sort of congestion signal from the network. So congestion will always be a part of the FI, and it must be managed.

There has been a great deal of research on congestion control schemes, including schemes for explicit feedback of rates, and use of control theory to derive optimal schemes. While there has been little discussion of a preferred approach for an FI, there is a rich suite of research results to pick from.

"Relayering" congestion control: The current layer model of the Internet has congestion in the wrong layers. Since congestion arises in the interior of the Internet, the layer that is visible to the routers (analogous to the IP layer in the current Internet) needs to be more involved in congestion management, and an interface between this layer and the layer above it (e.g. some sort of transport protocol) should be part of the specification of these layers.

3.7. Quality of Service

- POV: The simplicity of the single, homogeneous best-effort delivery service will not be adequate to deal with the needs of tomorrow. A FI must be prepared to offer a range of transport service qualities. Evidence for this includes:
  - The widespread use of private intranets as a means to control service quality (among other reasons).
  - The popularity of current QoS tools in the enterprise context.
  - The emerging popularity of streaming media (but see below on the limited role of real-time delivery.)

QoS: A FI should support a range of service commitments (QoS) rather than a single best-effort service.

Whatever transport-level QoS features are provided in an FI, they should be provided for paths that cross multiple service providers. This objective has implications for economics and industry structure, discussed below.

- POV: Since there will be a greater range of service options, an application should be able interrogate the network to ask what is available, as opposed to the "try it and see" mode of the current Internet.

4. A framework for security

4.1. Background and points of view

While there is broad agreement that security is an important problem, and that there are many vulnerabilities in the Internet that should be mitigated, there seems to be little common agreement on the right approaches to take. This area is badly in need of some architectural framing, but even this POV may be disputed by some.

4.1.1. Confidentiality and integrity

In general, end-to-end encryption is seen as a success in dealing with confidentiality and integrity (for a simple definition of integrity) of data in transit. However, it is understood that the DNS names that are used in conjunction with a scheme such as SSL represent a path for deception and subversion of the encryption.

4.1.2. DDoS

Distributed denial of service attacks have received a lot of attention in the research community, but there is no obvious consensus on what the best approaches are to mitigate them, either in the network of today or tomorrow.

4.1.3. Protecting the insecure end-node

Again, there are a number of proposals in this area, including trusted hardware and virtual machines, and there seem to be wide differences in opinion as to which of these are worth pursuing.

4.2. Proposal for a more secure FI

My high-level assertion is that we can design a future Internet to be materially more secure.

The existing Internet, conceived in the mid-1970's, does not incorporate in its design any overall approach or framework to address issues of security. This fact means that dealing with security threats on the present Internet can be no more than a series of point-solutions to specific threats. Our understanding of network design principles is much more mature now, and we could do much better.

It is not realistic to imagine that a set of technical solutions will produce a network that is free of all vulnerabilities or attacks. The Internet is a technical artifact deeply embedded in a social, economic and human context. Attacks involve all those modalities. Our goal for the network technology should be to narrow the range of attacks, simplify the problem of detection and response, degrade certain forms of attack to the point that they are not useful to an attacker, and to allow the design of operational procedures for security to be positioned in the context of a clear model of what the network can and cannot do.

In the following sections, I break the "security problem" into a set of sub-problems that seem to make analysis easer to carry out. The sub-problems are:

- Building a more available network in the face of attacks.
- Dealing with attacks on an ongoing communication session.
- Providing a security framework for information objects.

- Improving our defenses against infiltration attacks on insecure end-nodes.
- Dealing with application-level vulnerabilities and attacks.
- Reducing the impact of DDoS attacks.

In each of these categories, I discuss specific improvements we can anticipate with respect to security.

4.2.1. Designing a future Internet with greatly increased availability and resilience.

In some respects, the Internet is very resilient in the face of component failures. Examples where the Internet does provide good availability in the face of failures include link failures and random packet loss. But these are point solutions to recognized specific failures. In general, it does not achieve the same reliability as the telephone system, and its design is not based on a systematic theory of availability, especially in the face of malicious attacks. Here is a proposal for a framework for availability.

A framework for availability must contain two parts: a set of methods to continue operation despite failures and attacks, and a set of methods to detect, localize and fix these events.

Continued operation in the face of failures

1) To achieve high availability, a network design must identify the resources critical to continued operation, and must allow for the incorporation of a rich diversity of heterogeneous alternatives for these resources. Heterogeneous resources are called for because heterogenous resources are less likely to present the same vulnerability. The ability to exploit diversity in resources must be supported both by the architecture (e.g. dynamic routing protocols) and in the deployed nets (e.g. provisioning of multiple disjoint links). (One of the advantages of the current Internet design, with respect to this approach, is that since the core function of the network is very simple, it is reasonable to identify and catalog all the critical resources.)

2) The protocols must provide the ability to detect that a failure has occurred. This requirement may sound obvious, but it is not. Many failures in the Internet today go undetected, unreported and unresolved. As discussed below, the use of an identity framework can turn many problems that today go undetected or mis-interpreted into clean failures, so that they can be consistently detected and hopefully corrected within the framework of availability.

3) The protocols must provide the ability for the actor at the point of detection to trigger reconfiguration so that different resources are brought into play. This is an application of the end-to-end argument, in its original form, to address correct operation in the face of failure. The idea is well-understood in the abstract, but has not been consistently applied, and this should be a goal of a future Internet architecture.

Failures manifest at the end-points: Many failures can only be detected at the end-points, where the application code is positioned and the application can relate what *should* be happening with what *is* happening. For example, a Byzantine attack by a router on a packet flow in which selected packets are deleted cannot be detected reliably by other routers in the network. Only the end-nodes know what was sent and what was received. This means that the ability to detect failures must be incorporated into end-node protocols, and end-nodes thus must have the ability to trigger network repair and reconfiguration.

Discriminating among failures: It is important for the network to contain tools to diagnose the class of failure, so that it is clear what sort of restoration of service should be undertaken. This requirement links the goals of availability to the goal of better network management. For example, if a connection cannot be initiated, it is important to discriminate

between a failure of the DNS and a routing failure. One must appeal to different sorts of diversity to work around these different classes of failure.

Thus, the minimal degree of discrimination required in the classification of failures is that degree necessary to allow the correct sort of restoration to be undertaken (e.g. routing recovery may not require that the end-node know exactly which node has failed—indeed it may represent a risk to allow that degree of visibility into the structure of the network.)

Recovery from failures: It is not enough to keep operating in the face of failure or attack. The source of the problem must be detected and restored (or isolated, in the case of an attack). Without this second step, the actual degree of diversity in the network may be less than is anticipated, and the attempt to reconfigure around a failed component may fail because the diverse resources are not actually in place and operational. This requirement can be somewhat decoupled from restoration of service, but it cannot be ignored. The issue of fault recovery is discussed in the section on network management; this requirement suggests the strong link that must exist between security and management.

4.2.2. Protecting the act of communication

We can design a future network in which communication is resilient to attacks launched against the communication by the network (or by an agent that has corrupted some part of the network).

As noted above, end-to-end encryption does a reasonable job today of protecting the confidentiality of data being communicated. Encryption also prevents packet modification and insertion. Other attacks on the communication itself can take the form of dropping attacks (in which selective packets are discarded), and redirection attacks (in which packets are sent to a point other then the intended end-point). Dropping attacks are at least to some extent managed through the retransmission mechanisms, and if they can be detected as a serious failure or attack (as opposed to an isolated drop) can be addressed within the availability framework above. Redirection attacks are more complex to deal with. In today's network, redirection attacks yield a wide range of consequences for which we have no methodical way to respond. Redirection attacks do occur today, e.g. DNS corruption or mis-direction of packets to the wrong IP address.

Dealing with redirection attacks

If the network (or more specifically the communicating end-nodes) can detect that a redirection attack has occurred, and that one of the end-nodes in the communication is not the intended communicant, then communication can be terminated, and the result can be dealt with as a availability failure, as discussed above. The problem today is that many redirection attacks are not detected as such. To detect them, what is needed is an end-to-end confirmation, as part of initiating a communication, to allow each end of a communication to confirm that the other ends are the ones that are expected.

There are two consequences of mapping redirection attacks into availability failures. First, by reducing the range of failures (many indirection attacks today produce obscure results), it is easier for the application designer to build in means to respond to the failure. Second, the network can use the same set of tools to recover from dropping attacks and redirection attacks—the tools described above to enhance availability.

End-to-end identity checks: What is thus called for as part of protecting the act of communication is including a confirming check, such as a cryptographic exchange like SSL, as part of normal session initiation, in any situation where there are concerns about security. This should be done independent of the use of encryption to deal with confidentiality concerns.

4.2.3. A more secure network architecture will include information security

Today, the assurance we provide with respect to the integrity and authenticity of information is based on providing a secure connection to the source (e.g. Web protocols using SSL.) Once the information (e.g. a web page) is downloaded, there is no way later to confirm the validity of the information. If it is forwarded on, e.g. as an email attachment or using a P2P system, all evidence about validity is lost.

Signed information: A FI should include pre-defined means for the creator of information to sign that information in a way that can be used to verify its integrity and authenticity. Once the assurance is attached to the information rather than the connection, we can partition the delivery problem from the integrity problem. This approach will greatly enhance our approach to availability and dissemination.

(To some extent, we have this capability with signed email, which is today a staged delivery. We do not have an equivalent capability for web pages and other sorts of information objects.)

- POV: If we presume that information can be signed, there is a clean division between the problems of network security and information security/assurance (IA). The lack of a clean IA architecture makes it hard to delineate the boundary between the two areas. However, signed information (which is a small part of the IA problem), illustrates that we can separate the two problem areas by some simple steps such as this. The IA problem is a critical one, but I do not discuss it further here.

One can view signed information as an analog, at the information layer, of turning redirection attacks into failures of availability. The design approach is to wrap an end-to-end check around the information, so that the receiver (independent of how the information is received) can verify that the information is not fraudulent. An attempt to deceive can be detected, and results in a condition where the receiver has just failed to receive what was sought—an availability problem. We can then address this using the same means as above: a rich and diverse means to search for and retrieve information.

4.2.4. A Framework for protecting insecure hosts

This section concentrates on attacks that involve *infiltration* of the end-node. Denial of service attacks are discussed in the next section.

Infiltration

By looking at attacks on the *communication* separately from attacks on *hosts*, we end up with a parsing of the problem into parts that require distinct (and to some extent independent) responses.

With respect to infiltration attacks, evidence from the present suggests that once a machine has been infiltrated, the attacker can turn that penetration to any purpose, including espionage (exfiltration), sabotage, or using the machine as a platform for further attacks. So there are two ways to address this class of attack: either prevent penetration in the first place, or limit particular sorts of damage (those that are deemed serious in the specific context) that can arise subsequent to infiltration, such as theft of information.

- POV: When we consider the problem of attacks on hosts, we must accept that general-purpose end-node operating systems such as Windows or Unix will always have flaws that present vulnerabilities.

- POV: There will not be any single silver bullet that will solve the problem of host infiltration. What will be required is a series of compatible actions that together raise the work factor to the point that adequate assurance has been achieved.

As a starting point to analysis, it helps to catalog the different actors in the attack and its defense, and assess the tools that each actor in the system has to deal with attacks. This situation is asymmetric—different actors have different tools.

- Applications, by their design, define the available patterns of communication, and what is revealed or concealed in the messages being communicated. For example, the design of email permits staged delivery, which in turn allows for the "out-sourcing" of virus checking and spam filtering. With respect to encryption, while encryption can occur at different levels in the system (link, end-to-end or application), only the application level can discriminate among different parts of the communicated information, encrypting some but revealing other parts.
- Users and their end-node computers control the initiation of activity.
- While the end-node software may always be insecure, tools such as trusted hardware, trusted peripherals, virtualization and a trusted computing base may allow us to depend on certain well-specified operations.
- Networks control topology and completion of connections. (e.g. who talks to whom under what circumstances.) Examples include both physical topology (making sure traffic actually passes through a firewall) and logical (e.g. VPNs). Networks (less abstractly, the devices in them) can see what is not encrypted and change what is not signed. There is no monolithic network, but different parts that may be trusted (or not) to different extents.
- While general purpose end-nodes will always be insecure, it should be possible to build fixed-function devices that are resistant to attack.

Design of each part cannot thwart the powers of the others, but can blunt their instruments. For example, encryption of information prevents networks from doing packet inspection and discriminating among different sorts of traffic based on content. This blunts their tools of control.

The research challenge: given the above, can we devise trusted components, application patterns of communication, and controls on connectivity, which together can protect an untrustworthy end-node and mitigate attacks (or their bad consequences)?

While we cannot ignore direct attacks on the OS, and other sorts of tunneling attacks, flaws and vulnerabilities in applications will always be a serious issue. So we must involve the application designer in the process of building secure systems. In doing so, we must be mindful to make the requirements on the application designer as simple and clear as possible. Most application designers are not trained in security.

Segregation of the problem space: Today, the research community tends to look at the insecurity of the end-node as a monolithic problem, since many hosts run the same (insecure) software. However, we should demand that operational steps be taken to separate machines into different classes based on their duties.

- Machines (e.g. Web servers) that agree to serve content to "anyone" are particularly vulnerable to attack, since incoming messages cannot be filtered based on sender. So they should not be used for any task (such as hosting confidential information) that makes them valuable as a platform if they are infiltrated.

- Machines that host valuable information should not also be used for any roles where they need to connect to unknown persons. Any machine used for this sort of task should be embedded in a strong identity/authentication/authorization architecture.

This division allows us to focus on the most vulnerable machines: the general purpose PCs (e.g. laptops) that are used for a wide range of activities, sometimes move outside the perimeter of a protection zone, and so on. These machines will warrant much of our attention, but it is important to remember when we deal with them that we are not also dealing with Web servers.

Guidance to application designers: A future network must include *design patterns* for secure applications as part of its design framework.

Distributed security services architecture: Application-specific (or more generally application-aware) devices can be placed in the network. If these are simple fixed-purpose devices, it should be possible to make them secure. Provided that the application has been designed in such a way that these devices can be incorporated into the patterns of communication without extra effort on the part of the user, these devices can implement checks that are designed to mitigate undesirable actions performed by/to insecure end node. The network, by controlling patterns of communication, can insure that these devices are not bypassed.

The objective of our research must be to develop and demonstrate patterns of communication (e.g. design patterns) that can be implemented easily by application designers to achieve these goals.

Examples:

This discussion of a distributed security services architecture may seem very abstract. Here are some examples from well-known protocols.

Email: The ability to impose virus-checkers and spam strippers into the path of email depends on the fact that the email delivery protocols allow for staged or relayed delivery through mail transfer agents. While the motivation for staged email delivery was not primarily security, we should note the benefit that arose from the design and explicitly capture it in future application protocols. In the future, if email were embedded in an identity architecture, an email filtering service could do much more to protect the receiver from unwanted mail.

Telnet: In contrast to email, the early version of Telnet did not allow any sort of intermediate node to be positioned in the path from one end to the other (there was no application-level addressing or naming that would allow the setup of a relayed path.) While it may be argued that there was no need for such a service, there is no option for including one.

Web: The original design of HTTP did not make it easy to deploy caches and CDNs. While they have been retrofitted, the design is somewhat of a kludge in places. There was originally no thought about which parts of the Web service might be implemented as a separate service. We now see the composition of higher-level services on top of the basic Web architecture in what is sometimes called Web 2.0.

Security-related services

As part of defining useful design patterns for application designers, we should catalog the range of security-related services that might be defined as separate elements that can be moved from the untrustworthy end-node to trusted service point.

Identity and authorization: With respect to security, one obvious task to be moved to a secure service node is checking of identity and/or authorization to contact the node being

protected. For public web servers that are willing to talk to "anyone", the value of such protection may be minimal, but such nodes are in the small minority—most nodes on the Internet do not expect to be contacted by unknown senders, and a check of credentials can make sure that most unwelcome contacts never occur.

Content-checking: Today email is checked for viruses and filtered to remove spam. Similar content-checking may be useful in other applications.

Blocking of unexpected application flows: If an application has not been installed on the end-node, then packet flows associated with that application should not be permitted. This rule may seem simple, but raises complexity with respect to management. When a new application is intentionally installed, the blocking rules should be changed without user involvement. But if the user is not invoked to change the rules, then an attacker may be able to change the blocking rules as part of the attack, thus eliminating the benefit. Protection of nodes from infiltration will require attention to ease of management of the end-node.

Perimeter defense and indirection

Today, the firewall is used to prevent unwelcome traffic from reaching a network or host (and also, in reverse mode preventing unwanted traffic from leaving.). The firewall blocks many attacks, but not all of them, because the firewall lacks the visibility and the necessary information to do a fine discrimination. Several current research projects have explored the concept of an *indirection scheme* of some sort, in which the sender does not send packets directly to the destination, but only to a relay node of some sort, that performs some security check and lets only selected packets through.

There is a lot to be said about the use of this sort of protection, both in preventing infiltration attacks and denial of service attacks. These various ideas differ in details—the placement of the points of protection in the network, what sort of information they can see, and so on. For the moment, we should note that these devices may be application-aware or application-independent. If they are application-aware, then they may become much more subtle tools, if the communication patterns of the application have been designed to reveal information that makes detecting acceptable flows easier and more precise.

Balancing the needs of society

Once we put such an architecture in place, we must recognize that such security services may be deployed both by trustworthy agents and by those who want to attack the communication. There is thus a balancing act between preventing attacks on hosts (or mitigating the consequences of attacks on hosts) and facilitating attacks on communication using the same tools. The application design patterns must be designed so that they sharpen to tools of control for nodes that are trusted, and blunt the same tools for nodes that are not trusted. One obvious example is to strip out of the information being sent security related information such as identity credentials so that other parts of the network do not have access to that meta-data.

Exfiltration

If the protections against infiltration fail, and a malicious attacker manages to penetrate and take over a machine, then the fallback position must be to limit consequential damage. One consequence is the use of the machine as part of a bot-net, discussed below. Another consequence is theft of information, often called *exfiltration.* Again, the tools of defense will include application design patterns, the use of trusted service nodes, and network-level controls on what connections are to be permitted.

To control exfiltration, assuming that the end-node holding the information has been corrupted, what is needed is application designs that move to trusted nodes the key policy

decisions that will determine whether the transfer is appropriate, and network controls to prevent the transfer from occurring without authorization from this element. The result is that some sort of an export permit from a trusted node is required before exporting information from a secure region of the network. This (presumably) would be based on a trustworthy set of identity credentials from all the parties to the transfer.

If, in contrast, the (legitimate) goal is unfettered information dissemination, a very different application-level communication pattern must be employed, in which the information is pushed into p2p and pub-sub frameworks that explicitly do not demand authentication. (This use of heterogeneous dissemination mechanisms should be seen as an example of our availability principle.) Application designs must be created that help to distinguish between these two sorts of use-cases, and "separate" them so that they can each be managed to the desired outcome. Design patterns must help the application designer understand how to achieve this goal.

While these can be seen as "arbitrary" alternative, the key (as discussed below) is the architected management of identity—either demanding it in the first case a or explicitly not in the second case.

The mobile machine: The above discussion depends on the embedding of the insecure machine in a secure context in which a trustworthy region of the network prevents unauthorized patterns of communication, and which requires (for example) export permits to move data out from that protected region. This picture is less clear for a machine such as a laptop, which can be physically moved by its owner outside the protected region. This case is perhaps the most challenging to reason about, because we can no longer depend on the context of trusted services and network-constrained patterns of communication.

In general, the solution must be to provide some sort of "portable trusted context" from which such machines cannot escape. Whether this is done by some sort of inviolate trusted computing base or some simple hardware device (easier to imagine for a wired connection than a wireless one), the purpose of this TCB must be to prevent broad classes of actions when the machine is in an insecure context. This approach in turn will raise lots of usability issues that will have to be mitigated. However, in general, the idea that a machine that is "on the net" at all will have to carry out certain actions only via a VPN connection back to the secure region does not seem like a major barrier to usability.

4.2.5. Mitigation of application-level vulnerabilities and attacks.

The concept of application design patterns is central to helping application designers recognize potential vulnerabilities and design to mitigate them. The most pressing examples today include email and web vulnerabilities, but all applications will presumably bring with them their own set of security issues. The range of options here is huge, and will require the study of existing applications to detect common patterns that can be generalized, but the obvious security-related tasks can be identified from the previous discussions:

- Identity checking, user authorization and tools for accountability
- Content checking and the filtering of "bad stuff" from untrusted correspondents.
- Logging of actions in support of audit and accountability.

The open and transparent nature of the Internet is best suited to parties that want to communicate without any interference or intermediation. However, if parties that do not fully trust each other undertake to communicate, they may prefer a mode of communication that is constrained rather than open and transparent. Under these circumstances, the network should not be totally open, nor totally closed. We need to recognize this reality in the design.

The tools above can be used to address this situation, using applications that have been designed to support communication patterns and appropriate revelation of what is being sent, trusted elements that can be interposed to perform such checks as can be designed as part of the application, and the network to ensure that this node is interposed in the path.

4.2.6. Tools to mitigate and degrade the potential of denial of service (DoS) attacks.

[This section will benefit from a comprehensive review of literature on DDoS attacks and defense.] DoS attacks seem to fit into a distinct category. While they might be classified as an attack on a host, they do not require infiltration of the host. However, in order to build a botnet for DDoS attacks, the preliminary infiltration of otherwise innocent hosts is required. In contrast to availability attacks that manifest as "point failures" around which the network can route with suitable design, DoS attacks are characterized by a highly distributed nature.

DoS attacks have a distinctive set of features.

- They are visible by design. While infiltration attacks may be designed to be stealthy and undetected, there is no such thing as a successful DoS attack that nobody notices.
- They are based on attack amplification. A single attacker achieves the desired effect only through the recruitment (usually unwitting) of other machines.
- The particular machines used for the attack do not matter, only the number of them. This shapes the nature of the recruitment phase.
- Because DoS attacks involve attack amplification, mitigating them will be a game of statistics.

In general, mitigating DoS attacks will require the combination of a number of steps, each of which shifts the balance from the attacker to the defender. These steps fall into two general categories:

Degrade the botnets.

Make it harder for the attacker to assemble and hold onto his collection of penetrated end-nodes. The tools described above to control infiltration are directly relevant here. What is distinct is the statistical nature of the game and the indiscriminate targeting. While an attacker intent on exfiltration will target a particular machine for infiltration, the creator of a bot-net does not care which machines he infiltrates, so long as he has the tools to exploit the particular system in question. This suggests techniques such as using random permutation of system and network state to disrupt the ability of a bot-master to infiltrate hosts in large quantity with one attack.

Other approaches include helping the owner of a machine detect that the machine is infested, and providing incentives to clean the machine up. If the user can be given suitable incentives, then the process of cleanup can be monetized.

Degrade the attacks.

Reduce the utility of a bot-net of a given size by reducing its effectiveness as an attack tool.

Detect and flag: Detect possibly infested attack machines and flag their traffic, so that it can be dropped if it is participating in an attack. Today, it is practical to detect that a machine is infested with malware and part of an bot-net. The question of what to do with such a machine is harder. If it is just disconnected, this will trigger a call to a service desk, which transfers cost to the ISP and does not seem to have the right incentives. What is needed is a response where "the

punishment fits the crime", where the crime is letting your machine get infested. The idea of tagging the traffic near the source (e.g. by the access ISP), but only dropping the traffic if the traffic seems to be doing "something bad" will have a less dramatic consequence for the owner, and could be structured to have the right incentives for cleanup.

Diffuse attacks: Another way to mitigate a DDoS attack is to diffuse it across a large attack surface. If the attack is somehow directed so that it impinges on a large set of nodes, rather than just one node, then the impact on each of these nodes may be manageable. One way to implement this would be to give the machine being protected some sort of anycast address (DNS nodes are sometimes protected in this way.) It would not always be necessary for the node itself to be replicated behind the anycast address, but just a set of machines that "front" for the host, performing such security-related tasks as identity-checking.

Rate-limit or charge flows (linking to economics): Rate-limiting is a specific example of thinking in economic terms about attack and defense. A DDoS attack is a deliberate attempt to cause congestion, either of a server or a region of the network. If our FI has a strong framework for dealing with congestion, then those mechanisms will be triggered by a DDoS attack. Perhaps the consequence is that the co-opted machines that launch the attack will get a bill, which would provide an incentive for them to clean up their machine. Alternatively, attacking machines may be rate-limited in ways that allow legitimate users to continue preferentially.

Distinguish between public and private servers

We must accept that for a machine that accepts connections from anyone, it is difficult in the limit to distinguish between benign and malicious overload. So long as the intended users are human (i.e. this is not a machine-to-machine web interaction), techniques such as CAPTCHAs can be used to detect bots.

But for machines that do not need to communicate with "anyone", we can draw on techniques such as indirection, VPNs, etc., to allow nodes to control who sends to them. If these services are replicated and addressed using anycast, then attacks will be substantially reduced in impact.

4.2.7. Identity, authorization and accountability

To support the above, a future network must contain mechanisms to allow the parties to a communication to verify with whom they are communicating. The previous sections have depended on this capability in a number of ways.

- Detect misdirection attacks on communication.
- Allow application/network to pick desired communication pattern, to insert the desired degree of checking into the path between communicating parties, depending on the degree of trust between the parties.
- Detect invalid (unauthentic) pieces of information.
- Validate identity/authority of incoming connections to prevent infiltration attacks.

Such mechanisms are sometimes called "identity" mechanisms, but the word "identity" has many associations that trigger concerns. While a future network must have mechanisms to confirm whom we are talking to, different mechanisms will be needed under different circumstances. Sometimes it will be helpful if each end can present to the other credentials certified by a trusted third party, sometimes the parties to the communication will have (and will prefer) a private means to verify their mutual identities. (An example of the latter mode is the idea of *continuity-based* key exchange.)

Sometimes it will be helpful to have the evidence of identity revealed in the packets, so that third parties can see it. Some times it will be valuable to have it hidden except to the end-points themselves.

This is an example of the principle that the application can, by controlling what is revealed and what is hidden, change the range of operating modalities and the balance between the end-nodes, and trusted and untrusted parts of the network.

Whatever identity schemes are developed to meet this need, they will be the next target of attack. The design of these schemes gives us a change to engineer in good security from the start. Attacks will include availability attacks, identity theft and deception and redirection attacks. Deception will be attempted at all levels, from low-level attacks on technology to high-level deception attacks on humans and the tools they use (such as browsers). One way to make these identity schemes more robust in the face of attack is to see identity itself as a rich space of heterogeneous mechanisms and information. The approach we have proposed to deal with availability is a strategy we can use to enhance identity management.

Identity architecture:

Since different schemes will be needed in different contexts, and the context will to some extent be independent of application, the architecture of a future network should define a framework for management of identity information, but not constrain what form that information takes. Different schemes can then be used by applications as they tailor their overall communication patterns to the circumstances at hand.

5. Availability

The goal of availability is sometimes lumped into security, along with confidentiality and integrity. In the previous section on security, I did address availability issues. However, availability warrants a separate discussion, since it is so very important for the Internet today, and for any FI tomorrow.

Overall, the Internet of today is not as available as the telephone system. To equal or exceed that level of availability seems like an important goal for any successful FI.

5.1. Background

There is no general theory of network availability. The articulation of such a theory (or an argument that such a theory is not relevant) seems like an important research goal.

The early design of the Internet proceeded pragmatically. It was recognized in the original design of the Internet that routers and links would fail. So dynamic routing was developed as a means to build a more available network. Since DNS servers might fail, replication and parallel search of DNS was developed to build a more available DNS. Since (to first order) the only components in the original Internet were links, routers and the DNS, the issue of availability was dealt with in a reasonable way by point-solutions to an enumerated set of potential failures.

Similarly, since it was understood that packets would be lost, TCP was designed with a robust retransmission mechanism. The result is a highly resilient packet retransmission scheme.

Given this, one of the limiting factors in the availability of today's Internet is the limitations of these schemes. Slow convergence in dynamic routing hinders quick response to link and router failures. Constraints imposed on inter-ISP interconnection due to business plans may limit the range of options in case of failures. While TCP will keep trying to retransmit "forever", a lossy link still degrades throughput and responsiveness. TCP has no way to "route around" a lossy link.

5.2. Management

Some studies [cite] have shown that perhaps 30% to 50% of all network outages are caused by operator error. This fact implies that to improve availability by an order of magnitude will require a major assault on the management tools in the Internet. There does not seem to be any general view as to how to go about this.

5.3. Application-level availability

Many user-perceived availability problems arise inside applications, not at the packet level. To the user, the layered model of the Internet is less visible and less clear than to a network expert. If email fails or Web pages are not available, this is seen as an Internet problem to the user. So to achieve a high, overall level of user-perceived availability, designers will have to pay more attention to the application layer; designing dynamic recovery schemes to deal with component failures, and adding management tools to allow effective fault isolation and recovery.

6. Economics

6.1. Fundamentals

As stated in the earlier section on constraints, the following seem fundamental: some facilities are expensive (require significant up-front capital expenditure), and different parts of any FI will be built by different actors.

Since facilities are expensive, a successful FI must be structured so as to motivate ongoing investment. In this context, there will be a persistent tension between the desire of investors to capture the benefits of their investments by building systems that are closed in some way, and the presumed desire of FI designers to deploy an open system that permits easy entry by unrelated innovators. The so-called open nature of the current Internet, serving as a platform for a wide range of experimentation and innovation in different applications and services, seems like a valuable attribute to preserve in a FI. The design of a FI must thus encourage investment without allowing the investors to have total control over the resulting network.

6.1.1. The Fundamental tussle

The *fundamental tussle*, mentioned earlier, is the result of this reality about investment and control. Broadly, the tussle about the overall control of the network includes a range of actors including governments and regulators, application designers, users, spammers, copyright holders and the like. But the "narrow form" of the fundamental tussle focuses on the special role of the investor in facilities, and the degree to which ownership of facilities privileges the owner in the control of the net.

Various new features have been proposed for a FI, including mechanisms for naming and disseminating information, management of names and identity, management of aggregates of packets, and the like. Each of these can be designed in such a way that it is integrated and entangled with the basic connectivity and forwarding mechanisms that are implemented by the facilities owner, or (alternatively) can be designed in such a way that they are separable, so that they can be implemented by providers unrelated to the facilities owners. These sorts of decisions will influence both the motivation to invest and the extent of control by the facilities owners.

Roughly speaking, we can look at the different options for privileging the facilities owner as alternative "cut points" in the layering of the design.

- Perhaps the most extreme alternative being proposed for a FI is that of fully virtualized facilities, in which the facilities owner provides physical links and routers, which are then "virtualized" and sold to service providers at the next level, who then define the approaches to be taken for packetization, routing, security, and the like. This approach provides tremendous flexibility and options for competition among different architectures (as I have used the term here), but which seem to provide the facilities owners with the smallest resulting business opportunities.
- A slightly higher cut point would give the facilities owner control over the basic architecture of packets, QoS, some aspects of security, but would allow competing routing and addressing schemes to exist, which would allow competition in service delivery, the use of addressing as a tool of security, and so on. Since (in this version) the facilities owner still has no control over routing, some framework would be required to provide the right signals and incentives so

that the right circuits and routers are put in place to support the needs of the higher-level service provider.

- Slightly above that would be the option where the facilities owner provides one "default" routing protocol but allows others to run at the same time. This version now gives the facilities owner some control over the use of paths, which makes it easier to justify investment decisions. However, if a open market for competitive routing does not emerge, some sort of structural separation might be required to protect those who seek to sell competitive routing schemes.
- Moving the cut-point higher, one can imagine a FI in which the facilities provider implements services such as DTN forwarding and information dissemination.
- Even higher up, one can imagine that the facilities owners have control over the name space for services and information objects, users and the like.

For wireless networks, the analysis is more complex (and remains incomplete), because the right technical layering for wireless networks is unclear. One technical argument about wireless networks is that the abstraction of "links" is inappropriate. Rather, one must design forwarding and routing schemes that are much more directly linked to the physical aspects of the system. If this is so, then it is unclear how one would create a cut-point that allows competition in routing. Another design proposal for wireless networks is cross-layer design or cross-layer optimization. The idea behind this approach is that higher-level mechanisms such as retransmission should be integrated with lower level technology so as to deal with technology-specific issues. Again, this cross-layer linkage would seem to make certain sorts of virtualization more difficult to contemplate.

Competition in routing

While the correct cut-point to privilege the facilities owner is not clear, I believe that competition in routing, forwarding and management of address spaces is the right general objective. This will raise both technical issues (e.g. wireless), and regulatory (e.g. "network neutrality" with respect to competing routing algorithms. However, this cut point allows competition in schemes that directly bear on security, availability and management.

6.2. Interconnection

Since different parts of the network will be built by different actors, these actors must agree to interconnect in order that a global network come into being. However, these actors may not have interests that are totally aligned--they may be competitors even as they need to cooperate. This means that the features of the FI that deal with interconnection must be designed to deal with economic and business issues, not just simple data forwarding. Points of interconnection will be important points of tussle, and the protocols designed to deal with interconnection will facilitate or limit the various interactions among these regions.

In the Internet today, the most important example (and perhaps the only important example) of an interconnection protocol is BGP. It is generally considered that the functionality provided by BGP is not adequate to deal with today's range of technical, security and business issues. However, routing (e.g. the objective addressed by BGP) is not the only set of issues that interconnected regions will want to deal with. Better mechanisms and protocols might address a number of issues that arise at the inter-region interface:

- Enhanced QoS and related transport services.
- Routing, service provisioning, accounting and billing.

- Security
    - Control of DDoS attacks
    - Detection of false routing assertions
- Management
    - Diagnosis of faults

It is worth noting that if higher-layer third parties are allowed to deploy their own routing protocols, then the points of interconnection among different facilities owners no longer define the points where regional and global routing protocols meet. There is no reason to believe that the EGP/IGP structure of today's Internet is the right approach at all. (Consider, for example, the internal structure of a global CDN such as Akamai. Their system may have hierarchy, but the structure need have nothing to do with facilities boundaries.)

6.2.1. Interconnection of higher-level services

While BGP is the only protocol defined as "part of the Internet", lots of higher level services exchange data and control. In some cases, ISPs may be operating these services, but since they are not at the IP layer, and not directly tied to facilities, they are not normally thought of as "interconnection issues". These include:

- Services with a signaling protocol, such as SIP
- Services that monitor network operation, such as CDNs
- Application-level systems such as email.

As a part of the work on application design patterns, we must pay attention to the economic motivations that arise when different parts of these higher-level services are deployed by different operators.

6.3. Congestion

Congestion and its control is an economic problem, not a technical one. There is a large body of literature that develops this idea. Congestion, in economic terms, arises when the behavior of one user (or set of users) create cost (e.g. delay or lower throughput) to another set of users. The issue with economic congestion is that the locus of congestion (perhaps a point of interconnection between two ISPs) may be far removed from the agents (e.g. the original senders) that are the cause of the congestion. This seems to imply the requirement for a network mechanism to convey information between the locus of congestion and the agents.

- POV: The recent work by Briscoe on *re-feedback* provides the correct flow of information, because it allows a network forwarding a packet to determine the probability that the packet will cause congestion "downstream" toward the receiver. This knowledge allows that first network to limit or charge the source appropriately.

Re-feedback: a scheme similar to re-feedback should be a part of the economic design of a FI, and should be integrated into its technical controls for congestion.

6.4. Competition

Competition imposes a discipline on service providers. Giving the consumer choice among service providers generates an incentive to offer superior service, and superior value for

money. The early design of the Internet did not take this consideration into account, and tended to view the decision as to whether (or not) to provide user choice as a technical alternative.

An example of a design choice that is debated today is giving the user some control over traffic routing, especially at the provider level. Right now, providers cannot compete to offer better services because the user has no ability to choose alternative routes (except by using overlay networks and other complex mechanisms.) Earlier, in the discussion of availability, I observed that to allow the user to route around failures, the user had to have some control over route selection. However, in that context, we could propose both explicit route selection and implicit route selection, where the user choice is limited to "give me a new route that is different from the old route". Implicit route selection weakens the vigor of competition to market superior services, and from this economic perspective, explicit route selection would be preferred. However, as I will note later, explicit route selection may raise security concerns in certain circumstances.

6.5. Regulation

The original Internet was not much concerned with issues of regulation, and its design did not take into account the needs and requirement of various regulators and other "government-like" actors. Any proposal for a FI will (in today's climate) receive scrutiny from those who wish to exercise various sorts of control over the network. It is not just investors who seek to control or regulate what is done using their facilities.

7. Management

7.1. Background

The original design of the Internet was motivated by the challenge of getting the basic function of data (packet) transport to work across heterogeneous networks. At the time of the initial design, there was not a clear model of what the range of management problems would be, or how to design/integrate mechanisms into the architecture to deal with these problems. As a result, tools for management have been to some extent "glued on" to the original Internet.

Effective management is a challenge today, both for large ISPs (who have high operations costs for network management and who must hire highly-skilled and trained employees just to manage day-to-day operations) and for individual users (for example, residential consumers, who have few tools and little recourse if their home networks fail).

7.1.1. Theory and architecture

The research community does not have a model for how to frame or modularize the problem of network management. There does not seem to be any theory for how to conceptualize the problem.

- POV: there is no such thing as “management”, just a collection of essentially unrelated functions. The only important thing in common among these functions is that they will share some critical interfaces (see below) and involve people.

7.1.2. Industry structure

Since different parts of the Internet are deployed by different actors (who may not have aligned interests or the willingness to share internal information fully), management across interconnection boundaries is a significant issue. Today's Internet provides few tools to facilitate inter-provider management. Management (and operation in general) across the regions operated by different actors raise a number of issues, including:

- We cannot assume all parts of the network are equally reliable, trusted, or stable.
- A change in one place can trigger an apparent fault in another.

7.1.3. How to decompose the management problem

There are several ways to slice the management problem into parts, none proven right by any modularity principle.

One is to look for critical interfaces, which seem to break the problem into modules that relate to some fundamentals of the network. There are two candidate approaches to modularity, both potentially useful:

- Layering: recognizing the distinction between applications, network and technology. I have assumed that the FI will have to work with heterogeneous network technology. Each such technology, just as today, will have its own sorts of management requirements. And since the network is to support multiple applications, each of these will have management issues. A FI should pay more attention to the issues associated with managing applications.
- Regions: recognizing that different parts of the network will be operated by different actors.

Another way to break the problem into parts is functional: to look at broad classes of problem areas. At least two such areas are obvious:

- Diagnosis and correction failures and problems.
- Network planning and configuration.

In the discussion that follows, I will use the functional decomposition to structure the section, and point out the importance of the critical interfaces as appropriate.

7.2. Fault diagnosis and correction.

What is a fault? We understand the idea of failure, but not precisely. Any change can disrupt some operation. A fault is an *unintended* change, but diagnosis will proceed identically whether the change is intentional or not.

The concept of fault or change makes sense, in general, only in the concept of a pattern of use, e.g. an application. If a physical box fails (e.g. fail-stop) the situation is simple and (usually) clear, but if an application module fails on a box (so that it still responds to pings, for example), the only way to proceed involves testing at the level of application behavior.

Application-level fault management: As part of application design patterns, we need to suggest models for diagnosis, reporting and correction that can be built into all applications.

Applications today are usually designed to deal with component failures. For example, if a mail transfer agent is down, MX records provide a way to fail over to a different agent. If this does not work, the mail is just queued for later delivery. However, applications usually do not include tools to try to report or correct the fault. They presume some other means will serve to discover and correct the fault over time. If one mail transfer agent discovers that another agent is down, there is no protocol to report this fact.

It is interesting (if inconclusive) to speculate on why this sort of reporting mechanism has not been implemented. One answer is that the protocol seems complex, and is not considered important. Another answer is that since applications, in general, span operational boundaries, this mechanism would be an example of an inter-region mechanisms—in that same ignored category as the network-network interface. Specifically, since the protocol would have to cross operational boundaries, there are issues of trust and security that would have to be worked out. In some cases, one region may not want to reveal to another region the information necessary for accurate fault analysis.

An important aspect of application-level fault management is that applications run (among other places) on the end-nodes of users, and it is often only at those points where the intention of the user can be modeled, so that the presence of a failure can be detected. This implies that the process of detecting, isolating and reporting a fault will not only involve crossing region boundaries, but must involve the end-nodes.

The above points notwithstanding, a FI must include techniques to identify and report failed or malicious components in a distributed system. This capability is central to the goal of increased availability.

Diagnosis: In the Internet today, we see several patterns.

- The “N-1” approach, where components are invoked in a linear chain, and each component is responsible for detecting that the next component has failed. (For example, each mail transfer agent can usually detect when the next agent is not operating properly, so long as the failure manifests as a non-standard outcome in the transfer request.)

- End-initiated tests: The best example is at the packet level: *traceroute*. Traceroute gives the initial sender some clue as to which router along the path has failed. Because of the way addresses are assigned, and because traceroute is an afterthought, the evidence is sometimes misleading, but it about the best there is today. There are no obvious examples at higher layers—there is no mail-level equivalent to tracetoute, for example. Trace-route uses a "in-series" method, and is flawed in that a node being tested can detect that it is being tested. This will not matter for faults, but will not work in general for malice.
- To avoid the risk that the failing node can detect (and react to) the fact that it is being tested, there are also "in-parallel" techniques, which are not much used (to my knowledge) in the Internet today. Imagine a hop-count with the semantics that when it counts down to zero, the node in question duplicates the message and sends it by a disjoint path. Then the recipient reports if it got one but not two copies.
- Instrumentation of the data plane: Protocols, both at the packet and the application level, could be augmented with features that help to isolate a point of failure along a path. This approach also tries to avoid the problem of a malicious node disrupting the data but responding to the diagnosis messages.
- There may be forms of collaborative tomography that can detect malicious nodes, but we must remember that a malicious node may be attacking just one connection.

In the design of tools to diagnose failures, it may help to distinguish "normal" and "malicious" failures. Again, the two ideas may have a blurry boundary, but the distinction may prune off lots of obvious cases.

To the extent that we cannot isolate the locus of failure, the degree of bypass or restoration must be larger in scope (as discussed in the section on availability), and the detecting node cannot take specific steps to describe or request correction of the error. Schemes such as tunnels mask the existence of physical elements and turn them into simpler virtual elements. That makes diagnosis at the physical layer impossible. Virtualization and other schemes that mask physical structure make diagnosis harder, and require that larger regions of the network be bypassed in response to failures.

The methods proposed must not allow them to be subverted into attacks, or attack amplifiers. Any methods for diagnosis that expect a node in the network to respond are in principle a tool for a DoS attack.

Reporting failures: If a failure can be isolated to a specific component, then there is the option of reporting that failure to the owner/operator of that component. By and large, the Internet has not included mechanisms of this sort, for a number of reasons. The design of such a mechanism requires us to consider a number of questions:

- What needs to be standardized or "architected" to make this work? One obvious requirement is a clear means to name or identify components. Another is a means to establish a communications path to the operator of such a component. There must be some means to determine where to report the failure. Finally, there must be some sort of language, representation or what you will to permit a dialog about the status of a component.
- Failure-reporting messages need to be delivered in times of failure. What does this mean for packet forwarding?

- This is a new set of mechanisms to attack. And can they be used for attack amplification?
- If I make an intentional change, which breaks things in other parts of the network, to what extent am I forced by this scheme to reveal that I did so. Am I allowed to "change silently"?
- As an alternative, to what extent can components self-test themselves? Can we build a system in which "wait" is the correct response to a perceived failure? If the change is intentional, then I would wait forever. How do we deal with this?

In the limit, the availability of the system cannot depend on the reporting capability. If the component and its operator just vanish, a user of the service has to be able to cope. So allowing the node that detects the failure to just "route around it" and keep going must be the last recourse.

Where are failures reported? In a FI, part of the management architecture should be a view that all components have a "management" interface, as well as a service interface. The present Internet pays reasonable attention to the service interface: services can be named to some extent in the DNS, and so on. But dealing with failures seems to imply that any service component sits in some larger management context, which will hopefully keep operating even if the component itself is failing. So an obvious extension to the "DNS replacement" in a FI is that when one looks up a service, not only the service interface but also the management interface is returned.

Dealing with intentional change: Intentional change can come in all forms, but as a starting point to understand the issues, perhaps we can simplify change as "out of service", or "in service". A basic function would be to take a component out of service in an orderly way, so that it does not trigger diagnosis.

One option would be to enhance the "DNS replacement" so that when it is queried about a service it can return some sort of "out of service" notice to the query. If an invocation of the service fails, the diagnosis protocols should require that the invoker retrace the steps that were used to find the service, so that such a notice can be detected. That is an easy form of notification (and also of declaring that no further diagnosis is required because the fault has been noted.)

As a part of taking a component out of service, there needs to be an agreement about how much advanced notice is required. A protocol might allow existing actions to complete, but refuse new actions (on the grounds that they should have been sent elsewhere), if there is a limit on how long existing actions can persist.

7.3. System planning and configuration.

The other major functional category of management task is system planning and configuration. It is not clear (to me) to what extent this set of objectives is deeply tangled with the architecture of the network. This area requires further study and thought.

7.4. Management of applications

Since the Internet is a neutral platform for a range of applications, and does not have (at the packet level) any idea what these applications are, one cannot detect or debug failures of applications at the packet level. The Internet, lacking any model of what should be happening,

cannot detect failures. This fact implies that the locus of fault detection and resolution must be the end-node--the platform where the application code runs.

However, there has been as little attention to management tasks (e.g. fault detection and recovery, reconfiguration, etc.) at the application layer. There are no design patterns, nor any support tools, to make applications managed. This problem is perhaps even more challenging than management of the router-level Internet, because each application will have its own structure of servers and services, its own industry structure, and it own failure modes.

As part of providing design patterns for application designers, we should provide advice on how to make highly distributed applications easy to deploy, diagnose and maintain.

8. Longevity

In comparison to many artifacts of computing, the Internet has lived to an old age—it is over 35 years old. 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.

The objective of longevity 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. 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.

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.

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.

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.

8.1. Architecture and longevity

The term "architecture" is often used 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. Again, with respect to architecture and change, there are two subclasses of theories:

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.

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.

9. Architecture design principles

The preceding discussions of various requirements provide one of the inputs to the architecture design process. This design process should also take into account what we have learned about design, based on our experience with the Internet, emerging theory, and the like.

9.1. A history lesson—packet addressing

The initial idea for Internet addresses was that they would be drawn from a single, global address space. The Internet has diverged from that idea over time, with private address spaces, NAT devices and the like. The 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. Further, it has turned out that there is no way to enforce the idea of a global address space, had the architects attempted to do so

What constrains the range of addressing that is available in the Internet is the expressive power of the packet header, which has more to do with its format than any semantics. The IP address field has had a most interesting history in which the only constants are that it is a 32 bit field, that whatever value it has at each end must remain constant for the life of a TCP connection (because of the pseudo-header) and that at any locale in the network, addresses must provide the basis for some router action. Addresses 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 matters is that they are 32 bits long, and that at any point, they must have at least local meaning to a forwarding process.

. We were initially fearful that if we deviated from the simple concept of global addressing, 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 a common meaning to those addresses. Those self-consistent regions need not be global.

The recognition of regions is very important in the analysis of addressing because within a region, where an address has a consistent meaning, it will be easier to recover from an error that caused a packet to lose its way towards its destination. Without a meaningful address, there is probably no way to recover. So the size of regions, and the design of addresses within a region, will have an important influence over resilience, and other issues as well such as security and management.

However, the more important lesson concerns the initial ideal of global addressing. While it was stated as a design principle, there was nothing in the architecture that constrained the use of addresses to be global in scope. Without such a constraint, it is not surprising that the way addresses were used evolved away from the initial design, based on emerging requirements. (It could be argued that we tried to constrain addresses to be global though the use of the TCP pseudo-header, but that the constraint failed in face of divergent incentives.)

9.2. Design by constraint

[This term and the ideas behind it are principally due to John Wroclawski.]

The idea of "design by constraint" is that architecture should be seen as something that precludes certain behavior and permit other, rather than a catalog of objectives or goals. As a practical matter, it is the constraints of a system that determine what it can and cannot do. Thus, the format of the IP header has turned out to be a significant constraint. Since the initial design did not provide any means to reformat the syntax of the header between parts of the network, to change the header implies massive coordination (as we now see in the debates over IPv6). Nothing constrained the semantics of the address field, so it evolved and diverged from the initial concept over time. An alternative approach would have been to define the semantics of the network, and define which aspects of the semantics must be preserved among the communicating end-points. If the architecture defined the semantics, then the design could have easily included "format converters". Once these had taken root in the network, it would have proved as difficult to change the semantic definition as it is today to change the format definition.

One can argue about this specific choice—would it be better to have fixed the semantics or the syntax? In fact, this alternative was argued in the early days of the Internet; the decision to fix the format and let the semantics float was based on the fear that fixing the semantics would be more likely to reduce the generality of the network, in the face of the great unknowns back then.

But the more general question is what sorts of constraints can actually be brought into play to help shape an architecture. Unless a constraint is real, and "constraining", it is not worth discussing.

9.2.1. External and design constraints

Section 2 lists some external, or fundamental constraints, such as the cost of facilities and the existence of bad guys. These are very real, and invite creative ways to invalidate them, since they can constrain in painful ways.

More interesting are the *design constraints*: those ideas intentionally included in the architecture to shape the network and its evolution. In general, constraints seem to take the form of "fixed points" in the design that become hard to change over time. The packet header has this characteristic. At a higher level, DNS names and their embedding in URLs are hard to change. While the conversion of names to location/address can change, the syntax (again) is hard to evolve, since it is embedded in lots of systems.

Different sort of addresses can be seen as offering both capabilities and constraints. The idea of *anycast* (discussed below), for example, is a constraint on the sender in that it prevents the sender from having a name for an instance of a service, but only a service name. This prevents the sender from using the address to attack an instance of a service. Constraints such as that, which can be added by one actor to limit the behavior of another actor, are interesting, in that while the architecture might call for them, it cannot make them happen.

Another class of constraint is one that can be imposed by the end-node. If end-nodes are allowed to encrypt much of the packet, then the network is constrained from using that information as part of its operation. However, if the network had grown up differently, and the network had come to depend on access to that information, that outcome might (as a practical matter) prevent the introduction of end-to-end encryption. Again, this sort of constraint seems as much a race between two ideas to see which can become entrenched first, which is not really a fundamental design tool.

9.2.2. Approaches to constraint

The idea of "hard to change" seems like a fundamental aspect of a constraint. However, that concept is probably rich and nuanced once it is explored, and there may be specific

techniques for introducing constraints into an architecture that should be illustrated and cataloged. A rich exploration of this space should be part of the foundational work in architecture design.

9.3. Carrots and sticks

Constraints seem like "design by sticks", things that keep you from doing something. One can also "design with carrots", by providing a means that is not mandatory, but is so convenient that it becomes the norm even though it is not mandatory. Examples include the widespread acceptance of TCP and the DNS. Indeed, one can find an ongoing low-level philosophy debate as to whether the DNS is "part of the Internet architecture", since it is not technically mandatory. In fact, carrots turn into sticks (constraints) over time, and in this respect represent a "high-wire act" approach to architecture design, since if the carrot fails, the stick never emerges (to mangle an image in a horrible way.)

The next section, for example, proposes a number of objectives for naming. These schemes should be seen as carrots trying to grow up into sticks, as the DNS did.

9.4. Incentive alignment and regions

Lacking (in many cases) the tools to "force" a constraint on designers, the most realistic approach is to find a set of actors with aligned interests, and use that alignment as a way to insert a constraint into the design. In this respect global constraints are the most difficult, since global alignment of interests will not often occur, except at the most basic of level (having the Internet is better than not having it…). As we saw in the example of addressing, and as we have argued with respect to security, availability, and economics, it is a very powerful idea that the Internet can work in different ways in different contexts. This idea in turn suggests that the idea of "regions of like-minded actors" is the idea that will best allow an architecture to grow, evolve and survive over time. Recognition of the idea that actors in the Internet do not always have aligned interests, and a resulting approach to design that allows "tussle" to be contained, and to have a different outcome in different contexts, is a critical idea.

9.5. The minimality principle

Given the above, part of our job in designing an FI is to discover all the places where we do *not* have to agree—places where our architecture need not propose a constraint. Where we can "de-architect" an idea, we will be better off so long as the network still works. [Dan Geer, a noted security expert, stated the design goal for Internet security as being "as insecure as possible so long as it still works".] We should resist the temptation to excesses of architecture design, and we should resist the temptation to proposed objectives for which we have no matching constraint to enforce the conformance to that objective.

9.6. The architecture of security

The requirements of security are an important case study here, because in the case of security, the objectives of the parties are not aligned. There is no way to exploit incentive alignment to enforce constraints. In this case, it is necessary to fall back on the limited set of constraints (including external ones) that are really fundamental. These sorts of constraints are discussed in section 4.2.4. Briefly, the approach is to recognize that the network is not homogeneous. The sender (attacker?) and receiver (defender?) are each positioned within a region of the network in which they can hopefully place some trust. Within each region, the interests are aligned. Between the two such regions is a middle region where both parties make minimal assumptions—basic forwarding and the ability to route around failures.

Within each region, network has control over topology, routing, and which nodes can connect. Because the network can control patterns of connection, the network can enforce application-level patterns of communication (see section 14 below) to further the goals of that actor. Then, when each end point has configured their region to maximum advantage, they engage. If the application patterns are properly used, the balance of power comes out in favor of the desired party, and the result is at worst a simple failure of connectivity.

9.7. Formal theories of architecture derivation.

[Doyle stuff?]

[What else?]

10. Naming

10.1. Background

The current Internet has only one "architected" name space, the set of Domain Names. Domain Names were initially devised when it still seemed as if physical machines were the right entities to name. It has since been exploited to name services (e.g. the suffix in an email address), and information objects (e.g. URLs).

- POV: While the domain name system has proved robust and scalable, domain names are not optimal for the naming of either services or information objects.

The use of names cannot be separated from the process of converting from names to lower-level identifiers such as addresses. To the extent possible, I have discussed issues of addressing in the section to follow, but there is necessary entanglement.

10.2. Naming of services

The term "service" can be used to describe almost anything in networks, from application layers services (e.g. mail transfer agents) to the QoS that is (sometimes) provided by the network layer. What I am referring to here are those services that are explicitly (or *intentionally*) invoked by other service elements or end-points. Since they are explicitly invoked, there must be some explicit way to identify them.

In this context, an important feature of services is that they can be implemented on more than one physical machine, and (to a degree that depends on the specific service), the party invoking the service will not care which instance of the service is selected. The earliest example of using the DNS to name a service is the suffix in email addresses, which is used as the name of the mail server that should receive the mail. To "unbind" the name from a physical address, an application-specific feature was added to the DNS: MX records. Email works fairly well, but MX records seem like an ad hoc addition with little generality. DNS SRV records are a more general option for allowing the DNS to name services, rather than hosts, and to identify more than one instance of the service, and is closer to what will be required in a FI.

There are a number of reasons why the designer of a service may want to replicate it on multiple physical machines. One is to provide the service at locations distributed around the network, to offer low latency and higher performance when the service is invoked. Another is to spread the load over multiple machines to avoid overload on one physical machine. (If this were the only goal, the machines could be physically co-located). A third reason is replication for availability, and there are a set of reasons related to security, discussed below.

Beyond the use of MX and SRV records, the DNS has been exploited to name multiple instances of a service by deploying a specialized DNS server responsible for the name of the service, which implements some sort of specialized computation to select the preferred physical address when presented with a service name. So long as the operator of the service also operates the relevant part of the DNS, this "trick" can be quite powerful.

10.2.1. Service Naming System (SNS)

I propose that the FI should be equipped with a system designed specifically for naming services, called the Service Naming System, or SNS. SNS could be seen as the DNS replacement, but there might also be other services used to name physical machines, people, information objects and the like.

A primary task of the SNS is to take in a name for a service, and return some set of "addresses", or "identifiers" that can be presented to the packet forwarding system. More on addressing below. In addition to this function (which is essentially all the DNS does), other parts of this document have identified other functions that the SNS should implement.

Service authenticity: as part of mitigating "redirection attacks" and other sorts of attacks, the discussion on security proposed that any initiation of a connection should include some sort of authentication verification to confirm that the service that has been reached is the service that was intended. To do this, the user of the service will require some sort of certificate that binds the credentials of the service to a public key for that service. Today, TLS transfers a certificate to the client once the connection has been invoked, but there may be (I would argue that there are) advantages in having the certificate in hand before initiating the connection. To implement this option, the certificate can be returned by the SNS along with the name.

Service management information: in the discussion of management, I proposed that the SNS should return not only the "address" of the service, but the "address" of the management interface to the service. This would be a separate interface to which one could report failures, enquire about service status, and the like.

Service invocation method: in the Internet of today, if there are any specific requirements for invoking the service, they are specified as part of the service itself. If the service uses UDP, or multiple TCP connections, or requires some sort of special authentication method, this is generally "hard-coded" into the specification of the application. However, in a number of places I have proposed that applications should be designed to allow different communication patterns to be used in different circumstances. In this scheme, some aspects of the invocation may not be static, but determined more dynamically at invocation time. For example, one mail service might require an encrypted connection and authentication, while another might accept clear-text connections from anyone. Part of what the SNS should return are the specific requirements that specify how this service should be invoked.

10.2.2. Design of the SNS

The hierarchical nature of the DNS, and its approaches to replication, have proved very robust. There seems no reason to move away from this approach. So I proposes that Service Names be hierarchical, just as they are today. There are two issues that may call for a modified design.

Support for dynamic information: as described above, the SNS will return more information than the DNS, including bulky information (e.g. service certificates) and dynamic information (e.g. service status information). These capabilities do not seem consistent with a server that is dealing with millions of names, as the ".com" servers do. The design of the names and the hierarchy should assume that the SNS server that provides information about a service will be operated by that service itself, or by a SNS provider that serves a small number of services. This is what happens today when the DNS is used by a service provider to implement "special" name to address binding to implement regionalization or load balancing. This structure will be the norm.

Resistance to attack: Today, the DNS can be replicated, and is resilient to failures of physical nodes. But it is not very resilient to attack. Nodes can be corrupted, the cache information can be "poisoned", and so on. This has led to the design of DNSSEC, which tries to assure that the sequence of hierarchical queries leads to the retrieval of the correct address. This security architecture will need to be rethought. If there is an end-to-end confirmation of identity at the time a connection is made, this will detect corruption of the SNS information. However, corruption of the SNS will still leave open attacks based on deception of the human-level user and

to availability problems. More work is required to sort out the range of attacks against which the SNS must be resistant.

However, at a high level, my earlier discussion of availability suggests that one tool that will increase availability of the system is heterogeneous replication. In the DNS community there has been tremendous resistance to having naming information available that is not authoritative, because dependence on non-authoritative information may increase the chance of mis-direction and malicious manipulation. However, if the client has access to heterogeneous tools for name resolution, and the proper use of these tools is imbedded in the software (e.g. user training is not required), heterogeneity in the naming system may lead to increased availability. The benefit of widespread use of end-to-end verification of identity is that using a non-authoritative naming service cannot "make things worse"—a wrong answer is like no answer at all, an availability failure. But a right answer can be independently confirmed without having to "trust" the naming service.

10.2.3. Dynamic address binding

As discussed above, the DNS is used today to allow a dynamic binding from name to address, to meet a range of needs including distributed service replication and load balancing. These are important goals, and should be available in a FI. However, there seem to be two "levels" at which they can be implemented: as part of the name-to-address conversion, or within the addressing system itself, where one address is mapped to another address. The best example of such an idea is anycasting, which is an afterthought in the current Internet. I will propose that the widespread use of anycast addresses is a valuable feature in the FI, but this is discussed below in the section on addressing. (The preference for anycast over dynamic binding as part of the SNS lookup is better mitigation of DoS attacks.)

10.3. Naming information objects

Today, the most common form of name for information is the URL, and its more general variants the URI. There has been a great deal written about naming of information—a book might suffice to review the literature. Here I propose a few key issues for the naming and identification of information in a FI.

10.3.1. Authentication

I argued in the discussion of security that information objects should be signed by their creator, with credentials attached to the object itself, so that the validity of an information object can be confirmed independent of how it is retrieved. Today, we use assurance about the information channel (e.g. TLS) to provide assurance about the objects retrieved. This means that if objects are retrieved and then passed on, for example as email attachments or by being put into some P2P distribution system, the validity can no longer be confirmed. A better approach is to attach the validating credentials to the information.

For email, we have mechanisms to allow the sender of email to sign (and to encrypt) the email. While these are little-used today, they suggest the correct way to go.

10.3.2. Independence of name and location

Just as we have learned that today's IP addresses mingle two ideas: identity and location, we see the URLs of today mingle the idea of the location where the information can be found (the DNS name embedded in the URL) and the idea of some long-term identifier for the information. These concepts need to be separated.

Information storage and dissemination: In a FI, there should be many services for information storage and dissemination. By the principle of tussle isolation and the fundamental tussle, I believe that there should be the option for many, not just one. The mechanisms for information retrieval should not be "built into" the routers, but should run "on top" of the packet forwarding system. There are other reasons for this division: there will be many sorts of information with different patterns of dissemination. A system for retrieving mail has very different characteristics than a system for retrieving popular web sites, which in turn will differ from systems for retrieving high-volume objects like movies. So the network may offer key architectural support for these services, but the architecture should not define the service.

Hints: if the name (or identifier) of an object does not tell where to find it, how does one retrieve it? The answer is that there needs to be some linkage between an object (or more properly the name of the object) and the service by means of which it is to be found. For efficient operation in the normal case, the name of the retrieval service might be bound to the name, but this should be seen as a retrieval hint, not as a part of the name itself. The design of the names (and the hints) should allow for one to seek out an object in any retrieval service. Thus, one might search for a popular web page in a P2P system, in an online library or using one's social network. If one part of the network is cut off from the rest of the Internet (e.g. as a result of some disaster) one might search for an object within the locally connected region.

Unique vs. "wild-card" names: I believe that the architecture of the FI should not constrain the structure of information names (aside from suggesting some preferred features, such as how hints are attached to identifiers). However, it is useful to discuss a few features of names that will make this framework work. In some cases the user will be looking for a unique object with a unique name. In other cases the user will be looking for an object with some attribute, such as "todays' version of the paper" or "any of my mail". These sorts of retrieval requests imply names that have some sort of "wild-card" or indefinite component, which will be bound to specific objects by the retrieval service. Different retrieval services may support different sorts of indefinite naming, which implies that the conventions for how to utter requests of these sorts needs to be done using some common constraints.

(This use of indefinite names is not the same as "search", as is provided by service such as Google. Search of that sort is a layer above this set of names. Information names would be returned by that sort of search.)

10.4. Naming people

In various parts of this document, I have identified the need for one user to have some assurance as to who the other users are in an interaction. Reasons include assurance as to the sender of mail, assurance that your identity with your bank has not been stolen, assurance that you are allowed to invoke a service, and so on. For this reason, identity credentials will show up in lots of places in the FI.

10.4.1. Not just one scheme, nor one identity

The design of identity mechanisms raise many issues that have to be balanced against each other.

- The convenience of using one identity everywhere, balanced against the potential loss of privacy from cross-linking all those uses.
- The benefit of third-party assurance about identity, balanced against the additional overhead and revelation that results from using such names. Taking issues of privacy into account, there are many circumstances where a private means of identification and authentication will be preferred.

- The benefit of facilitating policing and forensics by making some identifying information visible to third parties "in the network", against the resulting loss of privacy. Too much policing will have a chilling effect on freedom of action.

For these reasons, the FI should include mechanisms to transport identity information and incorporate this information into applications and supporting services, but should *not* propose one identity mechanism. The parties to an interaction should agree among themselves as to the approach they take (if any) to verification of identity. One of the bits of information that should be returned by the SNS is the range of identity credentials that are acceptable to use any particular service. The format of the data fields used to transport identity information will become one of the constraints on the FI architecture.

10.5. Naming and deception

As the previous discussions make clear, names (and associated credentials) will be a central part of the security architecture of the FI. This means that identity mechanisms will themselves be the targets of attack. Attacks will be both technical (seeking out flaws in design and implementation) but also human (seeking to confuse and deceive the human users of the system). Part of the design of naming schemes should be mechanisms that will help prevent deception of the human users. The need to train the users with clear and consistent instincts about possible deceptions may require that we impose constraints on the design of naming schemes—if each scheme has its own different modes of presentation and validation, this will lead to usability issues. So the desired constraints on the design of identity schemes may arise as much from human factors and usability as technical preferences and performance.

11. Addressing and forwarding

In the original Internet, addresses were primarily thought of as a means to drive forwarding. In a FI, the design of an addressing scheme must take into account many issues, such as security, management, and service mapping, as well as simple forwarding. It may be the case that the design of an addressing (and routing) scheme is the last part of the design, after we have a framework for these other issues, rather than a starting point.

Definition

In this context, the term "address" is used to describe some indication in a packet (or other transmission unit) that can be used as part of forwarding to select the path that the packet takes.

- POV: While there is much interest in flat name spaces and other innovations, the process of forwarding has to be efficient enough, and the issues of scale are difficult enough, that there needs to be some form of addressing that can provide a straight-forward mapping to the low-level forwarding decision. In other words, these addresses must map efficiently to network topology.
- POV: We can afford to have address spaces large enough that addresses are not scarce, but plentiful.

11.1. Scope

As I discussed in the section on the expressive power of the packet header, I believe that debates over whether there should be one global address space or multiple address spaces is misplaced, since once the header has been defined, experience has taught us that people will use the header any way they please. However, I believe that the expectation of a global address space in which all machines are located, as envisioned for the original Internet, is neither practical nor necessary. Private address spaces will be used for a number of reasons, for example:

- Ease of management--when networks with private addresses move their attachment point to the rest of the Internet, they do not need to be renumbered.
- Security--it is harder to attack a machine that cannot be directly addressed.

While the Internet today does not expect all the end-point to have an address in its global address space, there is still a *common addressing region* (CAR). The common addressing region has many advantages over a partitioned collection of regional address spaces, and has not caused many problems. Advantages of having a CAR:

- Any service that wants to be easily reached can associate itself with an address within the CAR. It then becomes globally reachable.
- CAR addresses (one or many) can be assigned to regions with private addresses, so that the regions as a whole become globally reachable.
- POV: On the contrary, there may be value in looking at schemes that are even more general than this, in which there is no CAR, but instead regions with their own address spaces, and mapping occurs at the boundaries. Global names at a higher level can substitute for a CAR in a FI.

As discussed in section 8 on architecture design principles, the size of addressing regions (regions within which addresses have a consistent agreed meeting) will have an influence over resilience and availability. If a packet somehow loses its way due to some routing transient, there

is some hope of recovering if the packet carries a destination address that is still meaningful. If, somehow, the packet has strayed from the region in which its address is meaningful, there is probably no alternative but to discard it.

The structure of the Internet has evolved from one global address space to a space of interconnected regions. One might speculate that the resulting structure is a pragmatic balance between the benefits of larger and smaller regions. One might also speculate on design options that would improve the balance and give a better mix of results.

11.1.1. State establishment

I have argued that we should not try to define the meaning or scope of addresses as part of the architecture. Instead we should focus on mechanisms that can be used to establish per-flow state (e.g. addressing state) in intermediate nodes, since these mechanisms will enrich the options for inter-region operation. These mechanisms, because they are implemented in end-nodes, will be constraining: hard and slow to replace or modify. So if we believe that they are necessary, they should be specified as part of the design. These mechanisms, as much as the format of addresses, will limit the range of options for addressing.

An obvious example from the current Internet is that the lack of a protocol by which an end-node in a private address space (e.g. behind a firewall) can establish a binding to an address in the CAR makes it hard and unreliable to establish a passive (i.e. "listening") service behind a firewall.

State establishment protocols: While per-flow state is should be avoided or minimized when possible (for well-understood reasons of simplicity and reliability), address binding and other sorts of per-flow state will be a necessary part of a FI, in particular because of private address spaces, delegation of protection to "front line" indirection machines, and so on. Mechanisms should include tools to establish state, detect that it is no longer in place, and re-establish it. These mechanisms may include new sorts of control packets similar to what is found in ICMP today.

11.2. What is addressed?

In the original Internet, the design assumed that the element to be addressed was a physical machine--an end point of a network connection. While at the lowest level, this fact remains true, it may be that it is useful to address higher-level entities, and translate to a physical machine as late as possible--while the packet is in transit. Higher-level entities include clusters of machines, and services that may be implemented at many locations across the network. The proposed SNS would normally provide the address of a service, which would be mapped to a physical location as late in the forwarding process as possible, perhaps by using an anycast address.

More specifically, as part of the security framework described above, a first layer of defense for a service could be "outsourced" to a set of "front-line" machines that validate credentials (for services that are only for authorized users), diffuse DoS attacks, rate limit different users and throttle attacks, and so on. If these front-line machines are the ones that diffuse DoS attacks, then using anycast addresses will help with that diffusion.

(It is not strictly necessary to use anycast; if there are many such machines, even if the physical addresses of each instance becomes known, the attacker must choose between attacking all of them, which divides his attacking machines across all of them), or attacking only one or a few of them (which may overload those machines but not all of them.))

11.2.1. Anycast

Anycast is an addressing mechanism that allows a packet to be sent to a service without knowing which physical machine supports the service. Anycasting has several implications.

- Efficiency: a routing protocol internal to the addressing mechanisms can pick the copy of the service that is closest, by some metric. (Different providers may want to use different selection schemes, which would suggest a motivation for competitive alternatives for anycast resolution, not one architected alternative.)
- Security: A DDoS attack cannot be directed at a physical machine, but only at the set of machines behind the anycast address, which will tend to diffuse the attack across a larger attack surface.
- Availability: Under the reasoning offered earlier about availability, use of anycast can degrade availability. If a service is protected by an anycast address (so that the address/identity of the physical components is hidden from the client), this will make detecting and avoiding failures harder. If a client is routed to a particular instantiation of a server by its anycast address, and it malfunctions, there is no way for the client to move to another one, since by intention the client does not have that level of control over server selection. Nor does the client have a way to name the failed component, unless we add to the protocol a new name (presumably not useful for routing and thus not useful for attack) that can be used to report a failure. In the context of reporting failures, the only option is for the client to report (by some means) that "some part" of the service has failed.
- Scale: In most cases, individual services will not want to use anycast addresses, because each anycast address adds an entry to the CAR routing table, unless some scheme is devised to prevent this. More likely is that blocks of anycast addresses would be allocated to services (indirection, authorization, load balancing or protection services) that serve a number of end-customers.
- State: These intermediate nodes will often require some state in order to carry out their tasks. They might need data to perform authentication and authorization, for example. They might require per-flow state to authenticate subsequent packets in that flow, which in turn raises the issue of how to maintain that state (is it soft state?), and what to do if it is lost, or the anycast routing computation changes so that subsequent packets arrive at a different physical service point.
- Design features: a suitable anycast scheme might include:
    - A way to bundle the state and send it back to the sender so it can be "soft-instantiated" if it is lost (or the sender is redirected.
    - A "persistent anycast", where the first interaction sends back a handle that the routing system can understand, but which does not form the basis for a DDoS attack. (RESEARCH PROBLEM.)
    - A service recovery scheme where the sender can signal "give me another one different from the other one" to route around a failed server. This situation cannot exploit "explicit route selection", because that would reveal too much about the anycast architecture.
    - Competitive (virtualized?) anycast routing computations.

In today's Internet, anycast addresses are syntactically identical to normal physical IP addresses. But this choice is due only to the expressive power of the current header. There is no reason why anycast addresses (or other sorts of "virtual" addresses) need to have the same format

as physical addresses, and the packet header of a FI could allow a number of address formats to co-exist, as long as they can be efficiently distinguished.

Options for service naming and dynamic binding: in the discussion of naming, I listed reasons why the binding from service name to address of that service is one-to-many and dynamic, including selection of a instance that is "close to" the user, and load-balancing. This dynamics can be implemented in two different parts of the system, within the SNS, or within the addressing system. In turn, dynamics can be implemented in different parts of the addressing system. As discussed above, I propose a "two-stage" protection framework, in which there are "front-line" machines that perform some security checks, and then pass the communication on to the servers that perform the actual service function. In this framework, there is actually need for two different sorts of dynamics, binding the anycast address to a suitable "front-line" machine, and then passing the connection from that machine to the best instance of the actual service (more like load balancing). This framework suggests that doing load-balancing as part of SNS is not useful—it gets load-balancing "in front of" security, and reveals the physical addresses of the actual service instances to the potential attacker. In fact, different "front-line protection services" may distinguish themselves based on their ability to offer related services such as load-balancing, tracking of which server instances are up or down, and so on.

11.3. Multi-homing

Multi-homing--the attaching of a machine (or set of machines) to the network by multiple paths, should be a critical part of a FI. In general, there seem to be only a few ways to manage multi-homing.

- Do delivery based on location-independent or flat addresses. This idea is of interest today, but scales poorly.
- Do delivery based on names that are flat within some region (e.g. a geographic region), so that flat routing tables need not be global. (This is the way the telephone system covers number portability). This approach may limit some forms of multi-homing, but may have promise.
- Assign multiple addresses to the multi-homed entity, and design the FI to deal with multiple addresses as a basic capability.

The drawback of the first two schemes is that they deprive the multi-homed machine with any control over which path is used is used for access. This interferes with the economics of the relationship between provider and subscriber, and prevents the explicit selection of paths as a part of enhancing availability. For this reason, I prefer the approach of assigning multiple addresses and adding tools to manage these addresses automatically.

Multi-homing will occur not just for single machines, but for networks of machines. So the choice among techniques (see the above list) must take into account that a whole network rather than a single machine is multi-homed.

There is a strong connection between the approach to deal with multi-homing and management. If a machine (or network) can deal with multiple addresses, and addresses that change over time, then the problem of manually renumbering machines with new addresses can be avoided.

Multiple addresses: A FI should be designed so that all machines (and regions) have multiple addresses, and can manage the use of multiple addresses at the local and remote ends of connections as a normal part of operation. The lack of tools to manage multiple addresses on an end-node is a significant and unnecessary limiting constraint on today's Internet.

11.4. Ports

In the current Internet, port numbers are used to distinguish among different services running on the same physical machine, and to allow the efficient dispatching of an incoming packet to the correct process/service. In this respect, ports are a logical extension of the physical address. In the current Internet, ports are not returned as part of the DNS request (except in the SRV record), but are statically assigned to services—"well-known" ports. Well-known ports are not a fundamental requirement, but just a shortcut that make certain probing attacks easier, and which make it harder to implement services that "listen" in a private address space (e.g. behind a NAT device.

No well-known ports: in a FI, port numbers should be returned as part of the SNS lookup, and not assigned using a static table of well-known port numbers. This decision will cause a number of small shifts in the landscape of tussles. Network monitors will no longer be able to determine what application is being used by looking at well-known port numbers. Firewalls will no longer be able to block or open up ports as a way to block or permit specific applications to be used by all machines behind them (although there are ways to mitigate this management issue). On the other hand, port scanning will be much more difficult (and can be made much more obvious if the ports space is larger, say 32 bits.)

12. Routing

[This section incomplete in this version.]

Routing describes those computations that are done in the background so that forwarding can be efficiently done when packets are sent. (In the case of virtual circuit networks that set up paths before sending, routing may be the tool that allows efficient setup of VCs.)

As with addressing, in a FI routing is not just about efficient forwarding, but will be an essential part of the architecture for security, management, cross-region interconnection and so on.

Background

Today's routing schemes are not adequate for the needs of the current Internet

Within an ISP, routes today are picked based on a connectivity graph by assigning "costs" to the links in the graphs. This allows for route construction based on some sort of single-dimensional optimization, but does not allow for additional considerations such as load balancing. So routing today is combined with a second tool called traffic engineering, which starts with the real connectivity graph and produces virtual links with costs, on top of which the routing algorithm runs.

Across ISPs, BGP finds routes based on bilateral parameters and AS pathlength. This is not adequate to express or implement the desired range of business relationships. There is no way to express these relationships directly, so they expressed indirectly, using these bilateral parameters and (again) link weights on the routing algorithm internal to the connected ISPs.

The present Internet uses a very simple routing framework, that is based on single-path routes and no load-based dynamics. There are wide-ranging opinions about how to design routing for a FI, which bring into question essentially all of the design decisions of the present Internet.

- POV: One view is that we should compute multiple routes and let the user select among them, or let the user compose preferred routes using a lower-level connectivity graph. Reasons for this POV include creating a competitive market for forwarding, and constructing routes with special QoS.
- POV: Another view is that routing should be in some degree random. Random dispersion of traffic across a large number of feasible paths can eliminate the need for traffic engineering (mentioned above) and eliminate certain sorts of DoS attacks on a link.
- POV: Another view is that by defining a new cost metric, we can find single-path routes that deal with some of the issues discussed here, including security attacks and dynamic congestion.

While the Internet today computes a single set of routes that are intended to provide reasonable support for a range of services, the work on overlay networks suggests that it may be useful to have different routing algorithms for different needs.

- POV: One view is that what we today call routers should just do forwarding, and there should be multiple route computation services that compete to offer the best routes for particular needs. The Internet should not support a single routing algorithm as part of its architecture (except perhaps for a low-level "boot" routing to deal with system initialization and fault recovery).

- POV: To facilitate this point of view, and for a variety of other reasons, there is interest in moving the route computation algorithm out of the routers, and off the data path, and into a somewhat centralized route computation service. This approach raises a number of very interesting challenges having to do with resilience, response to failures, boot-strapping, etc.

13. Transport protocols

[This section incomplete in this version.]

The default transport protocol in the Internet today is TCP. While in many respects, TCP works very well, the previous discussions have indicated a number of ways in which a FI will require a different set of features in its transport protocol.

Initial setup: The proposed design calls for the ability to authenticate and authorize a communication as early in the interaction as possible. A FI transport should be able to do a "one-packet" verification so that no multi-packet state need be maintained. This will imply some limitations, so that credentials are defined to fit into one packet and no multi-packet challenge-response is required, at least for the first tier of validation.

Intermediate state re-establishment: The set of considerations proposed here suggest more dependence on intermediate nodes, which perform functions such as per-packet authorization. If these nodes contain soft state that can be lost during the course of a communication, the transport protocol will have to be able to transition out of an "established" state and back into some sort of "authorization" or "state establishment" state, which was not a design consideration for TCP.

Connection handoff: If security services such as authorization are performed by an intermediate node, then once the action is completed, the connection will have to be handed off from the intermediate node to the final node. The transport protocol should be designed so that one end of the connection can be handed off during a communication session without disruption.

Network service query: Previous discussions have suggested that a application layer should be able to query the network to determine the sort of transport features that will be available, as opposed to the current "try it and see" model of the Internet.

Explicit API: The network today does not standardize the API to the transport layer, but the socket interface is the default standard. This API mimics the semantics of TCP, with interactive exchange of byte streams. Since I have proposed that many applications may want to use a higher-level and more general service model that includes staged delivery, the FI should include a specific proposal for the preferred service API to be used by applications to obtain network (transport) services. The linkage between the service provided by the interactive transport protocol and the application service invocation model should be decoupled.

13.1. DTNs

The previous discussion focused on an interactive protocol perhaps similar in many ways to TCP. However, I have proposed that a DTN service may be a preferred mode of operation for many applications, since it will cover a wider range of network conditions. All of the considerations in this document (e.g. security, economics, management and the like) must be applied to the design of a DTN service, in the same way as to an interactive service that is TCP-like.

Division of responsibility: It is possible that if a DTN service is built "on top of" an interactive transport service, that some of the issues can be localized to the transport. This point of view, however, seems speculative to me at this point.

Variation of service: As discussed below in the section on application design patterns, many applications will want to vary their patterns of communication in response to the context in which they find themselves operating. The intersection of that objective and the use of a DTN delivery mode seems poorly explored at this point. The DTN service itself may well need to offer a range of operating modes, either by incorporating them internally or through the deployment of competing DTN services that offer different feature sets.

14. Packet header design

[This section very incomplete.]

The discussion on architecture design principles pointed out that the design of the packet header plays a strong role in the overall expressive power of the architecture. In general, this is so because the information in the packet defines the range of "input data" that can be provided to the devices that can be found in the network along the path from the sender to the receiver. These devices are not just "forwarders"; they can execute a wide range of Per Hop Behaviors (PHBs) that are devised and deployed in the network. As well, the format of information in the packet will define the range of addressing options that can be included in the header (e.g. by the source) to arrange PHBs in a particular order.

The discussion of state establishment within the network also suggested that it may be worth having more than one sort of packet header—one an efficient format that presumes pre-established state, and the other a more expansive and expressive format used to establish state.

Intentional delivery: IP header options are a failed attempt to increase the expressive power of the packet header. They seem to have failed because they cannot be processed efficiently by the forwarding engine in the router, and because each such forwarding engine must parse them to see whether anything is relevant. Based on that observation, any such mechanism should be designed to be part of what I called *intentional* delivery: that is, if some additional information is included in a packet as input to some PHB, the destination address in the packet should be the address of the machine hosting the PHB. The design of the additional information should not require the inspection of that information "in passing" by all the forwarders along the path.

14.1.1. DTNs

As noted in the previous discussion about transport protocols, all the analysis at the packet level should be repeated at the level of any DTN service. The expressive power of the delivery unit (bundle or the like) will be an important consideration in the design of the DTN service.

15. Application design patterns

In various parts of this document, I have suggested that the description of a FI must include guidance to application designer—application *design patterns*. Many applications will not be designed assuming end-to-end transport connectivity, but will instead expect to communicate via intermediate relay points that provide services, either application-specific or intended to support some class of applications.

Evidence for this assertion can be seen in the trend of application design today, and even in the past (email). Some applications, such as SIP-based services, are mixed in design, with a relayed signal channel but sometimes direct data paths. Some, like IM, most multi-player games, or Web-based interaction, are always intermediated by a server.

- POV: It is possible that all applications may work in this way in the future, and the need for universal end-to-end transport connectivity will vanish.
- POV: Given that a preferred service platform could be located anywhere in the FI, the fact that applications depend on intermediate services does not reduce the need for "universal connectivity". The ability of an end-node to establish a connection to any other willing point of its choosing is central to the balance of power in the multi-stakeholder tussle space.
- POV: If relaying is the norm, then extreme care must be taken to be sure that it is trivial to deploy a new service platform in support of any new experimental application, or we will materially harm the utility of the FI as a platform for innovation.
- POV: Given that point, it is critical that we conform to the principle of Tussle Isolation, and make sure that the industry that provides application-level service platforms is highly competitive, and not restrictive. An example of this point today is the rich space of Web hosting services. It is no longer necessary to run your own Web server to offer Web services. The rich space of providers makes it easier to host casual Web content on third-party service than to "do it yourself". The presence of this set of providers facilitates innovation in Web content, rather than inhibiting it.
- POV: The goal of application-independent universal connectivity, if we pursue it, is at odds with the possibility that different parts of the network (e.g. a region tailored to low-cost devices such as sensors) might have a different sort of native scheme for multiplexing and transport. It will be highly desirable, if such differences exist, that there is at least some way to create a connection between nodes in different such regions without the assistance and consent of the facilities-based providers.

15.1. Variation in behavior

Many of the examples proposed earlier concerning application design patterns reflect the proposition that applications will not always “work the same way” but will operate using different patterns of communication among assembled parts depending on the circumstances in which they are used. An important justification for the development of design patterns is to help the application designer recognize the range of drivers that might cause the application to modify its behavior based on context, and to built into the applications suitable responses to these drivers.

In the following sections, I collect together the various discussions of contextual drivers, and how variation in behavior properly responds to these drivers.

15.1.1. Security

Using trusted components in the network: While the general purpose end-node will almost certainly be complex enough and general enough that it will have residual security vulnerabilities, it should be possible to build more simple, fixed function devices that are more secure. These can be used to protect the insecure end-node by "outsourcing" or delegating secutity-critical tasks to these machines. Today, the firewall illustrates this approach. In the future, a wider range of such devices, some perhaps application-specific, can be used to enhance security. The use of these devices, and the pattern of such usage, may vary depending on the specifics of the particular context, including the degree of trust among the parties.

Use of encryption: If communicating parties trust each other, they may prefer to encrypt their communication to enhance confidentiality and integrity. If communicating parties do not trust each other, one or the other may insist on communication in the clear to facilitate inspection and cleansing of the data by a trusted third party. (One could imagine still encrypting the information as it crosses the public Internet, but if the third party is to decrypt it to inspect it, it may not be desirable for the encryption to be based on the private key of the recipient, because that key would then have to be delegated to the trusted third party, which seems like more trust than necessary.)

Determination of identity: In order to reason about trust, the communicating end-nodes need to have sufficient ability to identify the other parties to the communication. It does not make sense to reason about the degree of trust one should place in an unknown party. So if an application is being used in a context where there is a high degree of variation in the degree of trustworthy-ness of the other parties, some sort of scheme for identity and authentication will be required. This step may be delegated to a trusted component. I also argued that there should be an identity confirmation among the communicating parties to detect redirection attacks launched by the network.

Protection from infiltration: Means for protecting an insecure node from infiltration include determining the identity and degree of trust that should be placed in communicating parties, checking (or not) incoming data, blocking (or not) communication from unknown parties), blocking attempts to invoke non-existent services and the like.

Recognizing the intention of preferred actors: In a situation where the interests of the various actors are not aligned, it will be helpful if the application can be configured in a secure way to reflect the interests of the actors whose interests should dominate. For example, consider the situation where information is being transferred out of a machine. In one case, this may be exfiltration, or theft. The machine hosting the content may have been infiltrated, so it is no longer to be trusted as correctly representing the interests of the owner of the information. In this case, the larger context might be configured so that an "export permit" of some sort is required from a separate authorization service before the network will permit the transfer. In another case, the owner of the information may have the goal of widespread free dissemination of the information despite the attempts of censors. In this case, the larger context might be the use of signed information (to allow recipients to confirm its authenticity), and the posting of the information into p2p delivery systems and the like. The simple act of "transferring information" will call for a wide range of communication patterns, and it is important both that the designers contemplate this range of patterns, and that they arrange for the correct party (in this case presumably the owner of the information) to pick the pattern.

15.1.2. Availability

As I argued earlier, a general theory of availability requires at least these three parts: all critical components must be replicated, preferably in a heterogeneous way, it must be possible to detect when a component has failed, and it must be possible to "fail-over" to a different instance of the service. In addition, I argued that because certain (though not all) failures can be detected only by the end-nodes invoking the service, that detection and failover must be performed by the end-node as well as other components of the service. These various objectives will impose requirements on the communication patterns of applications. In particular, for applications in low-criticality circumstances, the deployment may not include replication, even though the structure supports it, so the components are in a implicit fate-sharing relationship. In high-criticality contexts, heterogeneous replication and robust failover will be essential.

Detection of failure: It may seem as if detection of failure is obvious. For simple "fail-stop" events, this is indeed so. But for more complex failures (e.g. partial or malicious), detection is complex. And if the "failure" is intentional (one node blocking a malicious action by a second node), the goal of the defending node is to prevent any sort of diagnosis or repair, since that would just facilitate the attacker. Finally, if the application does not run in interactive time but in a staged mode, even a "fail-stop" event may not be detected. Email provides a good illustration of these considerations. If one mail transfer agent (MTA) attempts to connect to another and finds that it does not respond, there are responses designed into the mail architecture, including queueing mail for later delivery, and using MX records to failover to a different MTA. But if an MTA receives a piece of mail, acknowledges that it received it, and then drops it, this sort of failure is not detected. There is no end-to-end or multi-hop error check that can detect such a failure. If the drop is because the mail is deemed to be spam, it is the intention of the node doing the dropping that the event occur "silently". There is no signal back to the sender. So the design of mechanisms for detecting failures must take into account a number of issues, including the adverse interests of the various components that make up the service.

Failover: Failover requires at a minimum that the node detecting the failure know about alternative instances of the service. This information could be obtained from a Service Naming System, just as they can be obtained today from MX or SRV DNS records. Again, there is a security issue, as discussed in the section on addressing. If the service being invoked, to protect itself from attack, positions itself behind an anycast address so that the instances cannot be distinguished, then a different sort of failover must be designed. Some sort of implicit "give me another one different from the other one" is all that will work, since the invoker has been blocked, by intention, from having explicit names for the different instances. So there may be more than one failover pattern within an application.

15.1.3. Performance

We see in the Internet of today may examples of variation in communication patterns as a way to enhance performance, mostly related to replication or caching of content near the consumer.

Measurement of network service parameters: The original vision of Internet applications was a two-party end-to-end communication. In this context (where the location of the end-points is more or less fixed), the application was expected to deal with variation in end-to-end performance by adapting to what was encountered at the moment. However, if the application has replicated service components that are "in the application", the application may wish to pick among the replicas based on observed performance. (For example, a CDN may wish to pick among possible content servers based on throughput to the intended end node.) Under these circumstances, it is necessary for the application to have some pre-computed estimates of network performance to allow efficient selection. Today, this is done by having the application itself

gather this information in the background. In a FI, it may be desirable to have the network itself gather and make available some baseline performance metrics to relieve each application from the burden of measuring these separately. This objective will require the definition of these baseline metrics, and a new API to retrieve them.

15.1.4. Management

There are very few applications today that are built out of parts operated by distinct, independent actors. Email is a notable exception, with MTAs operated by most ISPs. Peer-to-peer systems are another exception, with their extreme degree of decentralization to end-nodes. The email system operates with a very low level of cross-domain coordination and management. Most mail agents are independently and statically deployed by the various operators, and there are few mechanisms (aside from MX records, as discussed above) to deal with failures, application-level dynamics and the like. Modern P2P systems, in contrast, integrate into their design highly complex schemes to deal with nodes that come and go, malicious nodes, dynamics of network and node performance, and the like. Most of this is automated, so that the application, taken as a whole, runs without much human intervention. Interestingly, at the present time, we do not see important examples of more "managed" applications (e.g. ones that might be deployed by commercial operators) that take advantage of this sort of automated management and control. If the goal of an application is to facilitate the easy deployment of parts by different actors, we should look for management patterns that perhaps resemble what modern P2P systems do, where there is perhaps as much attention to the processes of configuration and availability as to the actual data transfer.

Just as the early Internet designers gave perhaps insufficient attention to the "network-network interface", where ISPs and other transport providers interconnect, modern application designers may want to think about the "provider-provider interface", where the service elements provided by different actors interact.

15.1.5. Economics

The internal structure of applications—their way of deploying and using service components—is often a reflection of the economic motivation of the designer/operator. Many commercial applications often have a critical component that is conceptually centralized (while it may be physically replicated and distributed, the operator maintains exclusive control over it and access to it). This component allows the operator to maintain control over the application, and often to reap financial benefit from it. Many multi-player computer games and virtual worlds, web applications such as Facebook or eBay, and many instant message systems and the like look like this. (Of course, this "central" component plays other roles, such as imposing the desired behavior on participants with adverse interests, such as cheaters in multi-player games.) On the other hand, services are sometimes highly distributed (for example Content Delivery Networks), both for performance reasons and to attempt to dominate a market. For any given application, the designer must ask whether one of these patterns or both of them should be an option within the design.

15.1.6. Social and regulatory considerations

Application designers must take account of a very significant tension as they consider social and regulatory issues. On the one hand, application designers may be motivated to encourage the global uptake of their application by allowing variation of how it operates in different cultures and regimes. They may choose to avoid "baking in" specific norms such as a particular preference for privacy or open access. On the other hand, the designers may choose to do the opposite, and identify behaviors that are deemed inappropriate in most contexts (perhaps "spam-like" behavior) and design them out to the best of their ability. The balance between a

sensitivity to variation in local norms (in which case the application will work differently in different contexts) and a desire to impose fixed behaviors is a critical set of decisions.

15.1.7. Network service model

The network service model on which most applications depends today is the packet-level interactive exchange of bytes provided by TCP. However, in the future a staged or DTN model may prove more general and equally applicable. Application designers will have to decide whether to design applications that can adapt to a high degree of variation in end-to-end latency (ms. to minutes) or to only operate in part of this range.

API for application transport

The design of the Internet today does not formally standardize the API to its default transport service (TCP). In fact, the socket interface is almost universally seen as the application API, but this fact is not anywhere in the Internet standards. Since this informal interface directly mimics the semantics of TCP, this leads the application designer to assume that the application should be designed to expect this service. In fact, many applications could work over a much more general interface, which could be mapped both to a interactive stream and to a relayed message service. If the architecture of a FI explicitly included the specification of a general higher-level service interface, this would guide application designers to think about how to design applications that are more tolerant of service diversity than they are today.

15.1.8. Summary

These patterns transcend specific application objectives. They apply to broad classes of applications. Many of these considerations are complex,and most application designers will benefit from guidance as to how to think about and implement them. This guidance is what I have been calling design patterns.

In many cases, the factor that will determine which pattern is preferred is assessment of which actors are most trustworthy. So management of trust, and by extension identity, must be native facilities. In many cases, it will be beneficial if trust assumptions can span applications. So tools for management of identity and trust should not be designed to be “inside” individual applications, but should be designed as a common service. On the other hand, there should not be just one scheme for identity and one scheme for developing and tracking trust. There should be many schemes, and the selection among time will be part of the variation that the application design must encompass.

16. Summary of architecture components.