FINAL TECHNICAL REPORT

New Arch:
Future Generation Internet Architecture

David Clark, Karen Sollins, John Wroclawski,
Dina Katabi, Joanna Kulik, Xiaowei Yang
MIT Computer Science & Artificial Intelligence Lab

Robert Braden, Ted Faber, Aaron Falk, Venkata Pingali
USC Information Sciences Institute

Mark Handley
University College London

Noel Chiappa
Independent Consultant

6/30/00-12/31/03

Sponsored by
Defense Advanced Research Projects Agency (DoD)
Information Technology Office (ITO)
Under Grants # ***F30602-00-2-0553*** (MIT) and ***F30602-00-1-0540*** (ISI)
Issued by the Air Force Research Laboratory
Rome, NY

(This page contained Army distribution form, omitted here.)

Table of Contents

Future Generation Internet Architecture

EXECUTIVE SUMMARY

This document is the final report of a collaborative research project to explore the architectural fundamentals of the Internet. At the MIT Laboratory for Computer Science, work was carried out under DARPA grant F30602-00-2-0553, administered by Air Force Research Laboratory/IFKF, Rome N.Y. At USC Information Sciences Institute, work was carried out under DARPA grant F30602-00-1-0540, administered by Air Force Research Laboratory/IFKF, Rome NY. Other, unfunded, participants in the project included the UC Berkeley ICSI Center for Internet Research (Mark Handley), and an independent contributor (Noel Chiappa).

This report is a combined final report on the work done at MIT and at ISI, since the effort was highly collaborative.

Charter

The goal of this project was to consider the following question: if we could now design the Internet from scratch, knowing what we know today, how would we make the basic design decisions? The phrase "design from scratch" implies that we deliberately set aside issues of migration—how we would get from where we are today to a new design tomorrow. We wanted to free ourselves, to the extent possible, from the pressures to think incrementally. Our philosophy in this respect is the following: if we set a long-range vision, short-range implications often reveal themselves as a result, but if we start out thinking about short-term considerations, we will never set a consistent long-range direction.

Our view is that most progress in networking today is shaped by short-term thinking. Many small changes are being proposed to the Internet to fix specific short-term problems, but there does not seem to be an overarching vision of what the future network should be. The various short-term changes being proposed and implemented today may be at odds with each other and may push the network in inconsistent directions as we move into the future. We fear that as these changes accumulate and the inconsistencies and tensions become more pronounced over time, further progress will become difficult, and eventually impossible. It is this unwelcome outcome that we hope to mitigate by setting some long-range directions.

Plan of work

Our research plan was as follows.

- Review of current requirements—what has changed, what did we miss the first time?
- Review of architectural principles of the original Internet in light of our present knowledge.
- Identification of key architectural questions for consideration, and assessment of how they interact.
- Detailed exploration of specific architectural issues.
- Recommendations for further work.

Following our review of key architectural questions, we identified areas where specific in-depth effort was justified, and we carried out specific projects in these areas. These projects are briefly summarized below, with pointers to more detailed descriptions later in this report and published elsewhere.

Requirements—what has changed, what did we miss?

The original design of the Internet emerged almost 30 years ago[1], a time that predates both the personal computer and local area networks. Fiber optics had yet to emerge from the lab, the fastest cross-country circuit commercially available was about 50 kb/s, text was the only common user interface modality, and computing had not begun to reach the masses. The communications industry was defined by the role of AT&T as a regulated monopoly provider of telephone service. Clearly, a lot has changed since then, in terms of technology, the computing context in which the Internet sits, and the broader societal impact that computing and networking now have.

A technologist might want to consider the vast changes in the shape of network technology. We now have links that are almost a million times faster, wireless communication in several forms, personal computers on every desk, and a coming wave of embedded and pervasive computing. These are powerful and transforming technologies. However, our deliberations led to the conclusion that the contextual changes surrounding the Internet are perhaps more important—the set of commercial, societal and policy considerations that now shape the future of the Internet.

This Executive Summary contains a summary of the high-level issues we identified in our review of requirements. A much more detailed list of requirements will be found in Appendix A. While we identified and considered many detailed requirements in the course of our project, most of the compelling changes from the origins of the Internet that we see today can be cataloged under one of the following major headings.

The Internet as a commercial undertaking

When the Internet was first conceived, communication was provided by "the telephone company", AT&T. There was no clear model of either a market for data services or how an industry might emerge to meet that demand. The Internet Service Provider, or ISP, really emerged only in the 1990s as a commercial response to the success of the government-sponsored NSFnet of the 1980s.

The emergence of the ISP as a commercial offering illustrates a basic architectural point: *the structure of the industry is induced by the network architecture, in particular by the points at which open interfaces are specified.* This point was understood, if imperfectly, in the 1970s and 1980s. The original requirement for open protocols among routers was driven by the desire to create a competitive market in routers, and the design of the Border Gateway Protocol (BGP) in the 1980s was driven by the desire to create a competitive market in wide-area Internet service providers. However, the full economic implications of the Internet architecture were not appreciated in the first design.

There are two key factors that emerge from a consideration of the commercial Internet today. The first is that the forces on the wide-area service providers are economically stressful. The market, by design, is highly competitive. The second is that open architecture assigns to the ISP a simple packet-carriage business, which can become a commodity product. And the industry must make large, up-front investments in fiber, right-of-ways, central offices, and so on, before offering services—a "sunk cost" structure. The combination of these factors can lead to economic disaster. In a competitive, commodity business, prices are driven toward incremental cost, and the incremental cost of delivering packets is zero. But if prices drift toward zero, there is no way to pay off the sunk costs. This leads to bankruptcy, mergers, erosion of competition, and low rates of investment, all of which have been seen in practice over the last few years.

---

[1] See V. Cerf, R. Kahn, *A Protocol for Packet Network Interconnection*, IEEE Trans. Comm., COM-22, No. 5, May 1974, pp. 637-648, and V. Cerf, P. Kirstein*, Issues in Packet Network Interconnection*, Proc. IEEE, v.66, 11, Nov. 1978l. The original Internet architecture was recorded 10 years after the fact in: D. Clark, *The Design Philosophy of the DARPA Internet Protocols*, Proc ACM SIGCOMM 88, Sept. 1988.

Commoditization could play itself out in two different and undesirable ways, but architecture can provide some resistance to either of these paths. One path would lead to ISP monopoly. If one ISP can gather enough of the market, it is in that ISP's interest to commoditize the market. This allows the ISP to begin using the other market forces (high barriers to entry and economies of scale) to capture the whole market by non-technical, monopolistic techniques. For example, an ISP with enough market share could force other ISPs out of the market using below-cost pricing in competitive areas subsidized by increased pricing outside those areas. By encouraging innovation in a way that promotes user goals, an architecture can resist the growth of a monopoly ISP (see Section 3.3.)

On the other hand, ISPs, to avoid the economic trap of monopoly or bankruptcy, will want to avoid becoming providers of a pure, commodity packet carriage service. They will want to provide higher-level, user-visible services that offer the possibility of product differentiation and higher profits. But this motivation will lead them to implement services in the part of the network they control, the "center", by introducing so-called "middleboxes." Such an approach is in direct opposition to one of the core design principles of the Internet, the end-to-end arguments (see Sections 2.1.4 and 2.4.) So ISPs, in their drive to remain profitable, are motivated to violate the most basic of the Internet's architectural principles.

The high-level architectural implications of commercialization can be summarized as follows:

The modularity of the architecture can induce different industry structures, some of which are more likely to lead to vigorous investment and innovation in the core of the network (Section 2.1.8). Design should favor this outcome, so long as it does not inhibit the vigorous innovation at the edge of the net that has defined its past advances.

To resist the trend toward commoditization, ISP's will be motivated to move function "into" the network. Thoughtful design may provide opportunities for ISPs to offer services that are less disruptive to the architecture than today's trends (Section 4.2).

The erosion of trust and the need for security

The original design of the Internet has been described as *transparent*: what goes in comes out. The net does not observe, filter, or transform the data it carries; it is *oblivious* to the content of packets. This transparency may have been the single most important factor in the success of the Internet, because transparency makes it possible to deploy a new application without having to change the core of the network. On the other hand, transparency also facilitates the delivery of security attacks, viruses, and other unwelcome data. When the network was small and there was a high degree of trust and shared context among the users, the power of transparency outweighed its risk. Today, with a world of users that mistrust each other, devices such as firewalls are deployed with the explicit goal of disrupting transparency by blocking unknown traffic. And today we see both the benefit and the peril of devices such as firewalls—it is becoming much more difficult to deploy new applications on the Internet, because firewalls may block them.

We concluded that the service of the Internet should be more adjustable than what we have today: transparent among users that choose to trust each other, but highly constrained among users that do not. *This trust-modulated transparency should be the basic delivery service of a new Internet.* (See Section 4.1.4)

The emergence of conflicting interests

When the Internet was first conceived, it was assumed that most users had interests that were aligned. Various groups of users would want to communicate, and would do so, based on mutual desire. But today, we see that the stakeholders in the Internet space often have interests that are adverse or conflicting. Users want to have a private conversation, while law enforcement wants the ability to carry out lawful intercept. Users want protection from spam; spammers try to circumvent their

protections. Conspiring users share music; the rights-holders try to prevent this. The tussle among these various interests occurs in many ways: in debate over protocol design, in law and regulation, in technical choices made by different groups of users and providers, and so on. These conflicts, while perhaps intrinsic to the nature of society, can impair the operation and utility of the Internet, slow innovation, and prevent the achievement of what many would take as essential goals, such as increasing the security of the net.

"Design for tussle" must become an overarching objective of all architectural decisions we make for a next generation Internet. We must recognize and catalog techniques for designing for tussle, and make these techniques a basic part of the designer's toolkit. (See Section 2.4).

The emerging broad base of users

The original users of the Internet were members of the network research community. The pool of users has expanded—from network researcher to CS researcher to scholarly user to a broad cross section of worldwide society. The original user was technically sophisticated and trained in the operation of the net. Today's user is not technical, and wants to view the network as a utility, like the telephone system.

The so-called "transparency" of the Internet has the consequence that it pushes functions (e.g. application software) from the core of the network out to the edge. This works to the advantage of sophisticated users, because they can then modify these functions easily. But for naïve users, pushing the function to the edge means that these users must manage, configure, and debug functions about which they know nothing. On the other hand, allowing these functions to migrate into the core of the network might stifle innovation and freeze the network in its current state.

A critical requirement for an Internet of tomorrow is that it balances the goals of being open to innovation and novel uses, while not demanding technical virtuosity in its users.

New application requirements

The original Internet provided a very simple and minimally specified packet transfer service, sometimes called "best effort". Crudely, what "best effort" means is that the network makes no specific commitments about transfer characteristics, such as speed, delays, jitter, or loss. It was assumed that end-system software, both transport layer protocols and applications, would (and must) take this unpredictability into account.

The best-effort paradigm was very powerful—it meant that a wide range of communication technologies could be incorporated into the Internet, technologies with a wide range of basic characteristics. One of the factors that made the Internet protocols a success was that they could work over "anything."

However, as the Internet has "grown up" and become a commercial product with paying users, there is increasing pressure to provide a more uniform experience and more control over operating parameters, to support the more demanding applications such as real-time multi-media streaming. There is thus an emerging tension between two design goals— preserving the ability of the Internet to operate over a wide range of technologies under a wide range of operating conditions, while also providing a predictable and uniform set of operating conditions for the applications that run on top of it.

A next generation Internet must take a more sophisticated view of how to manage this tradeoff, compared to the simple original "best effort" service. Such network tools as Quality of Service, routing based on Quality of Service, and visibility into network topology and operating parameters, must be married to more sophisticated methods for application designs, including moving parts of the application function from place to place in the network to deal with variation in performance.

At the extremes, it becomes impossible to reconcile these two pressures. Consider, for example, an inter-planetary network with latencies measured in hours or days and no real-time end-to-end services. *It is necessary to limit the range of operating circumstances that we choose to accommodate in a real-world Internet if we also want to make any commitments to the application designers.*

New technology

Since the Internet protocols were first proposed, a succession of new communication technologies has emerged: high-speed local area networks, then new wide-area technologies such as Asynchronous Transfer Mode (ATM), and now MPLS and wavelength division switching. In each case, there has been the claim that these technologies would be both so compelling and so distinctive that the Internet would have to be redesigned to accommodate them. In general, this has proved false. In fact, it is remarkable how robust the central design principles of the Internet have been, as they have accommodated new generations of technology.

We concluded that *wireless* is the one technology that does challenge the original Internet architecture. Some of the technical features of wireless (e.g., the higher loss rates under some situations) raise specific issues (e.g., congestion control), but these are not the central concerns. Mobility itself—the movement of a node (or a sub-net) within the larger Internet—is the key requirement that the original architecture does not handle well.

Mobility raises issues of addressing, routing, security, variation in service, and so on, which the network architecture must recognize and deal with. Furthermore, mobility exacerbates the erosion of trust.

Architecture then and now

One of the stages of this project was to identify and catalog the key architectural features of the original Internet, and to review the validity of those features in the light of the changing requirements listed above. Appendix B of this report contains a detailed listing of features and a discussion of how we would change them in today's world. This section is a brief summary of the more important architectural points.

Packets

The original Internet used the *variable-size datagram packet* as a uniform data representation in all parts of the network. There have been calls to abandon packets in favor of something else,[2] but we concluded that the basic idea of the packet is a robust idea that has stood the test of time. We recognize that our conclusion may seem conservative, but we concluded that packets provide a general solution to the problems of statistical multiplexing, fine-grained congestion control and quality of service, and support for a wide range of applications.

However, we concluded that the packet abstraction should be rethought to meet today's needs.

- *There is a need for an architectural abstraction to deal with aggregates of packets.* The Internet actually displays two different modes of behavior. Near the "edge" of the network, traffic patterns are dominated by the bursty and unpredictable patterns of a small number of sources and sinks. In the "middle" of the network, traffic patterns are defined by the statistics of aggregation, where large numbers of flows combine to produce overall patterns very different from those at the edge,

[2] See, for example, Ilia Baldine Harry G. Perros George N. Rouskas Daniel S. Stevenson, *JumpStart: A Just-in-Time Signaling Architecture for WDM Burst-Switched Networks,* Proceedings of the Second International IFIP-TC6 Networking Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; and Mobile and Wireless Communications, Springer-Verlag, 2002 London, UK

with fewer short-term fluctuations in traffic. Lower level technology such as MPLS has been introduced to carry these aggregated packet flows. We concluded that the Internet architecture itself should recognize aggregates of packet flows as a distinct architectural entity.

- *The pure datagram model may be too simple.* In the original Internet, each packet is fully self-defining. It carries a full source and destination address, and it can be forwarded without any setup or signaling. This mode of operation must not be lost, lest we lose the ability to exchange packets with little overhead. However, the goals of trust-modulated transparency and other aspects of security, together with the goal of providing quality of service and other aspects of service enhancement, suggest that under some circumstances there will be more state in the network related to specific packet flows. *The Internet should be able to shift smoothly between different delivery modes with different quantities of control state in the net.*

Addressing and identity

The original Internet presumed a single, global address space. The addresses serve two purposes—they provide both an indication of the *location* of the end point, and an indication of its *identity.* We concluded (as have many others) that these assumptions must be rethought. The design of a new scheme for location and identity is a critical architectural requirement to address issues of security, mobility, routing, and regional autonomy. For this reason, we undertook a detailed exploration of this aspect of architecture, as described in Section 3.1. We generated a high-level design that separates location from identity, and then derived a more elaborated version of this scheme. We argue that it is possible to separate the ideas of location and identity, both of which are represented by the IP address in today's Internet, and that the resulting architecture facilitates mobility as well as solving other problems with today's network.

For reasons of management and security, the Internet has abandoned the single, global address space, although there has been no formal recognition of this in the architecture. We concluded that there is in fact no fundamental need for such a global address space, although the consequence of avoiding it is more complexity in application session establishment and less homogeneity in network management and fault diagnosis. However, a network without a single global address space is feasible, and it meets practical needs. *A future Internet should be designed without the requirement of a global address space;* see Section 3.1.2.

Internet security

Better security is perhaps the most pressing requirement for an Internet of tomorrow. Our initial work in this area was to articulate the relationship between trust, identity, and transparency as a step to better security. This work is described in Section 4.1.

Our conclusion from this stage of the research is that one of the issues in improving Internet security is that we lack a clear articulation of "the security problem". In order to fix a problem, one must first be able to state the problem. This conclusion led us to a search for contextual framing of security—what are the set of forces that drive or impede better security. Our conclusion is that *security must not be viewed as a single-dimensional space in which more is better, but a multi-dimensional space that is shaped by powerful value conflicts*. In other words, the search for better security has been impeded by the community's past failure to recognize the emergence of conflicting interests, discussed above.

Expansion on this idea was not completed within the period of DARPA funding. We are continuing this work and seeking additional funding to support the effort.

Application architecture

The original designers of the Internet focused on the core function—the forwarding of packets among end nodes. There was less attention given to the proper design of applications. To some extent, it was assumed that application would just happen as needed after the network was done, and it would be obvious to application designers how to use the facilities of the network to build their functions.

We concluded that much more attention needs to be given to proper application design. Issues such as naming, scale, manageability, and security must be considered in application design. Since these issues are common across many applications, it is wasteful for every application designer to work them out afresh. Section 4.2 collects a number of design considerations for applications, drawing on the various deliberations within this project. Our goal will be to continue to refine this document, using additional funding as available, and publish this as a guide for the Internet community.

User empowerment

We described earlier the need to "design for tussle". One approach that we explored is to *empower* users by allowing them to choose the complete provider path to be taken by their packets. This could improve the vigor of competition among providers and should stimulate the deployment of advanced network services. At the same time, if done wrong, user empowerment might increase the complexity that the end-user must deal with, it might raise new forms of failure, and it might make packet forwarding less efficient.

We took the position that, given the reality of the commercial Internet, a complete replacement for the BGP inter-domain routing protocol is required to gain such user empowerment. We developed a new routing scheme, called NIRA (New Internet Routing Architecture) for this purpose; NIRA is discussed in Section 3.3. The NIRA approach has the additional advantage that it reduces the information that has to be propagated globally, so it should scale much better than BGP.

Weak semantics

The transport semantics of the Internet are quite vaguely defined: best effort throughput, pretty good delivery latency, mostly preservation of packet contents, and so on. This weak specification has led to a tolerance and flexibility critical to permitting operation over diverse technologies and diverse implementations. As the Internet matures, the primacy of weak semantics is being challenged by two forces: the phenomenon of the least common denominator, and an increased emphasis on security.

The phenomenon of the least common denominator is easily understood by observing that in any sufficiently large system, variability tends eventually to be driven out. Alternatives that do not provide significant benefit or that incur high costs will become less common than others, as a result of which fewer portions of the system will tolerate them well, as a result of which they will become even less common. Over time, the de facto system specification becomes more precise, and less weak. This does lead over time to higher performance and greater efficiency for existing applications. However, its cost is the ever-increasing loss of the flexibility and evolvability that is a hallmark of systems with weak semantics. To counter this drive toward homogeneity, the maintenance of a healthy application development and research community serves to continually throw diversity into the mix.

A second force acting against the existence of weak semantics is increasing concern about security. As any infrastructure grows in importance to society at large, security concerns correspondingly increase in importance. One important manifestation of this is often a change in mindset from "what is not explicitly prohibited is allowed" to "what is not explicitly allowed is prohibited". Precise system semantics, by offering more direction to the developer, may lead to system implementations that are easier to develop and verify, more able to detect and defend against illegal inputs, and less likely to

have bugs or unhandled unusual cases. Thus, security concerns create a direct challenge to the weak semantics paradigm and its concomitant benefits.

An alternative approach to security with weak semantics is run-time checking to detect messages that may represent an attack. Firewalls are a "brute-force" filter, and there are more sophisticated techniques that have been proposed, such as *protocol normalizers*.[3] These approaches, however, may still interfere with innovation. Our principle of *trust-modulated transparency* is the key to preserving both security and the ability to innovate. If innovation occurs among trusting collaborators, and if they are able to disable the filters and obtain a "clear channel" by mutual consent, then they can choose from the best of both worlds.

Other new architectural directions

The project also investigated four other specific and important areas in the architecture.

1. Congestion control

We concluded that requirements for higher speeds and longer delays in the network are pushing the Van Jacobson congestion control scheme to the limits of its utility, with the result of slow adaptation to changing link loadings, poor link utilization, and significant congestion losses. We developed a new and different scheme for congestion control, called XCP (eXplicit Congestion Protocol), described in Section 3.4. A follow-on to the NewArch work is performing further testing and demonstration of XCP in an operational environment.

2. An alternative to protocol layering

Since its beginning, protocol architecture has been dominated by the familiar layered protocol model. However, the increasing complexity and rigidity created by strict layering led us to consider the possibility of protocol design without layering. We developed a preliminary approach to a non-layered architecture, which we called "role-based architecture" (RBA), which is described in Section 3.2.

3. A region-based architecture

The classic visualization of the Internet is a cloud of routers, surrounded at the edges by hosts and servers. In reality, the "cloud" of routers is composed of sub-clouds that are provisioned and operated by different administrations—ISPs, corporations and other organizations, and even home-dwellers. Many attributes are linked to this regional structure—scope of shared trust, common administration, payment for services, and so on. However, the Internet architecture provides few means to express the notion of a region. We studied options for capturing the concept of regions in a new architecture, as discussed in Section 3.5.

4. Efficient distributed event notification

To explore a facility to support next-generation applications, we undertook the design of a scalable and efficient *event notification system*, which can be used to deliver notifications to multiple users using an overlay forwarding network. Compared to prior approaches, our scheme provides better options for privacy and security, much greater efficiency as the number of subscriptions for notifications grows, and a design more consistent with the objectives of the end-to-end arguments. This work is described in Section 3.6.

[3] M. Handley, C. Kreibich and V. Paxson, "Network Intrusion Detection, Evasion, Traffic Normalization and End-to-End Protocol Semantics", *Proc. USENIX Security Symposium 2001*.

5. The Knowledge Plane[4]

We conclude that the architecture of the Internet is a direct contributor to some of the usability issues that plague naïve users. The design of the Internet pushes application function out of the net and into the end-nodes. This means that management systems in the core of the network have no visibility into what the user is trying to do, or what *ought* to be happening to achieve some user function. So if something in the net is preventing an application from operating successfully, the network cannot tell that something is wrong, because the failure manifests in the end-node.

We conclude that the correct approach is not to modify the design of the routers to move function back into the center of the net. This might allow better management of existing applications, but would destroy the "transparent data carriage" model that has been the basis of innovation and the deployment of new applications. Instead, we propose a more daring revision of the Internet architecture, which is to develop a new, highly distributed component of the network, an overlay that gathers information from the end-nodes about what *should* be happening, gathers information from the network about what *is* happening, and integrates this information in a way that can detect and correct failures, and interact with the user, in terms meaningful to the user, about the nature of the problem.

Network function is often organized into layers, or planes. At the bottom is the data plane, responsible for the actual forwarding of packets. Above this is a control plane, and then a management plane. Since we view this new capability as sitting above all these existing planes, observing and gathering information, and reasoning about what it has gathered, we named this new capability the *knowledge plane.*

At the request of DARPA, a major part of MIT's effort in the last part of this grant was the development and elaboration of the concept of the knowledge plane. DARPA is in the process of launching a program around this concept, and has hired a program manager for the purpose. The work carried out under this grant is described further in a paper presented at SigComm 2003.[5]

Summary and status

Our work did not lead to rejection of the fundamental nature of the Internet—an application-independent data carriage service based on variable length packets. We affirmed that this basic design is sound, has passed the test of time, and will serve well in the future.

We did conclude that many attributes associated with this basic packet service need to be reexamined, including the pure datagram assumption, global addressing, the linkage of location and identity, and universal transparency.

Many of the results from this project were presented at a workshop on new architecture for the Internet, held last August as a part of the SIGCOMM 2003 conference.

Acknowledgement

The participants at MIT and ISI would like to thank the Defense Advanced Research Project Agency, and in particular Mari Maida, the program manager who initiated this program, for the vision to ask a long-range question and for the confidence and enthusiasm they have shown for the work we have done. We would also like to thank our collaborators from other sites, some of who participated without explicit funding, just for the fun of it.

[4] Research on the Knowledge Plane is not reported in this joint MIT-ISI version of this final report. ISI was a collaborator on the Knowledge Plane research, but on a separate grant separate from the NewArch work.

[5] D. D. Clark, C. Partridge, J. C. Ramming and J. Wroclawski, "A Knowledge Plane for the Internet". *Proceedings of ACM SigComm 2003,* Karlruhe, Germany, August 2003

Future Generation Internet Architecture

1 Introduction

Today's Internet technology is rooted in a coherent technical design that was developed in the late 1970s under DARPA's Internet research program. The global technical principles, or *architecture*, of the Internet design represented a deliberate choice out of many design alternatives, informed by an understanding of the requirements. Roughly, network architecture is a set of high-level design principles that guides the technical design of the network, especially the engineering of its protocols and algorithms. Architecture provides coherence and consistency among the many technical decisions required to standardize the protocols and algorithms of the network. Section 2 of this report contains a deeper and more complete discussion of the meaning of "network architecture".

Current and future reality is eating away at the viability of the original Internet architecture. The goals for the Internet, and therefore the requirements on its architecture, have changed significantly since the 1970s. Much of the coherence of the original architecture is vanishing in a sometimes-conflicting patchwork of technical embellishments. If present trends continue, the Internet protocol suite seems likely to become decreasingly effective and to increasingly fail to meet the demands of military and civilian applications.

This report describes the results of a DARPA-funded research project entitled "NewArch: Future Generation Internet Architecture." The basic goal of the NewArch project was to begin formulating a new long-term technical road map for the Internet, considering current and projected realities. Put simply, the question the project contemplated was the following: "if we could design a network like the Internet today, without the need to worry about migration from the existing deployed network, what approaches would we pick?" The project participants hoped that the results might provide an improved framework to help guide the evolution of the Internet. The NewArch project was unique in its long time scale, its breadth, and its range of levels of abstraction. It looked simultaneously at the requirements for a future Internet and at a wide range of key aspects of the basic architecture, in search of unifying principles.

The success of the original DARPA Internet research program resulted from a synergy of practical experimentation with architectural reasoning. We believe that the development of a new architecture must include a combination of top-down reasoning about architectural principles with design and prototyping to explore the technical consequences. Top-down protocol design starts with a small set of abstract design principles and derives successively more concrete designs by filling in levels of technical detail, until the level of engineering of bits and bytes is reached. In fact, it is only the success of the derived technology that gives one confidence in the necessity and perhaps sufficiency of the original abstract design principles.

Designing and validating an entire new architecture would require a coordinated effort by a major research program, far exceeding any realistic goals of the NewArch project. Given its limited resources, the NewArch project had to be selective in its goals and scope; it could not realistically create a complete technical design for network protocols. The NewArch project had sufficient scope to allow detailed design and prototyping in only a few specific areas (see Sections 3.1, 3.3, 3.4, and 3.6).

There are already many individual ideas for architectural changes in the literature and under development, and the NewArch project could have concentrated on integrated the most useful of these ideas into a complete, coherent, and consistent new architecture. We chose instead to attempt a more fundamental rethinking of the issues.

Our approach included the following components:

- Examination of the changed and changing requirements.
- Examination of the failures of the original architecture.
- Consultation with experts in relevant technical areas, such as mobility, economic models, and embedded computing.
- Development of new architectural principles and meta-architectural principles (e.g., very high-level abstractions from architecture) to meet the new requirements.
- Development of proposals for specific design principles and approaches.
- Implementation of proof-of-concept prototypes when feasible.

The project carried through one example of top-down design and did (limited) prototyping of it (see Section 3.1). However, the project concentrated its limited resources more on high-level architectural abstractions rather than on detailed design or prototyping. In addition, the project chose to generally ignore backward-compatibility issues.

Section 2 of this report discusses the nature of architecture and the highest-level abstractions of the Internet architecture – modularity, interoperation, and functionality. This section is somewhat philosophical and may be skipped by those familiar with these issues. Section 3 contains short summaries of particular areas in which the project made progress and published papers. Section 4 contains other areas in which we were able to formulate a new understanding of the problem, but did not reach technical conclusions. The material in Section 4 should provide guideposts for further research. Appendix A (Section 5) gives a brief overview of the changing requirements for the architecture, while Appendix B (Section 6) provides a cross-cutting summary of the principles of the original architecture and possible modifications of these principles in a new architecture.

2 The nature of network architecture

What do we mean by architecture? In what ways does the architecture of a system describe, define, and constrain it? Specifically, when we think about an architecture for a *network*, what are the basic approaches and design principles?

Directly or indirectly, architecture defines what the network is for, and how it accomplishes its functions. At a very abstract level, the architecture of a system defines how the system is broken into parts and how those parts interact. It defines the assumptions that one part may make about another, and in doing so, imposes both constraint and freedom on the parts: constraint, in that the parts must "obey the rules", but freedom, in that parts can "do as they please with confidence" as long as those rules are obeyed.

Architecture should be explicit about what it does *not* specify, as well as what it does. It should specify only that which is necessary, and be clear where no constraints are imposed.

The fundamental objects of an architecture define that architecture in ways that are even more powerful than the specific module boundaries. By giving names to parts, architecture gives designers and others a vocabulary to talk about a system. This provides a framework (a "reference model" [ISO84]) that permits efficient conversation and modular explanation as well as modular construction.

2.1 Modularity in the architecture

2.1.1 The layered model

Modularity and abstraction are central tenets of Computer Science thinking. Modularity breaks a system into parts, normally to permit independent construction and replacement, reuse of parts, and so

on. However, selecting and specifying good module boundaries is an art. Abstraction is a refinement of modularity—it is the gathering together of some complex mechanism into a module, and providing a simple interface by which this complexity is hidden. Abstraction suggests a structure for the system. A popular and simple structure is a *layered model*: lower layer mechanisms provide interfaces to their functions, which are then used by higher-level functions. Closely related to the layered organization is the concept of "dependency", which shows how each module depends on the correct operation of others. Mutual dependency is known to be complex, and this leads to the desire for a loop-free dependency graph, which implies layers.

The OSI effort gave us perhaps the most thoroughly crafted and detailed layered reference model for networks, and it has shaped discussion and design modularity for almost two decades [ISO84]. There are other rich layered models of computer systems, as in the sixteen layer abstraction model of PSOS, a provably secure operating system [Neumann 80]. However, layering is not the only useful way to organize a system.

2.1.2 Functional slices

Certain system aspects do not seem to fit into a layered model. For example, system performance pervades all layers; there is no such thing as a "performance layer". One can try to slice the performance problem up into layers by allocating a share of the responsibility to each layer, but one must take a holistic view to do this, and that holistic view must survive as part of the architecture, so there is a "performance specification" that of necessity spans all the layers. It is a vertical functional slice laid on top of horizontal layers. Similarly, security requires a holistic design and description, and it does not admit of being corralled into layers.

So, even if the primary description of a system is in layers, there will be functional vertical slices that must also be described. Some architectural frameworks have functional slices only as descriptive elements. Some systems, such as the Aspect Programming system of Kiczales [Kiczales97], try to organize the actual code into functional crosscuts, so that a specific aspect of each module, for example "security", can be extracted and modified without rewriting the whole system.

2.1.3 The reality of topology

To the surprise of some designers, distributed systems are actually distributed: physical placement is relevant to system operation. If it were not, the whole system could be gathered into one room. However, layered and functional slices do not directly describe physical distribution and thus do not frame any architectural requirements related to actual placement. Physical distribution imposes a different sort of modularity on the system—the parts are in different places and each may contain multiple levels or modules of the architecture. Furthermore, because they are in different places, the parts may be administered by different organizations.

Details of physical distribution are usually abstracted to some extent, but aspects such as proximity, cost of communication, and the probability of independent failure may be very important. However, most designers have little experience in choosing the right degree of abstraction or the co-ordinate space in which this abstraction should occur.

There are at least two planes of distribution, the system topology ("what is connected to what, with what constraints?") and the physical location ("what is physically near what?"). They are not the same. Two devices sitting next to each other on a desk may be many hops away in network topology; for example, one may be on a local Ethernet while the other is on a global wireless network. Many architectures include some specification of system topology. For example, they may constrain the system to be a tree or define algorithms that cause a tree to appear (e.g., the spanning tree algorithm for bridged Ethernets). Other systems specifically do not restrict topology. Both choices constitute architectural statements.

Another reality of topology is that different parts of the system belong to different owner/operators, and this fact has important implications for the architecture. One may want to use one's own resources by preference, to avoid using the resources of untrusted operators, and so on.

The traditional "cloud model" of the Internet is an attempt to boil the topology down to its most simple form—edge and middle. The "cloud of clouds" picture adds one level of detail—the different regions of the network (see Section 3.5).

2.1.4 The End-to-End Arguments

The end-to-end arguments [E2Earg81] (see Section 6.1.5) are an attempt to link two architectural frameworks—a layerist model and a topological model (at the sophistication of "edge vs. middle"). The end-to-end arguments assert that functions "in" the network should be restricted to the lower layers of the dependency/abstraction model, while application layer functions (higher level functions) should be moved "up and out": up through the layers and out to the edges. This produces a middle that (at best) can be general, simple, and flexible in the face of new demands. We return to a consideration of the end-to-end arguments later.

2.1.5 Generality and re-use

Architecture is a world of parts combined to make a whole – a world of building blocks, modules, abstractions, and so on. It is also a world of systems cut into parts—component modules with their interfaces. With skill and understanding, building blocks can become general and reusable. Reusable parts reduce the total design effort, not only by breaking a problem into parts, but also by reducing the total number of parts to be designed. However, reusability requires that the designer find these general building blocks and isolate them. This is perhaps the most artful part of architecture—seeing the common function in a number of different places and defining a single module with a single interface that can serve multiple purposes.

The Internet is conceived as a general-purpose, shared infrastructure. The reuse of its services by multiple applications is a primary requirement. It was not always clear what the primary reusable service of the Internet should be. In the beginning, the reusable service was thought to be the reliable byte stream of TCP [CerfKahn74]. However, in the early stages of experimentation during the late 1970's, it became clear that not all applications (in particular, voice over IP), want a reliable byte stream, so TCP was split from IP to expose the more primitive service of IP as the reusable layer.

Section 2.2 discusses how the Internet architecture approaches the problem of realizing a general purpose, reusable transport service.

2.1.6 Architecture and the design process

A system as big as the Internet cannot be designed by one team. The design teams must be broken into parts, and there must be interfaces between these teams, just as there are interfaces among the system components. Architecture is a structuring tool during the process of design; it breaks the design process into parts, so the tasks can be performed with relative independence. The structure that is proposed for the design process can have a profound outcome on the structure of the system itself. The OSI seven-layer reference model [ISO84] was proposed in part to organize the design teams, and the resulting specifications had a strongly layered structure. On the other hand, the management of the IETF has a dual structure that exposes both layers and vertical functional crosscuts, most obviously security and network management.

2.1.7 Modularity of implementation and deployment

The modularity of a design also shapes the modularity of implementation. Further, for a physically distributed system such as the Internet, these separately implemented parts are connected to build the

actual running network. In other words, the architecture defines a modularity that applies at design time, at implementation time, and at deployment time.

Parts that are tangled up in undefined ways at design time have undefined dependencies, and they tend to emerge from implementation as single monoliths; as a consequence, they become the basic building blocks of deployment and operation. For example, the Internet architecture has no defined interface between the route computation code and the forwarding engine in a router. So they get combined at implementation time, and there is no way that a network operator can replace the routing algorithm with one implemented by a different vendor. Defining the routing/forwarding interface (especially if it were defined as a network protocol) would permit parts of routing and route management to be moved from the router onto a separate processor.

Parts that are specified as separate modules can more easily be implemented as separate modules. Modularizing the implementation process to produce code modules of manageable size, to permit module replacement, and so on, is often the goal. However, sometimes a different implementation goal is more important—combining in one implemented module functions that were separately specified, perhaps to achieve a higher performance. This approach was behind the idea of Integrated Layer Processing, or (ILP) [ALF90]. Implementations of multi-layer network software with each layer in a separate module can trigger very high performance costs, especially if the modules are implemented as separate processes. ILP proposed that a single code module implement relevant functions from multiple layers in a protocol stack.

Interfaces between modules also provide a place to add new components. NAT boxes were not conceived as a part of the Internet architecture. But they can be added, architecture or not, because there is a defined interface at the point where they are added to the system. If some of the protocols were not open and specified, there would be no way for a third party to insert a "module in the middle," i.e., to insert a *middlebox*. So interfaces also provide an opportunity for disruption of the architecture.

2.1.8 Critical interfaces and the economics of architecture

Because architecture and the placement of interfaces define the parts that can be produced, sold, and operated by different parties, architecture defines a marketplace. Only when the different producers in a market know their roles can there be an organized market. It is architecture that provides this structure.

For example, the set of interactions among ARPAnet packet switches (IMPs) was not specified as an open interface. It was an internal engineering interface, designed and implemented by the BBN Corporation. That made it essentially impossible for two vendors to manufacture interoperating IMPs. Later, in the early days of the Internet, there was a debate as to whether there was to be an explicit open interface between routers. The designers took this interface as critical to the evolution of the Internet and made an architectural assertion that all interactions among routers be fully and openly specified. As a result, we have multiple router vendors today. Architecture defines the framework that makes up the marketplace and the point of application of market forces.

Industry has lobbied for certain "cut points" in the design of the Internet, to define what parts may be implemented, sold, and operated by different parties. The term "critical interfaces" has been used to describe these architectural proposals. See the proposals for the architecture of the National Information Infrastructure that were proposed by the Computer Systems Policy Project [CSPP94] and the Cross-Industry Working Team [XIWT94]. However, most system designers are not trained to think about the economic implications of an open interface—what information needs to be exchanged to make a rational market across that interface. But mis-design of an interface can flaw the ability to negotiate about economic factors.

2.1.9 Designing for change

A topic that has received relatively little attention is how architecture can lead to systems that are more or less able to change and evolve. Modularity and the resulting abstract interfaces provide a tool for change, since a module can be replaced so long as the interfaces remain the same. But this begs the harder question: the placement of interfaces determines what is hard to change as well as what is easier. If the internals can change, it is because the interfaces are frozen. Cutting a system into many small parts may freeze it, because all the resulting interfaces become fixed points of the design. Conversely, a system composed of a few big parts may offer the implementer less guidance, but may be easier to upgrade. As the number of parts approaches one, upgrading becomes hard again, because it is essentially total replacement.

The distributed nature of real networks (the physical modularity of the system) imposes its own barriers to change; it is impossible to change all physical nodes at ones. Having defined the inter-router interface (the IP protocol) for packet forwarding, as noted in the preceding section, the early Internet designers realized after awhile that they had solved only part of the problem. At first, all Internet routers used the same routing protocol. There was no way to upgrade that protocol without a "red-letter day" in which all routers in the world were changed simultaneously, a clear impossibility even then. This led to splitting the routing computation hierarchically into "interior gateway protocols" (IGPs) and an "exterior gateway protocol" (EGP). The Internet is divided into administrative regions (autonomous systems) that each run their IGP of choice, while inter-domain routing is handled by the single global EGP (currently BGP-4). This allowed regions to upgrade their IGPs independently (but upgrading the EGP is still very difficult!)

2.2 Assuring interoperation

The basic job of a computer network is to allow different nodes to interconnect and interoperate. All the layers, the modules, the interfaces and so on must be understood in this context. So a fundamental part of the architectural specification is to make sure that interoperation can happen in the desired way. How might we think about what "the desired way" means, and how might an architecture undertake this task?

Superficially, interoperation is simply the ability of a set of computing elements to interact successfully when connected in a specified way. From the consumer's perspective, this operational test is sufficient. The network might be thought of as a giant black box with interfaces that application code attaches to; if these interfaces allow the application to work, the network is successful. However, at a deeper level, it is useful to ask *why* interoperation was achieved in a particular situation, because the *why* question will shed some light on when and where this interoperation can be expected to prevail. Interoperation can be achieved in a number of ways, with different implications.

As the previous section discussed, network services, called *protocols* in network parlance, are often organized into layers, which help structure the dependencies that exist in achieving interoperation. Interoperation is most commonly achieved by agreeing on a common definition of these protocols. Thus, in the Internet protocol suite, an application such as mail (which has its own specifications and standards) depends on the transport protocol TCP, which in turn depends on the network protocol Internet (IP). These protocols, organized into layers of building blocks, provide an infrastructure that makes it easier to achieve interoperation in practice. This sort of dependency in protocols is often illustrated by drawing the protocols in a stack. Interoperation is successful because all parts of the network use the same specification of the protocols.

However, if one examines the various layers of protocols, one usually finds that there is a layer below which common standards need not be used to achieve interoperation. Again, using the Internet suite as an example, mail can be exchanged between two machines even though one is attached to an Ethernet, and one on a dialup modem connection.

2.2.1 The spanning layer

Why are common standards needed at one layer, but not at another? Certain protocols are designed with the specific purpose of bridging differences of diverse lower layers. Such a bridging protocol provides the definitions that permit *translation* to occur between a range of services or technologies used below. Thus, in somewhat abstract terms, at and above such a layer common standards contribute to interoperation, while below the layer translation is used. Such a layer has been called a *spanning layer* [Clark98]. As a practical matter, real interoperation is achieved by the definition and use of effective spanning layers. But there are many different ways that a spanning layer can be crafted.

2.2.2 The Internet Protocols as an example

Expanding the Internet example above may make these points clearer. The Internet protocol suite contains a protocol that plays the role of a spanning layer, the Internet protocol, or IP. Consider how IP supports the forwarding of simple text mail. The format of mail is defined by the standard RFC-822[6], the required common agreement at the application layer. This protocol in turn depends for its delivery on the Internet's transport protocol, TCP. And finally, TCP uses the services of the IP protocol, which provides a uniform interface to whatever network technology is involved.

How does the IP spanning layer achieve its purpose? It defines a basic set of services, which were carefully designed so that they could be derived from a wide range of underlying network technologies. The Internet element called a *router* actually plays two roles in the Internet. First, as its name would suggest, it routes packets along the chosen path towards the final destination. Second, it translates or converts among the various low-level representations of packets used over the various network technologies it connects to. Router software, as a part of the Internet layer, translates what each of these lower layer technologies offers into the common service of the Internet layer. The claim is that as long as two computers running RFC-822 mail are connected to networks over which the Internet Protocol IP is defined to run, RFC-822 mail can interoperate.

This example illustrates the role of the spanning layer as the foundation for interoperation. To determine whether two implementations of RFC-822 mail can interoperate, it is not necessary to look at the implementation of Internet over any specific network technologies. The details of how IP is implemented over Ethernet or over ATM are not relevant to the interoperation of mail. Instead, one looks at the extent, in practice, to which IP has succeeded in spanning a number of network technologies. And the practical conclusion, as reflected in the marketplace, is that the Internet protocol defines a very successful spanning layer. The functions and semantics of the IP layer are well defined, as shown by the fact that many companies have become very successful by selling routers, the devices that implement the translation required by the Internet protocol.

The goal of the IP layer in the Internet is to support a range of applications above it (e.g. email, the web, voice, games, etc.) and a range of network technologies below it (e.g. Ethernet and other LANs, point to point links, wireless broadcast, cable TV and other residential distribution networks, satellite, etc.) This protocol structure has sometimes been drawn as an *hourglass,* with the narrow part representing IP itself, the single point of converged agreement, and the wide parts above and below representing the diversity of protocol options there. This hourglass picture is an elaboration of the simple “stack” picture of protocol layers.

2.2.3 Other spanning layer examples

An IP layer is not the only way that one could achieve interoperation. There are other approaches in use today, with different objectives as to the range of spanned technologies and the range of supported applications. Some specific examples may make the alternatives clear.

---

[6] The Internet standards are specified in a series of documents called RFCs, or Requests for Comments, which are available on the Internet. Citations to individual RFCs are not provided here. See http://www.rfc-editor.org.

NTSC video delivery: All the technologies that are part of delivering NTSC -- over the air broadcast, cable, VCRs, satellite, and so on -- interconnect in a very effective manner to deliver a video signal from various sources to the consumer. But NTSC video delivery is the only application supported. This circumstance has a different set of objectives: a wide range of technology that it spans below, but a single application it supports above. Compared to the IP hourglass, this would look more like a funnel sitting on its wide end.

ATM (Asynchronous Transfer Mode): Some of the developers of ATM, which is one of the network technologies in the bottom of the IP hourglass, initially expressed the ambition that ATM become so ubiquitous that it come to serve for all networking needs. Were this to happen, it would not then be necessary to define a separate spanning layer above it, because the service provided by ATM would become the universal basis for application development. While this vision did not work out in practice, it illustrates an approach to interoperation at a lower point than the IP spanning layer. This, if drawn, might look like a funnel sitting on its narrow end. The virtue of this approach is that a single technology solution might be able to offer a very sophisticated range of services, and thus support an even wider range of applications than, for example, the Internet approach. So the funnel, while narrow at the bottom, might be very wide at the top.

All of these schemes have in common that there is some “narrow point” in their structure, with a single definition of service capability. Whether it is a definition tied to one application, or to one network technology, this narrow point in the picture corresponds to a spanning layer, and is what forms the foundation of the approach to interoperation.

2.2.4 Comparing interoperation approaches

The various examples given above suggest a fundamental way to evaluate different approaches to interoperation, which is to assess what range of technology they can span “below”, and what range of applications they can support “above”. The central claim of the IP architecture is that it is essential to have an hourglass rather then either of the two possible funnel pictures as its key spanning layer. An approach that spans a broad range of networks is needed because there has never been, and in practice never will be, a single network technology sufficiently powerful and general to meet everyone's needs. The ATM solution, if fully embraced, would require replacing all the legacy networks, such as Ethernet, the telephone digital hierarchy, and so on. And even if this victory over heterogeneity occurred, we must expect and plan for new innovations that would in turn replace ATM, for example, IP over WDM. This evolution will always happen; it is a necessary consequence of innovation, cost reduction, and so on.

At the same time, the Internet must support a broad range of applications, not just one, because the Internet is designed to hook computers together, and computers are used for all sorts of purposes. There is no one single “killer app”, and there never will be. So the Internet architecture proposed a middle point of interoperation, which permits both a range of applications above, and a range of technologies below.

2.2.5 Spanning layers -- one or many?

Must there be a single point of narrowing in the hourglass, or instead can there be multiple points of common agreement? While the Internet protocol is the primary spanning layer of the suite, it is not the only one. For example, the Internet mail transfer protocol SMTP (RFC-821), is carefully written to be independent of TCP, or any other transport protocol. Thus Internet mail has "built in" an alternative spanning layer. Any reliable byte stream can be used to deliver Internet mail. And indeed, there were viable products called mail gateways, prevalent before the Internet came to dominate the network scene, that supported mail forwarding among networks with different transport protocols (Uunet, Phonenet, Bitnet, etc.) The generality of the SMTP spanning layer allowed mail to have a very wide reach during this period, much broader than the Internet itself. The price for this was that the only application supported was mail. This spanning layer provided no access to other applications -- remote login or file

transfer, for example. Such interoperation at the mail level could again be likened to a funnel sitting on its wide end.

As this example suggests, the situation that prevails in the real world is more complex than the single hourglass might imply, and one finds multiple points of narrowing in the same architecture. But the idea of a narrow point is always present in any general approach to interoperation. The hourglass of the Internet should thus be thought of as both an abstract and a concrete approach to interoperation. It illustrates the key idea of a spanning layer and then makes a specific assertion as to where that point ought to be.

2.2.6 How to spot a spanning layer

Below the spanning layer, the architecture presumes that there will be different technologies, with different features but not necessarily any global commonality. Since a spanning layer must allow conversion among these different feature sets, it must provide enough information for this conversion to be accomplished. In particular, the spanning layer must define what is needed end to end, or globally. The most basic element of a spanning layer is some form of end-point addressing that describes the eventual destination, in contrast to the addressing in the lower layers, which may be missing all together (e.g. on a point-to-point link with only one next-hop destination) or local to the technology (in principle, Ethernet addresses are globally unique, but they are used only locally). IP has its global addresses, email has its email addresses, and so on. TCP has no addressing built in, and needs an IP layer to act as its spanning layer.

More generally, a spanning layer is a specification of the end-to-end service: what it must preserve, and what it can change or ignore, in performing the conversion. We consider this point in Section 2.3.

2.2.7 Summary

The concept of *spanning layer* provides a framework for understanding and assessing interoperation.

We offer a careful definition of interoperation, beyond the operational test of "does it work":

Two implementations of an application can interoperate if 1) they are based on common definitions and standards at the application layer, 2) they use supporting services in a consistent manner, 3) they are based on a common spanning layer as a foundation for these services, and 4) the range of underlying network technologies to which the applications are actually attached are within the range that the spanning layer can reach.

There are two important aspects to this definition. First, it is stated in terms of interoperating applications. If the question of interoperation had been framed in terms of a different entity, such as a computer, there is no well-formed answer. Computers may interoperate, or not, depending on what application is running. Second, the spanning layer is defined as the “foundation” on which the definition rests. This word was chosen because one does not have to look below it, when testing for interoperation. If the spanning layer is well defined, and is known to work over the network technology in question, then the fact that the application is defined in terms of this spanning layer is sufficient.

We argue that spanning and conversion are key to network evolution. Given that spanning layers can be defined to operate on top of other spanning layers (for example the Internet mail spanning layer on top of the IP spanning layer), it is easy to bring a new spanning layer as well as a new application into existence, by building on one of the existing interfaces. If the service proves mature, there will be a migration of the service "into" the infrastructure, so that it becomes more visible to the infrastructure providers, and can be better managed and supported.

Acknowledgment:
This material on spanning layers is abstracted from a white paper written for the Computer Science and Telecommunications Board in 1998. [Clark98]

2.3 Specifying function—syntax vs. semantics

Most discussions of system modularity describe modules as providing a *function*. One program calls another program, that second program "does something", which may have side-effects and which may return some values that are a function of the internal state of the program and the input values. This form of description emphasizes the semantics of the module—what matters is what it does, not the syntax of the arguments.

In contrast, many books on Internet protocols spend pages on the formats of packet headers, and emphasize the syntax of the process, not what the network *does*. This emphasis arises from the fact that what the Internet does is very simple, it transports bits—what comes out is what goes in.

This "what comes out is what goes in" specification goes by various names that imperfectly describe it—it has been called *transparency* or *openness*. A better word might be that the network is *oblivious* to the meaning of the data it transports. A system that is oblivious to the meaning of its data cannot do much with it—it cannot process it, sort it, reformat it or compute results on it. It can still do some things, for example, information-conserving transformations like encryption, duplication, traffic analysis; it can also totally fail to deliver it. This level of functionality stands in contrast to what a database system does with data. A network is more like a file system than a database—a file system is oblivious to the meaning of the bits in the files, a database can perform operations on the data because it "knows" what the structure of the data is.

The oblivious network provides a clear assignment of duty among the various parts. Any understanding of the meaning of the bits lives only at the edge, in the applications and systems that are attached to the network. Applications send messages to each other, and these messages are literally delivered in the same format as sent. This is why, when we talk about Internet protocols, we tend to focus on the syntax: that is about all there is.

What would a network look like that is *not* oblivious to the meaning of the data it carries? The telephone system is such a network: the digital telephone system is designed "knowing" that the bytes it carries are voice encoded in a specific format. This knowledge can make the system much more efficient. For example, on expensive trunks, e.g. trans-Pacific, designers can install data compressors that more than double the capacity of the links. The compression is lossy and depends on the fact that the bytes encode voice. This is the advantage of knowing the meaning of the bytes. On the other hand, when modems were introduced, the compressors prevented the modem signals from being recognized at the receiver. To work around this, when modems begin a session, they send a tone that turns off the compressors. The designers of the phone system had to add this mechanism after the fact to deal with a new application, and in fact, it has the effect of making the system more oblivious.

2.3.1 Weak semantics

In fact, the specification of the Internet is remarkably minimal. Even with respect to functions that an oblivious network might perform, the specification is silent. The net makes no commitment about performance. Packets can be lost, reordered, duplicated and corrupted. The weak nature of Internet semantics is striking. Why is it thus?

The hypothesis of the designers was that almost all of the "problems" caused by weak network semantics can be "fixed up" at the edge. IP makes no commitment about reliable delivery, but TCP can number the packets, detect losses, and resend them until they get through. IP makes no commitment about corruption of data, but TCP can checksum the bytes, detect errors, and resend until they are delivered correctly. In the beginning, the Internet designers tended to dismiss or ignore those impairments in the net that the edge cannot fully fix up. The network designers expected the application designers to be find clever ways to compensate for this sort of problem. For example, variable data encoding in the application can compensate for variable network bandwidth. Weak semantics does not necessarily rule out broad classes of applications, but it may make the application designer's job harder.

In exchange, the benefit of weak semantics is that a wide range of underlying network technologies can be put to use to carry Internet traffic. If the Internet specification had called for a minimum data rate, a class of subnetwork technology would have been ruled out. If it had mandated classes of service, priority queuing, or error-free delivery in the MAC design, entire classes of technology would have been ruled. As it is, the Internet packets can be carried over almost "anything". So the claim is (again using the hourglass imagery) that the weak semantics of IP makes the application's job harder but does not actually narrow the top of the hourglass, while it makes the bottom as broad a possible. This was the original Internet design philosophy.

2.3.2 Today's reality

Today the principle of weak semantics is under attack from a number of directions, some technical, and some arising from the evolving interests (business or policy) of the various players.

- There are some data impairments that cannot be removed once they happen. There is no way to take latency out of a transfer at the receiver, so telephone calls and other real-time interactions like games sometimes don't work well. Application designers complain, with considerable validity, that some of these problems have to be fixed inside the network before the network can be a universal infrastructure. They also point out that some of these problems can be fixed within the network even if the network is oblivious to the meaning of the data.
- Even when it is feasible to fix a problem at the end point, doing so may make the application designer's job harder. Designers point out, again with some validity, that the purpose of the network infrastructure is to make their job easier.
- Network providers would like to offer enhanced services, and to do so they are tempted to provide more complex service agreements than the minimum specified by the standards. But for this to be valuable, the users and their application code has to be able to take advantage of it. In many cases providers want new standards for these enhancements, so they can persuade application builders to use the extra features.
- As the Internet has proven itself in the marketplace, it no longer has to be the "beggar" trying to run on top of whatever network technology it finds lying around. Network technology can be, and is being, designed that is tailored to the Internet requirements. Over time, if Internet engineers had an agreement, they could move to a new base of technology options with a richer set of functions, without narrowing the bottom of the hourglass in practice. Those technologies that could not meet the richer specification would just vanish from the market over time.
- A tighter specification of the semantics may reduce the cost of making components. Routers today implement the common packet functions in a so-called "fast path" implemented in silicon. Less common options are shunted off to a "slow path" that uses a general-purpose processor. But by dooming these variations to a low-performance path, they essentially doom them to wither and die, so that over time, the specification in fact, if not in principle, becomes simpler and more constrained.
- There is always pressure to break the oblivious model, and build knowledge of the meaning of the data into the network. The next section considers this point.

2.4 The End-to-End Arguments in today's world

Section 2.1.4 introduced the end-to-end arguments, which have served as a central design tenet of the Internet for almost two decades. The simple form of the end-to-end argument is very close to oblivious transport. The end-to-end argument says (in this context) to avoid when possible building function into the net, especially application-specific function. The net should provide a general-purpose service, which is adapted to the desired purposes at the edge. Building knowledge of the *meaning* of data into the network is completely counter to the end-to-end argument.

Today, the principle of oblivious transport and the end-to-end argument, as well as the principle of minimal semantics, is under attack. A paper published in 2001 [Blumen01] cataloged some of the forces that militate against the oblivious principle.

- Parties with interests adverse to the user (e.g. the government attempting a wiretap) cannot expect the user to implement this at the end-point. It has to be done "inside" the net.
- Internet service providers who want to enhance their service offerings by adding value want to do so at the application level, which is what the user perceives as the valuable layer of the protocols. (Even geeks seldom send bits just for the fun of it.) But ISPs do not control the end-nodes, so the only way to add value is to bring knowledge of the application into the net, in middleboxes.
- Providers want to stratify their users into classes, based on willingness to pay, by preventing lower-paying users from running certain applications (e.g. hosting a server in their home or playing specific games).
- Providers want to control usage (and thus cost) by blocking certain applications. This sort of blocking may also increase the provider's ability to sell a version of this application at a premium.

All of these examples have a common thread: they arise not from the desire to improve the basic semantics of the transport service, but as a result of a tussle of interests among subscribers, providers, and society. This tussle is fundamental—it is not something can be ducked by a clever architectural proposal. However, neither the oblivious model nor the end-to-end argument in its simple form can address the interests of the players who are "in the net".

The project produced a paper [CWSB02] on the fundamental nature of tussle. This paper contained the following proposal for a more complex alternative to the simple oblivious principle, or to the simple form of the end-to-end argument.

> **Evolution and "enhancement" of existing, mature applications is inevitable.** As applications become popular, lots of players, including application providers and the ISPs, will want to get involved in them, whether as a move toward vertical integration, as an enhancement of performance or reliability, or for some other reason. This will almost certainly lead to increased complexity, perhaps decreased reliability or predictability, and perhaps an evolution of the overall application away from the original vision. We should not imagine that anyone could do much about this. The best we can do is to design the architecture so that the end node can control what features are invoked "in the network."
>
> **Keeping the net open and transparent for new applications is the most important goal.** Innovation and the launch of new applications is the engine that has driven the growth of the Internet and the generation of new value. So barriers to new applications are much more destructive than network-based support of proven applications. Since new applications must, almost of necessity, be deployed incrementally, they most benefit from the transparent simplicity that the end-to-end arguments foster. By the principle of isolation of tussle, any barriers that are put into the network as a result of the desire to control mature applications or issues of trust should not prevent parties that want transparency from getting it.

Once an application becomes mature and the "meaning" of its protocols becomes stable and well understood, there are strong pressures for "the network" to move away from oblivious transport and start to be cognizant of that meaning. Our new design goal should be to make sure that this does not break the deployment of new applications, where oblivious transport is the necessary building block to get started.

2.5 Summary—network architecture

Network architecture addresses a more complex set of issues than simple modularity—it must consider functional slices such as performance and security, it must address both layering and topological distribution, and it must consider the resulting industry structure.

The basic function of the Internet is actually very simple—oblivious carriage of data from source to destination. This view was captured in the so-called end-to-end arguments posed in 1984. The minimal semantics of the Internet, as well as the basic premise of oblivious transport, are now being challenged by the reality of conflicting objectives among the parties that provide the "center" and the "edge" of the network. A future network architecture must take account of this tussle of interests.

2.6 References

[ALF90] D. Clark and D. Tennenhouse, *Architectural Considerations for a New Generation of Protocols*. Proc ACM SIGCOMM, Sept 1990.

[Blumen01] M. Blumenthal and D. Clark, *Rethinking the Design of the Internet: the End-to-End Argument vs. the Brave New World.* ACM Trans Internet Technology, *1,* 1, August 2001.

[CerfKahn74] V. Cerf and R. Kahn, *A Protocol for Packet Network Intercommunication*, IEEE Trans. on Comm, COM-22, No. 5, May 1974, pp. 637-648.

[Clark98] D. Clark, *Interoperation, Open Interfaces, and Protocol Architecture, The Unpredictable Certainty*: White Papers, Computer Science and Telecommunications Board, National Academy Press, 1998.

[CWSB02] D. Clark, J. Wroclawski, K. Sollins, and R. Braden, *Tussle in Cyberspace – Defining Tomorrow's Internet,* ACM SIGCOMM 2002, August 2002.

[CSPP94] Computer Systems Policy Project, *Perspectives on the National Information Infrastructure: Ensuring Interoperability,* 1994.

[E2Earg81] J. Saltzer, D. Reed, and D. Clark, *End-to-End Arguments in System Design*. 2nd Int. Conf. On Dist. Systems, Paris, France, April 1981. (Also in ACM TOCS, 2(4): 1984).

[ISO84] ISO. *Information Processing Systems - Open Systems Interconnection - Basic Reference Model.* ISO 7498, 1984.

[Neumann80] P. G. Neumann, R. S. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson. *A Provably Secure Operating System: The System*, Its Applications, and Proofs. Computer Science Laboratory Report CSL-116, Second Edition, SRI International, May 1980.

[Kiczales97] G. Kiczales, J. Lamping, et al., *Aspect-Oriented Programming*, Proc. European Conf. on Object-Oriented Programming, Springer-Verlag, 1997.

[XIWT94] Cross-Industry Working Team, *An Architectural Framework for the National Information Infrastructure,* 1994.

3 Specific project results

This section describes six specific architectural areas that the NewArch project examined more deeply, leading to concrete technical results and publications.

3.1 FARA: architectural model

3.1.1 Introduction

The NewArch project explored the use of top-down architectural design by developing and prototyping a new architectural model that we call *FARA* [FARA03] (also called "FARADS" in working documents). This underlying concept was to explicitly divide the reasoning process for designing a network architecture into two stages. FARA represents the first stage: a high-level model that satisfies a specific set of assumptions but has maximum generality. The second stage derives a complete architecture as an instantiation of the general model. A wide range of specific architectures could be derived from FARA; the NewArch project derived and prototyped one particular instance, *M-FARA*. A more complete description of FARA and M-FARA will be found in [FARA03, M-FARA03].

An important objective of the FARA model is to remove the overloading of the IP address that is found in the current Internet architecture. Thus, the current IP address is used both as an *address* to determine a network location for packet delivery, and as an *identifier* to identify communicating entities. This overloading provides some (minimal) security but makes mobility more difficult. There have been many proposals to create a separate identifier name space to simplify mobility (see [FARA03] for discussion of prior work). FARA avoids this overloading without requiring a new identifier name space.

3.1.2 FARA overview

The name "FARA" is an abbreviation for three fundamental elements of the model: "Forwarding directive, Association, and Rendezvous Architecture". The fundamental elements are entities, associations, forwarding directives, and rendezvous.

Entities

In the FARA model, the abstraction of host-to-host communication is replaced by packet exchange between *entities*. Intuitively, an entity is an application or similar thing. Structurally, an entity is an abstract concept, not linked to any particular implementation approach. An entity could be a process, a thread in a process, several processes, a whole machine, a cluster, and so on.

Associations

The use of the (IP address, port number) pair as a definition of destination identity is replaced in FARA by the notion of an *association*, which allows sequences of packets to access common state within an entity and synchronizes the communications between entities. Each packet carries an *association ID (AId)* that enables the receiving entity to demultiplex the message to a particular association. However, an association and its AId are strictly local to the containing entity; *FARA does not assume global name spaces either for associations or for entities.*

Forwarding Directive (FD)

FARA replaces the use of the IP address for packet routing by the more general notion of a *forwarding directive*, or *FD*, which routes the packet through the network and may be used for demultiplexing within an end system. Each packet carries a *destination FD*, which provides enough information to permit the forwarding and delivery of the packet to the correct entity. The packet may also carry a *source FD*, which will permit a return packet to get back to the source. The FD drives all forwarding actions to reach the destination entity, and then the entity uses the AId to locate the association state. An FD may be a generalized source route, it may use topological information that may or may not be globally unique, and it may be rewritten in route. The current IP address plays both the FD role and the AId role, while FARA specifically separates these roles. Note that *FARA does not require a single global address space*.

Rendezvous

Finally, FDs are obtained through the rendezvous process that returns, possibly through the use of a directory service, the FD of a destination entity and formatting instructions for the initial packet in an association.

The essential FARA modularity separates the forwarding mechanism of the *communication substrate* from the end-to-end communication functions performed by entities. It has been useful to visualize a metaphorical "red line", with the communication substrate "below" it and associations operating "above" it.

An entity has two formal properties: (1) it contains *state* that allows its association(s) to persist over multiple packets, and (2) an entity can *move* within the network. In fact, the entity is that unit of FARA organization that is independently mobile. Associating persistent state with the unit of mobility is self-consistent.

The benefit of separating the association and the forwarding directive is the freedom to change the FD and therefore the delivery path, for packets belonging to a particular association. Entities can move from place to place, one route can be preferred over another, and so on. This freedom is a kind of "generalized mobility", including provider-based addressing, multi-homed hosts and networks, mobile addressing, and so on. Of course, mechanisms must be defined to ensure that each entity has the up-to-date forwarding directives to reach other entities involved in ongoing associations. These mechanisms are not defined by the FARA model, but must be defined by an instantiating architecture (e.g., M-FARA).

The FARA model does not specify the content or format of an FD. In general, an FD will contain a series of sub-forwarding directives, each defined within some scope, and these will be used in turn to forward the packet. When an entity moves (i.e., when its topological address changes), carrying its associations, the corresponding FDs must be updated. How this happens depends on how rapidly the entity moves (does movement occur several times in a round trip, or does it take a few round trips to move, for example), so there may be several different ways to inform other entities on how to reach a mobile entity. The FARA model requires some mechanism to keep an FD up-to-date, but it does not specify how this should work.

3.1.3 Rendezvous: getting started

The previous discussion presumed on-going associations, but how do associations get started? That is the purpose of the rendezvous framework. The framework consists of two parts, *discovery* and *initiation***.** The discovery part returns a pair (*FD, rendezvous string***).** This is somewhat analogous to a DNS lookup that returns an IP address. The FD is needed to get the (first) packet to the desired entity, while the rendezvous string *(RS)* is used by that entity to decide how to process the initial packet and create the association.

The RS has much in common with a URL that redirects a query to a new Web site. URLs can be tagged with information telling the receiving Web site where the query came from, the objective of the query, and so on, and URLs are loaded with all sorts of dynamic information that determine rights, payments, and so on. The RS captures this idea in a general way. A rendezvous string template may be published by entities wishing to be found. This rendezvous string template provides instructions to an entity on how to construct a properly formed RS to reach a destination entity. The template may require the sender to perform calculations or include local information in the RS.

The operation of the discovery phase and the format and semantics of the RS are intended to be very general and extensible. For example, the FD returned by discovery can be that of intermediary or *agent*, which will operate on the FD to direct packets to the correct entity.

While a user can employ any tools, search engines, catalogs, directories, etc., for the discovery of an entity with which to communicate, it is useful to have a set of globally unique names that may be used

as handles to obtain information about an entity. However, such global names are not mandatory; private namespaces may also exist.

3.1.4 Association security

Separating the association identity from the FD raises explicit security issues, since the FD can be trivially spoofed. Verification may be required when an association is established, and two entities must be able to gain assurance that no intruder has entered the packet exchange even if the FD changes.

The FARA model relegates this assurance problem to the entities, but allows a range of assurance mechanisms between consenting entities. Entities might use strong crypto as a way of building the association, which would provide a high level of assurance that the association was not corrupted. Alternatively they might use the relatively weak assurance of the transport protocol – sequence numbers, checksums, etc. Some intermediate levels of assurance should also be available, e.g., the use of *nonces*.

3.1.5 An instantiation: M-FARA

The FARA model provides a high-level, consistent framework from which a wide variety of specific architectures can be derived. To gain confidence in its validity and to create a useful protocol architecture, the NewArch project created and prototyped the M-FARA architecture, one instantiation of the FARA model. M-FARA was designed to illustrate and exercise the mobility and addressing generality aspects of FARA. Thus, M-FARA included specific mechanisms for addressing, forwarding, FD management, and security. M-FARA was incomplete; the important issues of rendezvous and directory service were deferred for lack of funding.

M-FARA assumes multiple private addressing realms in the communication substrate, but unique addresses within each realm (hence, the destination FD is independent of the source, within a realm). An FD therefore contains a generalized source route of sub-FDs. To simplify the problem of computing an FD, M-FARA assumes the existence of a distinguished *core* domain in the "center" of the topology (see Section 3.3 on NIRA, below).

To update FDs for mobility, M-FARA defines a system of mobility agents, which act as rendezvous points and as third parties in communication. Corresponding to each entity there is an M-agent. An entity informs its M-agent whenever the entity moves (changes its FD). The scheme is outlined in [FARA03] and detailed in [M-FARA03].

For source verification, M-FARA uses the nonce system, in which a pair of tokens is exchanged during association creation serve as credentials. The two ends re-authenticate each other if an entity moves.

A prototype of M-FARA was built and demonstrated. The source code is available on the NewArch web site. This prototype used two addressing realms with IPv4 and IPv6 addresses and showed the ability to move between one realm and the other. See [FARA03] for the scenario.

The major conceptual design of FARA was due to Dave Clark. Aaron Falk and Bob Braden elaborated the design, while Venkata Pingali developed the M-FARA architecture and created and demonstrated an M-FARA protocol stack to achieve general mobility.

3.2 Role-based architecture

3.2.1 The trouble with layering

Under traditional network architecture, communication functions are organized into nested levels of abstraction called *protocol layers* [RFC46], and the metadata that controls packet delivery is organized

into *protocol headers*, one for each protocol layer [ISO84]. Protocol layering has served well as an organizing principle, but it is known to have serious limitations and problems, including the following:

- There is constant pressure for new layer violations, which may be assumption violations. Even the base Internet architecture includes implicit layer violations; for example, congestion is detected at the network layer but congestion control is exerted by the transport layer.
- Layer violations generally introduce implicit functional dependencies where layering had originally provided modular separation. These implicit dependencies cause unexpected *feature interactions* and a consequent loss of extensibility. The coarse functional granularity of traditional protocol layers also impacts extensibility.
- A reluctance to change working implementations and long-standing inter-layer interfaces often leads designers to insert new functionality between existing layers, causing layer proliferation. However, new services sometimes do not fit into the existing sub-layer structure at all, without introducing new layer violations.
- The rapid proliferation of *middle boxes* (firewalls, NAT boxes, proxies, caches, etc.) noted in Section 2.4 is a serious challenge to the architecture. Although such boxes are often introduced to meet legitimate engineering or management goals, they require control data that cannot be wedged into the existing protocol stack. The result is an impending proliferation of special-case signaling protocols, operating out of band from the data, in the "control plane".

We see that layering, useful as it is, has serious architectural limitations. The NewArch project therefore explored a fundamentally different architectural approach: a non-layered architecture, called *role-based architecture* or RBA [RBA02]. RBA provides an architected mechanism for selective non-obliviousness (see Section 2.3) in the network.

3.2.2 Role-based architecture

RBA organizes communication using functional units called *roles*. A role represents a communication building block that performs some specific function relevant to forwarding or processing packets. The inputs and outputs of a role are application data payloads as well as any controlling metadata that is required. Roles are abstract entities, and it should be possible to reason about them somewhat formally. Since roles are not generally organized hierarchically, they may be more richly interconnected than are traditional protocol layers. Since roles can correspond to features, these interconnections make feature interactions explicit and allow reasoning about them.

Roles are intended to be functional building blocks that are well defined and perhaps well known. We expect that a relatively few (tens to hundreds) of *well-known* roles would be defined and standardized (note that role-based architecture will not remove the need for standardization). However, the number of special-purpose, experimental, or locally defined roles is likely to be much greater.

RBA could allow re-modularization of current "large" protocols such as IP, TCP, or HTTP into somewhat smaller functional units. Examples of such units might be "Forward Packet", "Fragment a Packet", "Control Flow Rate", "Request Web Page", or "Suppress Caching". The assumption of RBA is that each of these separable functions may be assigned to a distinct role.

Moving to a non-layered architecture requires a distinct shift in thinking, with immediate implications. Traditional layering provided modularity, a header structure and ordering for the processing of metadata, and encapsulation; RBA must provide replacements for each of these.

- Under RBA, the structure of metadata in a packet header no longer logically forms a "stack"; rather, each packet carries a logical "heap" of protocol headers. The traditional nesting of packet header is replaced by a "container" that can hold variable-sized blocks of metadata, and these blocks may be inserted, accessed, modified, and removed in any order by the modular protocol units (roles). The metadata in a packet, called *role data*, is divided into chunks called *role-specific*

headers (RSHs). For example, role data can be used to carry middlebox control information in-band with the data flow.

- A non-layered architecture also requires new rules to control processing order and to control access to metadata, to replace the rules implicit in layering. By controlling the association between program and (meta-)data, RBA can explicitly control interactions among the different protocol modules, enhancing security as well as extensibility.
- In a layered architecture, each layer is encapsulated in the layer below. In non-layered architectures, a new organizational principle is needed for the data and metadata in a packet. Encapsulation does not disappear, but its role is much reduced; it is reserved for cases where the functionality is that of a container or adaptor, such as the encapsulation of a reliable byte stream within a flow of packets. Even in such cases, metadata about the data being encapsulated need not be itself encapsulated, as it would be in a layered architecture.

Under the idealized RBA model, all data in a packet, including the payload, is role data that is divided into RSHs. Just as roles modularize the communication algorithms and state in nodes, so RSHs modularize the metadata carried in packets. RSHs divide the metadata along role boundaries, so an RSH forms a natural unit for ordering and access control on the metadata; it can be encrypted or authenticated as a unit.

The granularity of RSHs is a significant design parameter, since role data cannot be shared among roles at a smaller granularity than complete RSHs. At the finest granularity, each header field could be a distinct RSH; this would avoid any replication of data elements required by multiple roles. However, overhead in both packet space and processing requirements increases with the number of RSHs in a packet, so such fine-granularity RSHs are not generally feasible. The optimal division of the meta-data into RSHs will be an engineering trade-off.

An end system does not necessarily know that a packet it sends will encounter a node that implements a particular role. The set of RSHs in a particular header may vary widely depending on the services requested by the client and can vary dynamically as a packet transits the network. The relationship between roles and RSHs is generally many-to-many; a particular RSH may be addressed to multiple roles, and a single role may receive and send multiple RSHs.

The role abstraction is designed to be independent of the particular choice of nodes in which a role will be performed. This *portability* of roles enables flexible network engineering, since functions can be grouped into boxes as most appropriate. Role portability may, but will not generally, imply role *mobility*, i.e., a role migrating dynamically between nodes.

3.2.3 Realizing an RBA

In applying the RBA concept, we can make various compromises with traditional layering.

- An architecture could be entirely role-based, i.e., all protocol functions from the present link layer to the present application layer as well as all middlebox functions could be replaced by roles or sets of roles. This would yield a completely layer-free, remodularized architecture.
- A less extreme approach would apply RBA only above a particular layer of the stack, retaining layering below that point and trading flexibility against efficiency.
- For example, if we consider link-layer protocols to be immutable, since they are designed by industry groups to match particular technological constraints, then an RBA subset might retain the link layer as a distinct layer “below” RBA.
- Alternatively, retaining the IP layer as the highly optimized common end-to-end packet transport service could significantly help to solve the efficiency issues with RBA; RBA processing would then be performed only in end systems and middleboxes.

- Finally, we could retain the transport layer and apply RBA only as an application layer architecture. Note that this could still help immensely with the middlebox problem.

3.2.4 Conclusions

The initial exploration by the NewArch project provided support for RBA as an architectural solution to the problems with layering that were listed earlier. A critical design decision when instantiating a role-based architecture is designing the packet format, since forwarding performance and simplicity is an important real-world issue. The NewArch project developed a few plausible examples and suggested a feasible packet data structure that could provide acceptable forwarding performance [RBA02].

Significant effort will be required to understand the potential of RBA and how to apply the RBA concepts to existing and future networking problems. The preceding section suggested a range of possible compromises between traditional layering and RBA. Another possible direction would use RBA to provide an unlayered network control (signaling) mechanism but with the current general modularity. The network functionality would be divided into major *protocol entities* that might (or might not) assume particular roles. This viewpoint emphasizes the addressability of a role; RSHs would generally be created by protocol entities but addressed to, and received by, roles assumed by other entities. Finally, the idealized RBA may be useful simply as an abstraction for reasoning about protocols, their functions, and their interactions.

Because RBA is such a major departure from our habitual layerist thinking in protocol design, its consequences are not readily understood without significant detailed design work. This should be the object of future research efforts.

RBA was explored by Mark Handley, Bob Braden, and Ted Faber.

3.3 Routing for user empowerment – NIRA

3.3.1 User empowerment in the architecture

Today's Internet users can pick their own ISPs, but once their packets have entered the network, the users have no control over the overall routes their packets take. ISPs make business decisions to interconnect, and the BGP routing protocol [BGP4-95] is used to select the specific route a packet follows. Each domain makes local decisions that determine what the next hop (at the domain level) will be, but the user cannot exercise any control over inter-domain routing.

A better alternative would give the user some control over routing at the domain level. User choice could foster ISP competition, imposing an economic discipline on the market and fostering innovation and the introduction of new services. An analogy can be seen in the telephone system, which allows the user to pick his long distance provider separately from his (usually monopolist) local provider. Allowing the user to select his long-distance provider has created the market for competitive long distance, and driven price to a small fraction of their pre-competition starting point. The original reasoning about Internet routing was that this level of control was not necessary, since there would be a large number of ISPs, and if a consumer did not like the wide area choice of a given local access ISP, the consumer could switch. Whether this actually imposed enough pressure on the market in the past might be debated. But for the consumer, especially the residential broadband consumer, there is likely to be a very small number of competitive local ISPs offering service. With cable competing only with DSL, the market is a duopoly at best (at the facilities level) and often a monopoly in practice. So in the future, the competitive pressures on the wide area providers will go down.

While it is only speculation, one can ask whether the lack of end-to-end quality of service in the Internet is a signal of insufficient competitive pressure to drive the deployment of new services. Certainly, there are many consumers who would be interested in having access to enhanced QoS, if it

were available end-to-end at a reasonable price. Such a service might have driven the deployment of VoIP, of various sorts of teleconferencing and remote collaboration tools, and so on. If the consumer could pick the routes his packets took, this might entice some provider to enter the market with a QoS offering, and a set of ISPs might in time team up to make this widely available. But there is no motivation to offer such a service today, since the consumer has no way to get to it. So one can speculate that lack of competition in the form of user-selected routes is one cause of stagnation in Internet services today.

3.3.2 NIRA – user choice over domain-level routes

The New Internet Routing Architecture (NIRA) [Yang03] is designed to give a user the ability to choose domain-level routes. A domain-level route is a sequence of domains a packet traverses, as opposed to a router-level route, which is a sequence of routers a packet traverses. NIRA uses a provider-rooted hierarchical addressing scheme for efficient route representation and scalable route discovery. Top-level providers obtain unique global prefixes and allocate subdivisions of these prefixes to their customers. These customers may recursively allocate address prefixes to their customers. Using this addressing scheme, a (source address, destination address) pair can uniquely identify a provider-level route. This addressing scheme also enables scalable route discovery. To reach a specific destination, a user does not need to know the entire inter-domain topology. Instead, the user only needs to know the part of the network that connects him to the top-level providers.

The Topology Information Propagation Protocol (TIPP) is central to NIRA. This protocol propagates to users address allocation information as well as topology information based on policy configurations. TIPP uses a (simplified) link-state-like algorithm. The protocol has the fast convergence property but not the complexity of typical link-state protocols.

Another important piece of the NIRA architecture is a distributed Name-To-Route Resolution Service (NRRS) that tells a user the addresses, and optionally the topology information, of another user to whom he wants to communicate. Combining this information and the information learned from TIPP, a user is able to choose an initial route to contact another user. Similar to DNS, NRRS uses a hierarchical namespace and hard-coded addresses for bootstrapping. However, because addresses in NIRA are topology-sensitive, a fundamental tradeoff is that topology changes would cause address changes. We think that NRRS server updates are likely to be manageable because only static topology changes, such as providers changing peering agreements, would cause address changes. Recent Internet measurement results have shown that the static Internet topology changes at a relatively low rate.

Route failure handling in NIRA is a combination of proactive notification and reactive discovery. TIPP proactively notifies a user of route failures in his part of the network. When a packet traverses the network, if a router finds the route specified in a packet header is unavailable, the router will try its best to send a control message to the sender of the packet, notifying him of the route failure. In case such router feedback is unavailable, users can always use timeout to discover route failures. As a user knows other domain-level routes between him and the destination user, once he discovers a route failure, he can quickly switch to an alternative route.

NIRA supports practical provider compensation. Similar to today's Internet, NIRA's provider compensation model is based on contractual agreements between providers and customers. As users can specify arbitrary routes in their packet headers, providers shall install policy filters to prevent illegitimate route usage. In cases where business relationships exist only in directly connected entities, policy filter checking for valley-free routes becomes verifying that the source address in a packet header matches the interface the packet comes from. For more complicated routes, policy filter checking may require matching source routing header against policy filters. For indirect business relationships, policy checking becomes verifying the identity of the sender of a packet at forwarding time. This problem is an example of the general and open problem of "single packet identification."

Overlay providers and second-hop providers that want to sell QoS to end customers face the same problem. Any solution to this general problem can be plugged into NIRA.

NIRA is compatible with IPv6 to facilitate deployment, and in particular it uses the IPv6 header format. A NIRA router's forwarding algorithm is changed to examine both the source address and the destination address in a packet header. This forwarding algorithm effectively prevents source address spoofing.

NIRA was designed by Xiaowei Yang as a Ph.D thesis project. She simulated NIRA using NS-2, including extensive packet-level simulations to validate scalability, robustness, and correctness using the most recent topology measurement and domain relationship inference results. She also developed an analytical model to study the system's behavior under various failure rates.

3.4 XCP congestion control -- general QoS framework

This effort developed a new approach to Internet congestion control that allows individual flows to obtain large end-to-end throughput (e.g., a few Gb/s). Current TCP-based congestion control [VJ88] cannot provide a large per-flow end-to-end bandwidth-delay product. This becomes a serious limitation as more users access the Internet using gigabit Ethernet or other high bandwidth link technologies. The limitation arises from two characteristics of TCP. First, TCP cannot quickly acquire large amounts of spare bandwidth because it increases by a constant amount of 1 packet/RTT. Second, TCP's throughput is inversely proportional to the packet drop rate. As a result, the packet error rate (PER) of the underlying link technology sets an upper bound on the end-to-end TCP throughput, which prevents the TCP from obtaining a very high throughput even over low error-rate fiber links [HZH01].

The project therefore funded development of XCP (eXplicit Control Protocol) [XCP02], a novel protocol for congestion control that outperforms TCP in conventional environments and remains efficient, fair, and stable as the link bandwidth or the round-trip delay increases. XCP generalizes the Explicit Congestion Notification proposal (ECN) [ECN99]. Instead of the one-bit congestion indication used by ECN, XCP routers inform the senders about the degree of congestion at the bottleneck. Since its per-flow congestion control state is carried in the data packets, XCP avoids per-flow state in routers and can scale to any number of flows. Further, the router implementation requires only a few CPU cycles per packet, making it practical even for high-speed routers.

Another new concept in XCP is *the decoupling of utilization control from fairness control*. To control utilization, the new protocol adjusts its aggressiveness according to the spare bandwidth in the network and the feedback delay. This prevents oscillations, provides stability in face of high bandwidth or large delay, and ensures efficient utilization of network resources. To control fairness, the protocol reclaims bandwidth from flows whose rate is above their fair share and reallocates it to other flows.

XCP enables very large per-flow throughput (e.g., more than 1 Gb/s), which is unachievable using current congestion control. Additionally, extensive simulations showed that XCP significantly improves the overall performance, reducing drop rate by three orders of magnitude, increasing utilization, decreasing queuing delay, and attaining fairness in a few RTTs.

Tools from control theory were used to model XCP and demonstrate that, in steady state, it is stable for any capacity, delay, and number of sources. XCP does not maintain any per-flow state in routers and requires only a few CPU cycles per packet making it implementable in high-speed routers. Its flexible architecture facilitates the design and implementation of quality of service, such as guaranteed and proportional bandwidth allocations. Finally, XCP is amenable to gradual deployment.

In further work, the XCP framework was extended to provide quality of service (QoS). Since XCP naturally functions with small queues and almost no drops, we focused only on bandwidth differentiation mechanisms. First, XCP was shown to provide an infinite spectrum of relative

bandwidth differentiation. Simulations show that XCP's relative bandwidth allocation is almost as accurate as Weighted Fair Queuing [WFQ89], although XCP does not require per-flow state in routers. Furthermore, XCP achieves a relative differentiation that spans more than three orders of magnitude, which is difficult to obtain in other schemes. Second, XCP was extended to provide guaranteed bandwidth service allowing a sender to reserve bandwidth for the duration of the flow. Admission control is done in a distributed manner without any per-flow state at the router. The resulting scheme is work conserving, does not increase the complexity of the routers, and delivers the promised service. Further, best-effort traffic enjoys a good service with a small queue size and almost no drops.

Beginning from an original conception by Mark Handley, Dina Katabi developed and simulated the XCP algorithms as a Ph.D thesis project. Other researchers have picked up this work on XCP. A separately-funded effort at the Information Sciences Institute (ISI) has built a prototype implementation of XCP in FreeBSD and demonstrated it over a gigabit testbed. Further research and deployment testing is expected under other funding.

3.5 Regions in the architecture

The NewArch group discussions of scaling, heterogeneity, user empowerment, and trust led to consideration of the generalized concept of a "region" as a first class object in the network architecture. This region abstraction could address problems of scaling and heterogeneity in a variety of different domains, as demonstrated by the successful local application of the region idea to solve these problems at a variety of points in the current network architecture.

The underlying premise evolved from a consideration of two very different communication design problems. Internet packet routing uses a two-tiered design in which there is routing among Autonomous Systems (ASs) and, at a more detailed level, routing within each AS. One reason for this partitioning is scaling, because routing information for the whole Internet is unmanageable. Heterogeneity also comes into play here; the routing algorithms used within ASs are often intentionally different from the routing algorithms used among them, in order to address different performance problems. At the other extreme, regions are relevant to a ubiquitous networked computing environment, in which applications provide interfaces to humans through a collection of small independent devices. When one further adds mobility into such an environment, the users and their applications may find themselves in a vast sea of options and possibilities of resources that might be called upon to support and comprise an application. How does the application writer bound the scopes of searches to find resources to fit its needs? How does the user distinguish communities or resources to trust from those not to trust? Is there some way to bound costs of such searches? Regions may supply answers to these questions.

The definition of a region includes a formal definition of the characteristics that distinguish each region. These may or may not reflect topological or physical proximity, administrative control, and so forth. For an AS, a definition might include IP address ranges, the administrative controller and the internal routing algorithm. For a user in our project trying to configure an application, a useful region might be an overlay network or peer-to-peer (P2P) system sitting on top of a set of resources committed by MIT and ISI to the project. In this case, the region description might include both the type of the P2P system as well as a specification of the trust identity. The assumption in this work is that a member of a region conforms to the description of that region; if it cannot conform to it, it cannot be part of the region.

After introducing regions as first class architectural objects, we can begin to explore its benefits. We propose two clear benefits available only through this general approach, one abstract and one more focused on actual performance. Normally, the concept of abstraction is used to mask or transform functionality, generally on the basis of individual entities or resources. The regions design allows discovery and exchange of information about individual resources through the region abstraction. An

example of this was explored in Li's thesis [Li03], resulting from this project. Li explored the use of AS information and AS path information to inform the routing that occurs in P2P systems. The original intention of structured (Distributed Hash Table or DHT-based) P2P systems was to dissociate any activity or placement of information in the P2P system from any knowledge of location or lower-level performance. The community then realized that the performance of such systems was potentially disastrous, so the common approach taken is that each member collects performance information. Our work demonstrates the utility of using more static and broadly-available information for performance improvement, by informing the P2P system of AS information. The core of the result is that one can achieve significant performance improvements with less network traffic and less storage overhead; this is a reflection of P2P regions being informed about AS region information.

The second major benefit of generalizing regions is that we can design and build an adaptation framework for them. The typical point solutions that incorporate a region-like approach are designed to solve a particular set of problems, such as the hierarchical partitioning of the Domain Name System (DNS). A more general mechanism can incorporate the ability of a region to reorganize its internal structure in order to adapt to changing conditions such as increasingly or decreasingly distributed membership, users, and so forth. Our in-depth work in this area has led to collaboration with local members of the machine learning community, in order to allow for learning from experience. We are also talking with Cisco about effective uses for this technology in real networks. This work is at the core of a mid-stage thesis.

Results to date of this work include [Sollins03], [Chan03], [Law03], and [Li03], but work in this area is ongoing. Another thesis project is exploring the use of regions in support of trust domains. To date, the majority of the results have been demonstrated in simulation, although the work of Steven Chan was completed in the context of the Goals and Pebbles projects at MIT [WTS03a, WTS03b].

The primary NewArch researcher on this effort was Karen Sollins [Sollins03]. The work began from an idea originally enunciated by Wroclawski in a workshop position paper [Wro97]. This work was supported in part by DARPA under NewArch, the National Science Foundation,[7] and by two supplementary gifts from the Cisco University Research Program.

3.6 Internet subscription systems

Internet subscription systems are mechanisms for notifying subscribers as quickly as possible of the arrival of relevant information on the Internet. Subscribers sign up for notifications on topics that are important to them. As soon as relevant information on a topic arrives on the Internet, it is routed to the appropriate subscribers. This information can include news alerts, stock market quotes, weather reports, and many other notices. Subscription systems can also support large-scale interactive games and transmission of on-line entertainment events. These systems enable the Internet to become not just a warehouse for storing information, but a clearinghouse as well.

Existing approaches to Internet subscription systems can be divided into three broad categories: (a) *unicast systems* that transmit notifications directly to subscribers over the Internet's existing forwarding mechanisms; (b) *single-identifier multicast systems* that send messages through discrete message channels to which subscribers with identical interests may subscribe; and (c) *content-based multicast systems* that forward messages based on their text content. No one now knows which of these systems will best meet the needs of Internet users.

None of these systems may be adequate for large, complex applications in which millions of users sign up for notices using complex, overlapping subscription categories. Unicast systems can easily clog network routers with numerous copies of the same message [Deering94]. With a small numbers of subscribers the problem of redundancy may be manageable, but with global scale systems the problems can become overwhelming. Single-identifier multicast systems cannot handle complex and

[7] NSF Grant ANI-0137403, **Regions: A new architectural capability in networking**, PI: Karen Sollins.

evolving subscription categories efficiently [Adler01]. And content-based systems take a long time to process notification messages. Studies have shown that the amount of time that it takes a content-based router to match a notification with its subscriptions grows linearly with the number of subscriptions in its table [Aguilera99]. As a result, the development of efficient matching algorithms for content-based systems has become a hot topic of research within this field [Guilera99, Preotiuc00, Banavar99, Pereira00, Altinel00, Carzaniga03].

Research funded by the NewArch project developed and evaluated an alternative approach to Internet subscription systems [Kulik03], known as the *Fast, Flexible Forwarding System* (F3). F3 is designed specifically for large-scale, complex applications, and it uses distributed multicast mechanisms to support such applications. The F3 architecture is built around two important features.

- All F3 messages pass through preprocessors before entering the F3 network. These preprocessors identify message information that is relevant for forwarding decisions and attach this information to message headers. In contrast with content-based subscription systems, which forward messages based on their text contents, F3 forwards messages based on these preprocessed headers. As a result, F3 can support a variety of message formats at the edge of the networkl using a single, efficient header-format within the network itself.
- F3 routers store subscription information using a data-structure, called *content graphs*, to represent the relationships among subscription topics. Using content graphs, F3 routers can determine the relationships between subscriptions and notifications efficiently. F3 can distribute a notification to the correct subscribers by simply looking up a single, numeric identifier in a hash-table.

Both simulated and working versions of F3 system were developed. The simulated version of F3 was developed using the *ns* simulator, and the working router runs on Linux platforms. Currently, F3's preprocessor takes subscriptions and notifications formatted as XML attribute-value pairs and produces content graphs. Future preprocessors will provide F3 applications with other interfaces, such as type-based interfaces [Eugster00, Eugster01]. An actual RSS news dissemination service also currently operates on F3. Subscribers reach the dissemination service through an interactive web page, where they sign up to receive RSS news updates on a variety of topics from various news sources. Whenever one of these news sources publishes a story on a requested category, F3 forwards the story to the appropriate subscribers, and a headline and summary of the story pops up on the subscriber's machine.

The performance of F3 was compared to unicast, single-identifier, and content-based systems in a variety of simulation scenarios. Preliminary results from these scenarios were presented at the Workshop on Distributed Event-Based Systems in June 2003 [Kulik03]. The scenarios involve three tasks: sports announcements, traffic alerts, and generic attribute-value notifications. They show that the time it takes F3 to match subscriptions with notifications grows logarithmically with the number of subscriptions in its table. Specifically, an F3 router running on a 2.8 GHz Intel processor was able to process a notifications in ~10 usec when it has received a million subscriptions in all scenarios studied. These processing times were between 1,000 to 10,000 times faster than those exhibited by a state-of-the-art content-based system, SIENA [Carzaniga03], under the same scenarios. Furthermore, F3 forwarding tables were at most 800 kB, between 100 to 1,000 times smaller than SIENA forwarding tables. In conclusion, F3 is able to support the same, rich interfaces as other subscription systems, at a significantly lower cost, and represents a promising development in the area of Internet Subscription Systems.

This research was performed by Joanna Kulik at MIT.

3.7 References

[Adler01] M. Adler, Z. Ge, J. Kurose, D. Towsley, and S. Zabele, *Channelization Problem in Large Scale Data Dissemination*, Proc. 9th IEEE International Conf on Network Protocols (ICNP), Nov. 2001.

[Aguilera99] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley and T. D. Chandra, *Matching Events in a Content-based Subscription System,* Proc. 18th ACM Symp on Principles of Dist. Comp. (PODC '99), May 1999.

[Altinel00] M. Altinel, M. J. Franklin, *Efficient Filtering of XML Documents for Selective Dissemination of Information*, VLD Journal, pp 53-64, 2000.

[Banavar99] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman, *An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems,* Int Conf on Dist. Comp. Sys., 1999.

[BGP4-95] Y. Rekhter and T. Li, *A Border Gateway Protocol 4 (BGP-4),* RFC 1771, Internet Engineering Task Force, 1995.

[Carzaniga03] A. Carzaniga, A. L. Wolf, *Forwarding in a Content-Based Network*, Proc. ACM SIGCOMM 2003, August 2003.

[Chan03] S. Chan, *A Semantic Checkpointing Framework for Enabling Runtime-reconfigurable Applications*, MIT Master's thesis, August 2003.

[CPRW03] D. D. Clark, C. Partridge, J. C. Ramming and J. Wroclawski, *A Knowledge Plane for the Internet*. Proceedings of ACM SigComm 2003, Karlruhe, Germany, August 2003

[Deering94] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, *An Architecture for Wide-Area Multicast Routing*, Proc. ACM SIGCOMM 1994.

[ECN99] K. K. Ramakrishnan and S. Floyd, *Proposal to add Explicit Congestion Notification (ECN) to IP*, Internet RFC 2481, January 1999.

[Eugster00] P. Eugster, R. Guerraoui, and J. Sventek, *Distributed Asynchronous Collections: Abstractions for Publish/Subscribe Interaction*, ECOOP, pp 252-276, 2000.

[Eugster01] P.Th. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, *The Many Faces of /Subscribe*, Tech Report EPFL, DSC ID:2000104, Jan 2001.

[FARA03] D. Clark, R. Braden, A. Falk, and V. Pingali, *FARA: Reorganizing the Addressing Architecture***.** Proc. ACM SIGCOMM FDNA Workshop, Karlruhe, August 2003.

[HZH01] R.Hui, B. Zhu, R. Huang, C. Allen, K. Demarest, *10 Gb/s SCM Fiber System Using Optical SSB Modulation,* IEEE Photonics Technology Letters, v 13, August, 2001.

[ISO84] ISO. *Information Processing Systems - Open Systems Interconnection - Basic Reference Model,* ISO 7498, 1984.

[Klein00] J. Kleinberg, *The small-world phenomena: an algorithmic perspective*. Proc. 32nd ACM Symposium on Theory of Computing (2000), pp. 163-170.

[Kulik03] J. Kulik, *Fast and Flexible Forwarding for Internet Subscription Systems*, Proc. 2nd Int Workshop on Dist. Event-Based Systems (DEBS'03), June 2003.

[Law03] C. Law, *Garbage Collection in Regions*, MIT Master's thesis, May 2003.

[Li03] J. Li, *Improving Application-level Network Services with Regions*, MIT Master's thesis, May 2003.

[M-FARA02] V. Pingali, A. Falk, T. Faber, and R. Braden. *M-FARA Prototype Design Document*. USC Information Sciences Institute. Available from ttp://www.isi.edu/newarch.

[OPSF98] J. Moy, *OSPF Version 2*, Internet RFC 2328, April 1998.

[Pereira00] J. Pereira, F. Fabret, F. Llirbat, and D. Shasha, *Efficient matching for web-based publish/subscribe systems*, Proc. Int. Conf. on Cooperative Information Systems (CoopIS), v. 1901, LNCS, Springer-Verlag, Eilat, Israel, 2000.

[Preotiuc00] R. Preotiuc-Pietro, J. Pereira, F. Llirbat, F. Fabret, K. Ross, and D. Shasha, *Publish/Subscribe on the Web at Extreme Speed*, Proc. ACM SIGMOD Conf. on Management of Data, 2000.

[RBA02] R. Braden, T. Faber, and M. Handley**,** *From Protocol Stack to Protocol Heap -- Role-Based Architecture,* Proc. Hotnets I, Princeton, NJ, October 2002.

[RFC46] E. Meyer, *ARPA Network Protocol Notes*. RFC 46, Network Working Group, April 1970.

[Sollins03] K. R. Sollins, *Designing for Scale and Differentiation*, Proc. ACM SIGCOMM 2003 FDNA Workshop, Karlsruhe, August 2003.

[SV00] P. Stone and M. Veloso, *Multiagent Systems: A Survey from a Machine Learning Perspective,* Autonomous Robots, Vol. 8 No. 3, pp345-383 (2000).

[VJ88] V. Jacobson, *Congestion Avoidance and Control,* Proc. ACM SIGCOMM 1988, August 1988.

[WFQ89] A. Demers, S. Keshav, S. Shenker, *Analysis and Simulation of a Fair Queuing Algorithm,* Proc. ACM SIGCOMM 1989, August 1989.

[Wro97] J. Wroclawski, *The Metanet,* Research Challenges for the Next Generation Internet, ed. Computing Research Association, May 14-17, 1997.

[WTS03a] S. Ward, C. Terman, U. Saif, *Goal-Oriented System Semantics,* MIT LCS Research Abstracts, MIT Laboratory for Computer Science, Cambridge, MA, Mar 2003.

[WTS03b] S. Ward, C. Terman, U. Saif, *Pebbles: A Software Component System,* MIT LCS Research Abstracts, MIT Laboratory for Computer Science, Cambridge, MA, March 2003.

[Yang03] X. Yang, *NIRA: A New Internet Routing Architecture,* Proc. ACM SIGCOMM FDNA 2003 Workshop, Karlsruhe, August, 2003.

4 New perspectives

The NewArch project members spent over 100 hours in detailed technical discussion of network architectural principles and designs. Some of this discussion led to the specific research results summarized in the previous section. In addition, there were some less concrete results, gains in conceptual understanding of the issues in particular areas of the architecture. This section describes new perspectives on two areas, trust and application architecture.

4.1 In confidence we trust

4.1.1 Introduction

As the Internet matures, one often-heard requirement is that the system should be "more trustworthy." The high level intention is clear, but the choice of the word "trustworthy" invites a discussion of what "trust" is. A discussion of trust turns out to be not of just academic interest, but of great practical relevance in the design of mechanisms for systems such as the Internet.

4.1.2 What is trust?

Trust, as defined by non-technical thinkers, is a relationship between trustor and trustee in which the trustor is willing to assume that the trustee will act in the best interest of the trustor. This does not mean that the trustor can predict exactly what the behavior of the trustee will be, but that the trustee will use judgment and intelligence to restrict the range of actions undertaken.

One who is not trustworthy may be malicious, or simply inattentive, incompetent, or in an unsuited role. Humans use a mix of means to assess the trustworthiness of a party: past experience, explicit information, the nature of the relationship (blood is thicker than water, etc.), the role in which the party is to be trusted, and so on. Among humans, trust is a matter of judgment and emotional reaction for all the parties.

Using this definition of trust, it is different to talk of trusting a person or a system. Systems do not use judgment, and they do not select actions knowing what is in the best interests of their users. They do what they are designed to do, blindly. A different word may be helpful to distinguish how we should think about a system—a user has *confidence* in a system if it has been implemented in such a way that it does what it is supposed to do. One would then talk about *high-confidence* systems, not *trustworthy* systems.

But this simple distinction masks an important reality of human behavior. While a system does not exercise judgment, a user may use the same sort of judgments to decide whether to trust a human or have confidence in a system. No normal user can be expected to understand how a complex system like Windows or the Internet actually works. The user observes the behavior of the system, seeks explicit information about what it does, evaluates the credentials of the designers, and uses judgment to integrate these facts. This feels, from the perspective of the user, very similar to the process of developing trust in a person. So it is natural for a person to say that he trusts his car, as well as say that he trusts his friend. Both the similarity and the difference are very important.

In fact, the Internet is so complex that most users do not know what it is actually supposed to do, let alone how the mechanisms work. Why do they use a system if they don't know what it is supposed to do? The real answer is that the Internet is designed and operated by people, people who have the power to exercise judgment, and the users expect those people to have acted in their best interest. This is precisely trust, in a human-to-human sense, even if the human is remote and the relationship weak.

The automobile illustrates all of these dimensions of trust and confidence. No normal car owner today can know in detail how a car works, and no normal owner could even understand the detailed design specs. Users develop a model of how they can trust their car based on experience with it, the context of

regulations within which it was designed, and so on. But when a real failure occurs, our reaction is to sue the car manufacturer. People view a failure of the car as a failure of a human-to-human relationship between the user and the creator of the device.

System designers may try to increase user confidence in the system by designing in constraints on what the system can do. One might dream about a computer system so well designed that "it cannot do anything bad." Such a dream is unrealistic—it is hart to conceive of a useful tool that has no opportunities for misuse. More to the point, in human society, constraint on behavior is not a basis for trust. Constraint is in some sense the opposite of trust. When one person trusts another, the trustee is expected to "do the right thing", even though not externally constrained to. A police state may greatly constrain what the citizens do, and this may provide certain sorts of predictability and assurance of behavior, but it does not induce trust. For real trust to develop, there must be freedom for the trust to be tested; there must be the potential for that trust to be violated. It is this risk, and the freedom that accompanies it, that is the essence of human trust.

4.1.3 Anxiety in Internet-land

The Internet is a complex system, and normal users don't know the design specification or how it works. They develop a model of how they are willing to use it based on trust-like criteria: past experience, explicit advice, trust in the creators, and so on. However, there is an additional dimension. When a person uses a car, or an operating system, they are interacting one on one with that device. But essentially all interactions across the Internet are shared interactions among multiple people—real people who may or not may trust each other. Whether the interaction is email, a web site, instant messaging, a chat room, or Internet telephony, there are real people responsible for the actions that occur at both ends.

So the Internet fully manifests both the problem of how a person develops confidence in a system and how one person decides to trust (or not trust) another person. Both of these modalities need explicit support. In particular, we as humans modify our behavior depending on the extent to which we trust the other parties in a transaction. We seek constraints on behavior when trust is missing, and relax them when trust is present. That variation must be mirrored in the Internet. This realization is the key to making the Internet more trustworthy and useful in an untrustworthy world.

The mantra of the future Internet may be "global communication with local trust", but it was not always that way. In the beginning, the Internet was operated and used by a small group of people with many ties to each other and a high degree of mutual trust. The design of the Internet was well suited to that trusting world. The Internet imposed few constraints on what parties can exchange with each other. The Internet tried to be as transparent as possible, which was viewed as a virtue. When there are no constraints imposed by the Internet on what users can send, innovation and creativity are given free rein. Part of the power of trust, as well as the risk, is the lack of constraint and the dependence on the judgment of the parties.

But in the current Internet, universal mutual trust is a foolish expectation. One alternative to trust is constraint—externally imposed restrictions on what one user can do with/to another. So firewalls change the Internet from a "that which is not prohibited is permitted" world to a "that which is not permitted is prohibited" world. This outcome trades away flexibility and the ability to do something new for the predictability that constraints bring. But that is a painful price to pay. Is there a way to have both?

4.1.4 Acknowledging the role of trust

In the real world, we recognize that we deal with people across a range of trust. Some we trust completely, with our life, our wealth, our well-being. And we associate great freedom of action with these relationships. With others we have little trust, and we use the tools of society to impose constraints on the interactions. We don't meet strangers in a dark alley, we use third parties to verify

and assure major financial transactions, we lock our houses (usually), and so on. Society gives us these tools and also the means to bypass them. And it is among trusting parties, where the overhead of interaction is lowest, that we most easily find innovation, novelty, and originality. Can the Internet mirror this?

Consider again the firewall. Today, the firewall blocks the transparency of the Internet for "outsiders," the strangers on the far side of the Internet. Inside the firewall these restrictions do not exist. New applications can be deployed easily inside the firewall, but not across it. What is primitive about the firewall is not the restriction it imposes, but the blanket division of the world into "inside" and "outside". Security experts know that the "insider threat" is a real one, and that one's closest collaborators may be far away across the network. If the firewall is frustrating, it is not because it imposes an unwelcome restriction, but because it imposes it on what may be the wrong set of people. In the real world, it is usually the people themselves who decide who they trust, not just a corporate security officer, and the pool of trusted people is not defined by network topology. We need to mirror this in the Internet. The topology-trust linkage implicit in firewalls is like looking for our keys under the lamppost.

We conclude that what is needed in the Internet of tomorrow is a delivery model that we call *trust-modulated transparency*. This concept implies that the network can offer a range of behavior when two (or more) nodes communicate, based on the declared wishes of those nodes. If all the end points request, the flow of data among them should be as transparent and unconstrained as the Internet of today. But either end should be able to require that the packets being received be checked, filtered, or constrained in ways that limit the risk of damage and limit the range of unexpected behavior. How might such a mechanism be built? The next two sections discuss the issues of identity and placement of the function within the network.

4.1.5 Identity as a first class building-block

One building block of trust is obvious. Interacting parties need to know whom they are talking to if they are going to place their faith in trust. It must be possible for two parties to have sufficient confidence that they are talking to those they think they are.

In its pure form, identity means that one person knows the identity of the other party, in a way that maps to a specific known person. However, there are sometimes cases where one is willing to make some assumption of trust without knowing the specific identity of the other party. One may know for certain only certain attributes, such as gender or nationality. One may trust an employee of a corporation to carry out his corporate duties properly, knowing only that he/she is an employee, because one trusts the corporation to vouch for its employee and stand behind their actions. In some cases, this sort of partial identity may be enough to justify a minimal assumption of trust.

Accountability is another dimension of the identity framework. If one is sure of the identity of another, and if all the parties share a context in which one can be held accountable for his actions, (e.g. a shared framework of laws and enforcement), one may risk some sorts of interaction without fully knowing or trusting the other party, because the threat of accountability imposes some constraints on what actually happens. One of the lessons of the Internet has been that as the interacting parties grow farther apart across the globe, accountability is less assured and less inspiring of confidence.

Identity theft destroys the basic fabric of trust, and must be prevented. Spoofing, masquerading as another, and so on are particular issues on the network, because we lack the full range of identity cues that we have when dealing with a person face to face. So it is necessary that the users have high confidence that the mechanisms that convey identity information work and cannot be subverted. Otherwise, the whole framework of trust is useless.

The importance of identity as a building block of trust, and of trust as a pathway to transparent, unconstrained operation, suggests that a system to manage and convey identity should be a fundamental part of the Internet architecture. However, an identity framework does *not* mean that all

transactions must always be linked to a known identity. There should not be a single mandatory global identity framework. System-wide imposition of identity is a constraint, not a building block of trust. Two parties should be allowed to depend upon a private way of identifying themselves to each other. It should be optional for parties to identify themselves reliably, to use multiple personae, or to be anonymous. A user may choose to be anonymous or to create a persistent identity that is untraceable to a real person, which is sometimes called a pseudonym. If the pseudonym is unambiguously associated with a single owner and cannot be stolen, a person interacting with a pseudonym can choose to develop a sense of trust based only on its past behavior.

If we believe that mandatory identification of the parties is not always necessary in order for applications to meet real needs, what design options exist? One design compromise would be to require that each party to a transaction know what the other parties have chosen to do, and be able to impose or relax constraints accordingly. It must be possible to discover how much is reliably known about the person with whom an identity is associated. If, for example, there is an on-line identity that cannot be traced back to an accountable person, that fact itself should not be hidable. A person choosing to act anonymously cannot hide the fact that they are doing so. Imposing that restriction on an identity and its owner seems like a defensible compromise, in that a receiver can now make a first line of decisions based on the willingness of an otherwise unknown party to reveal itself.

4.1.6 Sender vs. receiver verification

Where should trust or accountability be enforced? Identity has to be asserted at the sender—that is self-evident. But where should the decision be made as to whether this sender is trusted or can be held accountable? One answer is that this decision should be made near the sender. That is, if the sender is not a suitable participant in the network, he should not be allowed to send at all. Schemes of this sort try to build a network with only trustworthy (or accountable) people attached to them. The idea is to exclude untrustworthy people, and then, like the Internet of old, operate with few internal constraints and checks.

This sort of design has been called the "crunchy on the outside, soft on the inside" approach. The protection is at the network perimeter. The problem is that as the group gets bigger, the insider threat becomes more real, and as the scope gets more expansive, it gets harder to agree on a uniform standard for identity, trust and accountability. Imagine a region of the network with the rule that you can attach only if you agree not to be anonymous and to identify yourself so you can be held accountable for your actions.

The difficulty with this idea is who gets to decide if the users are properly identified and accountable. For a corporation, or even the government, this question may have a rational answer. But for a network of global reach, it gets very tricky. Either one regime declares that it is in charge and takes over the process of issuing user identities for the world, or various regimes need to trust each other to identify and track their users in a mutually consistent way. This raises problems, at a global level, of diplomatic proportions. Would all nations of the world be allowed to attach to a network where the users are expected to be held accountable for their actions? This seems unlikely, since there are countries without mutual extradition treaties. Who gets to make the decision that certain countries would be excluded?

The alternative to the "crunchy on the outside" networks is that decisions about trust are made closer to the receiver. One does not try to keep unwelcome traffic out of the global network, but just keep it away from the receivers that are not prepared to receive it. This partitions the problem—the networks must protect their own resources (routers and link capacity), and the receivers must form into units that protect themselves. These can be smaller, and thus easier to manage in a uniform way. It makes decisions about who is trusted a local matter, not a global one. In doing so, this approach ducks many of the policy questions that would arise in the other scheme.

4.1.7 Email as an example

Email is an application with rich structure that illustrates many of the points above. It was not designed with these ideas in mind; it has just grown up to match human needs.

Email is sent among email addresses, which are usually associated with people. There are no strong mechanisms that keep senders from lying about who they are, but until recently the sum of the features in the system made this sort of identity theft an uncommon event. It is possible for a person to have multiple mailboxes with distinct names, and the user may treat them differently, choosing to reveal more or less of his real identity with different mail addresses. Recipients react accordingly. There are examples today where merchants are not willing to trust a purchaser who uses an email address, which offers no linkage to a specific person (e.g., a hotmail account.)

The main problems for most mail users today are spam and virus attacks. Spam is unwelcome mail from senders unknown to the receiver, sent in bulk. Some of the most pernicious virus attacks work by penetrating one system and then sending outbound mail from that system in the name of the user. This is identity theft, and it may lure the receiver of a message to do something otherwise risky, such as opening an attachment, because the receiver is prepared to trust the sender.

One approach to dealing with spam is to impose constraints on what senders can do. Some outgoing mail forwarders try to detect sending patterns that look like spam and reject them. This produces the predictable set of problems: while it does prevent spam to some degree, it also can interfere with mailings that are welcomed by the receivers (e.g. large opt-in mailing lists), and it otherwise limits the creative use of mail.

Another approach to controlling spam, which has been implemented by some sophisticated users but is not a normal part of commercial mail software, is to deal with mail at the receiver based on knowing the source. The mail processing software at the receiver keeps track of whom the receiver knows, and treats mail from known senders in a different (more trusting) way. In one design, mail from a known user is delivered directly while other mail is held pending a check. The check consists of sending a message back to the original sender, asking that sender if they really did intend to contact the receiver, and asking them to perform a simple computation as a part of the reply. The idea is that a spammer normally does not process replies and often discards them, but a real person will not object to a "once in a lifetime" exchange of messages. Similarly, to control the spread of viruses, a mailer can be designed to deal automatically with attachments received from a known sender but to alert the user to the presence of any attachment from an unknown sender.

The essence of this scheme is dividing the world of users into those we know and those we don't. The level of actual trust and the associated risk is low—the only commitment is to receive a piece of mail. The power is in tracking a pool of known correspondents and opening up a transparent path only to them. One could imagine elaborating this idea of known identity to the creation of affinity groups, where membership in a group is restricted in some way, and joint membership in a group is enough to permit transparent delivery of mail. Once the basic idea of trust-linked transparency is accepted, there are lots of ways it can be made to work better.

Email is interesting because it sends messages asynchronously, so there is no possibility of a real-time challenge/response in the exchange. This reminds designers that the (or at least one) representation of identity must be verifiable in that context.

Two levels of trust-related control make sense for mail: (1) spam control and (2) deciding whether to trust the contents of the message. Spam control may not need a very robust mechanism, since the cost of a sender who is mis-identified is low—a piece of spam gets delivered. However, spammers can be expected to try to defeat whatever mechanism is globally deployed. For example, if mail software implemented the rule that it would accept mail from anyone that the receiver had previously sent to, one could imagine a "spammer's mailing list" service that sold lists of email addresses, and included

with each a false sender identity that the receiver had previously been tricked into sending mail to. One must think through the sequence of moves.

On the other hand, in order to decide whether to trust the contents of the mail, one may want a much more robust mechanism, typified by PGP or PEM (which illustrate the "bottom-up" vs. "top-down" approaches to mutual identification.)

4.1.8 Identity architecture for applications

The previous discussion leads to a set of application design principles related to identity.

- *Applications will need to convey information about identity from senders to receivers.*

  We can expect that identity management may be moved out of the individual application and into a system for this purpose. So there needs to be some standardization of what an identity "looks like".

- *There should be a standard format, usable by any application, for including identity information within an application-level message.*

  This does not constrain the semantics of the identity, only its representation. At the same time, the design should avoid locking users into one identity system rather than another.

- *Users should be free to use any means they choose to identify themselves to each other, so long as it can be represented in this standard format.*

- *There should be a range of pre-established systems for identity assertion that users can use when they don't choose to use a private means of identification.*

  These may vary in the robustness against identity theft, the degree of traceability of the identity back to an actual person, and so on. These differences should be characterized.

- *End systems, as part of their library of application support tools, should include a catalog of other users and their matching identifications, along with a set of trust assertions about these people.*

- *Applications should be able to register trust assumptions that they have derived, and a human user should be able to review and edit this catalog.*

  Many specific trust assumptions will be derived by applications and other programs, not specifically by people.

- *Applications should be designed so that their actions can be distinguished based on their potential to do harm if abused, and there should be some way to link these different actions to different levels of trust as expressed in the catalog.*

4.1.9 Protecting lower-level actions

The example above of user-level identity in email was at the application level. The same considerations seem to apply to other applications such as Instant Messaging, chat rooms, and so on. But what about problems at lower levels of the system? One can look at the lower levels of the systems that make up the end nodes and at the network itself. Both raise questions about trust, constraint on activity, and accountability.

Many security problems arise through the abuse of low-level mechanisms in the end nodes—SYN floods, smurf attacks, port scans and so on. The crafty attacks at this level are designed so that they are hard to detect and block, because they cannot easily be distinguished from acceptable behavior. So the "constraint" approach is less helpful. The particular "feature" of the Internet that makes many of these attacks possible is that the address of the sender in a packet is not validated, so one can have no confidence that it can be traced back to any accountable identity. Today, most proposals for dealing

with these low-level attacks depend on some trick for tracing the offending packet back to its source, so that either the sender or the source ISP can be held accountable. But in a new architecture, one can ask how to deal with this problem in a more structured way.

Our proposal is that the first packet of any interaction should carry identity information. In today's Internet, the first packet (e.g. the TCP SYN) carries no higher-level data, only a source IP address (network layer) and an initial sequence number (transport layer). Higher-level identity or authorization information can be exchanged only after the TCP state has been instantiated, so a receiver must invest resources without knowing to whom it is talking. In the FARA proposal (Section 3.1), the initial packet of any association, called the *rendezvous* packet, carries high-level information to initiate the association. If the high-level identity information is in the first packet, then the receiving entity has that information available before it decides to process even the first packet of an association. (This proposal, of course, raises questions about how this identity information can be delivered in an unforgeable manner, which is a good research problem).

There are still attacks that exploit implementation flaws at the lower levels of a system—buffer overrun attacks and their kin, which simply have to be identified and fixed. But the class of attacks that this "first packet identity" scheme prevents is the attacks where an action that is sufficiently benign when performed by a trusted correspondent becomes unworkable when triggered by a malicious party.

From the preceding we concluded that:

- *A mechanism for "first packet identity" should be devised that is robust enough that a receiver cannot be flooded by requests that require the receiver to take excessive action before they have verified the identity and trust at the application level.*
- *It should be possible to delegate this decision-making based on first packet identity to a guard machine that can take on the burden and risk of overload from suspect sources.*

4.1.10 Protecting the delivery process

The previous discussion concerned only low-level protection of the end node. What about protecting the Internet itself? Should the network change its behavior depending on whether it can identify and trust (or hold accountable) the sender of the packets?

Because the center of the network performs a relatively simple task, the range of attacks is similarly limited. In the center of the Internet we find links and routers. Routers can be attacked through the management interface, in which case they are acting as end nodes and the discussion above of identity applies exactly. Routers can also be attacked by sending bogus routing information. Again, the issue can be traced back to one of trust and identity. Routers inside an autonomous region (AR) are usually operated as if they trust each other. So a malicious router can inject false information unless it can be identified as an untrustworthy intruder. This is the router-to-router equivalent of the insider threat. It is reasonable for routers inside an AR to trust each other, provided they can distinguish real and bogus routers. Routing protocols across a region boundary (e.g. BGP) generally operate in a situation where trust is limited. The designers of BGP understood this, but the set of tools that have grown up over time to deal with this are rather *ad hoc;* manually configured tables are generally used to validate or constrain routing assertions. These protocols depend on constraint rather than trust.

Links are attacked by flooding them. This observation links identity and accountability to resource usage and congestion control, which is a large topic in its own right. One framework is the diff-serv architecture, in which the network polices the resource requests. If a user tries to flood the network, either he has paid for the privilege, in which case he is doing nothing wrong, or his packets will be tagged as outside profile and the first to die.

The diff-serv architecture postulates a new element in the Internet—in addition to links and routers there are policing points. This design explicitly recognizes that the ISP may not trust the user (or

another ISP) to do the right thing, and that validation/constraint is the necessary alternative. However, the design tries to make the constraint as benign as possible. This is the motivation for tagging rather than shaping or dropping.

The inter-ISP boundary is an important trust case to examine. If one ISP buys service from the other one on behalf of its users, then the relationship (trust or constrain) is between ISPs. Many major ISPs today are essentially forced to "trust " each other, because the tools for high-speed verification are not available. However, in the future there may be a requirement for the network to be able to identify the actual user. In our NewArch discussion of competition and user empowerment, we proposed that the individual user should be able to pick the intermediate ISPs that carry his traffic. To facilitate this, the packet has to carry some sort of identity to verify which user is sending the packet. This raises questions of how this identity is coded, but again notice that the basic issue is identity. This identity should not be the same as the end-to-end identity used to establish an association, first because it is unwise to require the end-to-end identity to be revealed to all parties, and second since routers are stateless, they have no idea of an association. This identity test may require some sort of soft state in the router, but it should be designed in a way that is distinct from the end-point association.

- *There must be some way to associate different transmissions (e.g., packets in a packet network) with the party that takes responsibility for them. This mechanism must be robust enough to support usage accounting and prevent flooding.*

  This should not be built on the same identity representation that is used at the application level, because that may not be visible to the routers (e.g., it might be encrypted), it may reveal too many attributes of the user, and it may not be associated with each transmission unit (applications may be willing to install more state than routers are).

  Since it is different from the higher-level mechanisms, it need not be more robust than necessary to deal with resource usage and flooding attacks.

4.1.11 Beyond identity—is transparency enough?

Is an identity mechanism the only important building block for trust-mediated transparency? How can constraints be implemented when trust is not present? The end-to-end argument would say that the end node should defend itself from any incoming packet that is not welcome. This is a fine principle. It raises some residual issues.

- First is the problem of flooding the link incoming to a host. If the packets have to come all the way to the host to be discarded, they consume resources, which may raise both performance and cost issues.
- Second is the problem of protecting an end host with possible security holes from low level security attacks.

Interposing a service of some sort between the end node and the untrusted foreign party can solve both sorts of problems. Firewalls are an example of a device that tries to limit the range of low level attacks. Application-specific servers, such as a mail server that filters incoming mail, are an example of a device that tries to export the first line of constraint for untrusted parties.

Architecturally, a consistent design for trust would require that firewalls and application-specific devices protecting a given host be able to tailor their behavior based on a trust specification provided by the end node. The user and its end-node must be able to select and configure any such service they use, and explicit expression of trust must be a part of any such service.

4.1.12 Summary

In the current Internet, it is not reasonable to expect a uniform basis of trust among all parties. Users will want to modify the behavior they will engage in with foreign parties, depending on the level of

trust. The totally transparent Internet of the past still has a place, among trusting users. This context can still facilitate innovation and unregulated interactions. But among users who don't trust each other, constraint rather than transparency is going to be needed.

To implement constraints, identity and the trust assertion for the identity must be a recognized part of the architecture. Packets that create associations must carry the high-level identity, so that low-level tags such as IP addresses are not used as stand-in for high-level identifiers. The application architecture (discussed in the next section) calls for end-node control over which servers are used to delegate application function. This requirement provides a means for an end-node to choose to protect itself by moving some of its constraints externally.

There are a number of cases in the Internet where we see a range of mechanism and behavior, and we do not have a clear model of why this range of mechanism exists. We hypothesize that the presence or absence of trust is a primary criterion to sort out mechanistic options.

4.2 Architecting a networked application

4.2.1 Introduction

The primary function of a computer network is to provide communication among application programs running in end systems. It is a happy property of packet switching that this function can be accomplished at a cost that is roughly independent of the distance separating the end systems. Application builders, who are generally distinct from network builders, use this communication service without requiring a detailed knowledge of its inner workings.

Network builders depend upon design principles that they call "network architecture"; in layerist terms, network architecture is concerned with protocol layers up to and including the transport layer. In this low-level view of architecture, the application layer is a black box that is designed by someone else, the application builder. Application builders, of course, have their own abstractions. The most common abstraction is the "distributed system" model, in which the end application programs are components of a distributed application.

This section surveys a range of important design issues for network-based applications: application-layer servers, adversarial design, application-level naming and identity, robustness and resistance to attack, performance, economics, and user empowerment.

These application-layer design considerations (which could be called "application architecture") are in part orthogonal to the network architectural ideas discussed in other sections of this report, although there are some areas of interaction.

The Internet already has a rich set of distributed applications; the most popular today are the Web, email, and peer-to-peer (P2P) computing. This section will draw on these applications for examples to illustrate the principles under consideration.

4.2.2 Application-layer servers

The dichotomy between network architecture and application design is being weakened by the rapid proliferation of middleboxes containing *application-layer servers* (ALSs). An ALS is logically part of one or more applications, but it is located "within" the network. The economic drivers for this proliferation were discussed earlier in this report; here we concentrate on the technical issues for application design. From the viewpoint of the application designer, the ALS is part of the application, even though it executes on a machine remote from the users' end systems and not under their control. ALS middleboxes are interposed in some way in the communication path between the end points. An ALS may be specific to a particular application, or it may be a *common ALS* that provides a service available to any application. An ALS is sometimes called "middleware", but this term is overloaded and suspect.

The existence of ALSs is an apparent contradiction of the end-to-end arguments (Section 2.4). These arguments imply that all application-specific code should run in user end systems; the core of the network should be simple, application-independent, and focused on the basic network service of data carriage. However, this is an over-simplistic interpretation of the end-to-end arguments. Most of the popular applications today are not simply end-to-end. Email is not sent directly between user computers, it is sent from user computer to mail server to mail server to user computer. The Web, which was initially a simple system of end computers playing the role of client and server, is now rich with caches, proxies, and the like.

In fact, there are many possible good reasons for moving parts of an application from the user end-nodes onto ALSs. For example, an ALS can:

- Stage content or cache data close to where it is needed, to reduce latency, reduce cross-network traffic, and improve performance (see Section 4.2.6.1).
- Reformat or preprocess information before sending it over very slow links or to end-nodes with limited capabilities (see Section 4.2.6.2).
- Make applications more robust to attack by distributing and replicating content (see Section 4.2.5.4).
- Manage identity and support cross-application reputation management, non-repudiation of identity, etc. (see Section 4.2.4).
- Control the extent to which each party's identity is revealed to others (see Section 4.2.4.2).
- Allow rendezvous in a world with Network Address Translators (NATs) (see Section 4.2.5.1).
- Allow staged interaction among machines like laptops that are not always online, and hold copies of content so it is available even if the original source is unavailable.
- Relieve users of operational problems by outsourcing part of the application (see Section 4.2.2.1).
- Provide the opportunity to observe, monitor and perhaps modify (e.g., filter) content in unwelcome ways.
- Pre-screen incoming messages to detect unwelcome senders or unwelcome content (see Section 4.2.5.3).
- Constrain what users can do, to create tiers of service that can be priced differentially.
- Provide financial returns to the operators, which can drive the deployment of the application.

The ALS architecture is relevant to many other aspects of application design, as indicated by the section references in this list.

4.2.2.1 Helping an application to grow up

Applications have different needs at different stages of their maturity. Many new applications start out either as pure end-to-end implementations (i.e., code running only on the user end nodes) or as end-to-end designs supported by some simple central server (e.g. to manage location, identity, or rendezvous.) But as an application becomes popular, different parties will see a changing set of needs. If an application comes to represent a significant component of overall traffic, ISPs will want to take steps (like caching) to reduce traffic loads. The application may be seen as a source of revenues by a number of players. Governments may be interested in observing, regulating, or constraining what is done using the application. Many of these interests will manifest as attempts to inject different players into the application's behavior by way of ALSs. So an application that is mostly end-to-end when it is young may grow up into a much more ALS-oriented architecture.

There is a parallel here with the need for "application management". When an application is young, there is usually no role for an "application manager". The users are the managers—they detect that something is broken and do their best to fix it. However, as applications mature and the base of users shifts from early adaptors to main-stream (less well trained and adventurous) users, there is an increasing need to outsource the management to professionals who can make the service more reliable and robust with less user involvement. Most individual users don't run their own POP or SMTP servers, although the architecture would let them do so.

A set of rules for an application might be: *it should work "out of the box" without any ALS support, when necessary, but it should allow ALSs and external management to be added incrementally as needed.*

Peer-to-peer (P2P) systems represent a very interesting special case in this spectrum of managed/unmanaged and end-to-end/ALS. P2P systems assume that there are no special machines designated as "servers", but that all the participants in the application play both the role of server and client. Users of the application help each other by providing resources. This pattern could represent a very interesting way for an application to start growing up—as enough users appear but before the application has attracted the attention of "professional service providers", the users can support each other, as services are needed beyond what the end points can do for themselves. At the same time, looking at P2P from a critical perspective, applications designed in this way may suffer from a form of arrested development; they may never be able to become fully mature. This is because the P2P design is so deeply embedded into the architecture that it is impossible for a service provider to offer a managed P2P service and attract users. In the drive to make sure that there is no central locus of control, many P2P systems remove the user empowerment to pick what services and servers they want to depend on.

A very interesting design challenge would be to design an application that can shift "automatically" from pure end-to-end to a peer-to-peer mode to a managed ALS mode as an application matures. These mode transitions should also be possible in the reverse direction: using managed services when they are well run and tussle-free, falling back on a peer-to-peer structure if the servers are missing or unwelcome, and at the end falling back to a pure end-to-end mode if only two machines are left running the application.

4.2.2.2 Application identifiers in initial request messages

Most applications have some sort of initial request message to establish contact with a remote service. Examples include an HTTP get request or a Telnet login. In early protocols like Telnet, the end-machine initiating the request did the translation from name to location, and then it just sent a message with an entity identifier relative to that location. If a user wanted to log into machine mit.edu as "bob", the Telnet client would look up the string "mit.edu" in the DNS, get an IP address, and then connect to that machine. It might then send a login containing the string "bob", but the string "mit.edu" would not be sent. In this case, the receiver could not tell if the connection actually came to the right machine. Furthermore, it was impossible to insert an ALS or relay in the path from the source to the destination, because the ALS/relay did not know the final destination.

The first version of the Web had a similar problem: when a browser extracted the DNS name from the URL, the DNS name was not sent to the server. This meant that it was not practical to have multiple web servers with different names on the same physical machine. Later versions of HTTP included the whole URL, including the DNS part. This made it possible for a single machine to act as multiple servers by using the DNS portion to disambiguate the request, and it allowed a cache or proxy ALS to attempt to satisfy the user request.

In contrast, email messages have always carried the full destination email address in the header of the message. Any server along the path can look at the message and see where the email is supposed to go. In fact, a server along the way may have a different idea (hopefully a better idea) of where the mail

should go. Without the full email strings in the email header, the relayed architecture of Internet email would not work.

This suggests the design rule:

- *Application messages should contain explicit, full application-level names.*

This allows an ALS to be interposed in the application-level connection. Assuming that this is what the users actually want, it is an important and useful capability.

4.2.3 Designing against adversaries — tussle

In a simple world, all parties to a communication would have consistent shared interests, but this is not realistic today. A very important aspect of application system design is protecting the application from its adversaries and their goals. The word "tussle" [CWSB02] has been used to capture the reality that parties with adverse interests vie to determine what actually happens in the network. Tussle pervades everything that users do. For example, a user may want to send mail privately, but the government might be observing it; a user might want to look at web pages in private, but the ISP may be gathering a profile of what the user is looking at; a user may want to receive mail, but not from spammers; a user might want to share music files, but the music industry wants to block unlawful distribution.

There are several forms of tussle, which can be cataloged as follows:

Two (or more) users with common interests try to operate in presence of hostile or adverse third parties. This pattern captures the tussle between the desire for privacy and the government's desire to wiretap, for example.

Two (or more) parties may want to communicate, but may have conflicting interests and want help. In this case, some third party may be involved to constraint or validate what is being sent. For example, a user may employ an incoming mail server that checks attachments for viruses, or a buyer and a seller may depend on a credit card company to provide protection from fraud. The end-points deliberately reveal (perhaps part of) what they are doing so a third party can become involved.

One party wants to participate in an application, but not with some other party, who intrudes. The most obvious example of this is spam—users want to use email, but not to receive mail from spammers.

It is important for an application designer to consider how each of these patterns is manifested in the specific application. Following sections contain detailed examples.

4.2.4 Openness and identity

Since the email application was designed for a totally open community, it provides an easy space for spammers to invade. Most instant messaging (IM) systems use a more restricted model: even though user names are globally managed, a potential recipient has to add the sender name to her buddy list before the message can be sent. In contrast to the open nature of email, this is a more closed system, somewhat Victorian in its nature—you cannot talk to someone until you have been introduced. It is easy to conceive of totally closed communities that don't even utilize global names. However, a system that does not use global names cannot easily merge two user communities later, even if they desire it, because of the possibility of name conflict.

There are, of course, proposals for controlling spam that don't involve filtering based on identity. But, since identity is an important part of many applications, it is worth looking a little more deeply at the issues there.

4.2.4.1 Managing identity

A system that is not totally open needs some sort of identity verification or authentication. If one sender can masquerade as another, then controls on who can send are of little value.

One approach is pair-wise identity management, in which each recipient maintains the information necessary to authenticate all senders it will accept. This is reasonable where the pattern of communication is limited — the set of senders for any recipient is small. This pattern is the descendent of the era when a user would have a different password for any system he could log into (and the process of "introduction" often involved going and talking to a person to present real-world credentials in order to get a user name and password). When the number of pair-wise connections is great, having each recipient maintain the identity verification information for all senders is a burden for both the recipient and for the sender (who has to remember or maintain the necessary information for each recipient).

Another approach is to move identity verification to an ALS that is provided as part of the application. Most IM systems work this way: a participant's identity is validated by a central service before he/she can connect to other participants. Many multi-player game systems also work this way: identity is maintained in the game server, along with attributes of that identity, all of which can thus be vouched for by "the game".

The end-point of this process is *shared identity*, in which identity management is removed from the application and operated as a free-standing service available to multiple applications. Examples of this include the Microsoft Passport system and the competing Liberty Alliance Project. The use of SSL in the Web is an example of a solution with a mix of these features. The certificate hierarchy is available for use by any application, and the providers of browsers imbed a table of "valid" certificate authorities into each browser so that the user does not have to maintain this data for himself, but each web site may require a login from a user (a pair-wise solution), and only very few Web sites use personal certificates to validate the user to the server.

The tradeoffs among these schemes are well understood and much debated. The annoyance of having each user maintain multiple identities, one for each partner, must be traded off against the possible loss of privacy that comes from having a single identity that is shared across many of the different activities that a person engages in, a fear that is compounded when the identity is maintained by a large corporation whose motives may not be clear. The point here is that the designer of an application must be sensitive to this tradeoff and make a thoughtful set of design decisions. Different outcomes bias the tussle among different parties and will be more or less suited to different situations.

Of course, just having a verifiable identity does not solve the previous question of who can talk to whom. One of the advantages of building a shared identity system is that a shared identity can provide a basis for shared reputation maintenance systems, which are a subject of research today.

4.2.4.2 Hiding identity

Identity is something that one end reveals to the other as part of establishing communication. At the same time, one end may want to hide aspects of identity from the other end, revealing only that which is necessary in the context of the application. Which aspects of identity can be hidden and which must be revealed is a tussle space, of course. Many web sites will offer free services, but the "price" is that the user has to reveal personal information. Sometimes what must be revealed is a matter of law. In the US, it is illegal to send a fax without including the correct sending telephone number, and recent spam legislation requires that bulk mail have a valid return address. On the other hand, some chat groups don't reveal the sender's full email address to the group, so it is only possible to reply to the group, not the sender. Of course, the server that runs the group knows the sender's full email address.

In general, passing communication through an ALS allows more control on what each party reveals to the other. One benefit of having IM messages go through a server is that the IP address of each party need not be revealed to the other. An IP address may not seem like an important part of identity, but knowing an IP address opens up that machine to various forms of exploration and harassment. Opening a direct connection necessarily reveals the IP address of each party to the other, unless some NAT service can be interposed.

Finally, any discussion of revealing and hiding identity must discuss the importance, in some applications, of specifically allowing anonymous participation. There are often very important social and economic benefits of allowing anonymous behavior, but since it is hard to police a world with no accountability and there is little accountability without identity, systems that permit anonymous operation may have to impose much more restrictive technical constraints up front, to prevent rather than punish misbehavior.

4.2.4.3 Naming application entities

In some applications like email, IM, and VoIP, the entities that the applications deal with are people, so the only namespace of importance is the namespace of users. For other applications, such as the Web or content sharing, the application deals with information entities, and there must be some way to name these entities as well as the users. The Internet provides a low-level naming scheme, the Domain Name System, or DNS. This is a system that allows a machine to keep the same name even when its IP address changes. But with certain exceptions (e.g., the MX mail records), the DNS is intended to name computers — physical end-points attached to the network. At the application level, it is very seldom a physical end-point that really needs to be identified. If the application hooks people together, for example via Internet telephony, instant messaging, or teleconferencing, the application needs to name people and the problem is to find their current location. If an application provides access to information, the application needs to name the information objects and the problem is to find a legitimate copy. If the copy is authoritative, it does not matter where in the network it comes from.

Some application designers have been tempted to use the existing DNS to name application entities, even though it may not be optimal. Thus, the Web embeds DNS names in URLs. This approach had the advantage that the Web could be launched without building a whole new name resolution mechanism first, which might have doomed it. It has the disadvantage that when the DNS name for a Web server changes, the served URLs change, so names for information objects may have less permanence than might be desired.

Many modern applications build their own directory to catalog their objects and use the DNS only as a way of finding the directory. Music sharing programs don't put the names of tunes into the DNS, nor do instant message (IM) systems put people in the DNS. Instead they use an application namespace that is tailored to its requirements, at the cost of more complexity in the design.

In these examples, the directory and the lookup service is designed specifically for each application, so there is no sharing of mechanism among applications beyond the DNS. The obvious next step is to design a mechanism for naming higher-level entities in an application-independent way. The Session Initiation Protocol (SIP) is one such example. It was conceived as a protocol to set up Internet phone calls, but it has been used to set up multi-player games, and other purposes besides VoIP. It would typically be used to name people. The URI working group of the IETF designed a location-independent and application-independent name space for objects [URI], and CNRI has developed the Handle System to provide persistent names for digital objects on the Internet.[8] So there are number of options for entity naming other than complete design from scratch that are open to application designers today.

One end-point of this design process would be to pull application-level naming down into the network layer. Gritter and Cheriton [Gritter2001] have proposed that individual information objects be given IP-style identifiers, and that the routing mechanisms in the Internet be used to compute a path to the closest copy when it is requested. This raises a variety of issues, from the size of the routing table to the concern about whether ISPs (and the Internet layer generally) should be given the total control over the algorithm for finding a desirable copy of an object.

[8] See http://www.handle.net/

4.2.5 Robustness against attack

If an application becomes successful and society starts to depend on it, then it will become a target of attack. An application designer should think about how an application can be robust enough to be "mission-critical".

Many of the issues about attacks will be specific to what the application "does", of course, and are outside the scope of this discussion. And issues of poor implementation and the resulting low-level vulnerabilities should need no more than passing mention. But there are some issues that apply in common across applications.

4.2.5.1 Rendezvous in a threatening world

One aspect of hiding one's identity (or in fact one's existence) is to put all of one's machines behind a Network Address Translation (NAT) box. Machines that don't have public IP addresses are better shielded from probing and attack. However, the use of NAT boxes raises an important question for any application designer—how can communication be initiated between machines when none of them are visible on the net? This so-called *rendezvous* problem is becoming more critical as more and more of the end-nodes on the net vanish behind NAT boxes.

Rendezvous refers to the process by which two machines that want to interact in the context of some application identify each other, make initial contact, and successfully establish communication. Today, the simple model is that one machine gets an IP address for another and sends an initial packet to that address. But as noted above, more and more machines are hidden behind NAT boxes and don't even have a public IP address. And many users are uncomfortable revealing their IP address to others, even though they are willing to have some restricted interaction with them.

An ALS architecture can solve these problems. IM systems that use a central server can work even if all the machines are behind NAT boxes. Each machine makes an outgoing connection to the server, and then the server passes data back and fourth between the two. But it is much trickier, if indeed possible in general, to build an end-node-only application that can work when all the machines are behind a NAT box. Kazaa, for example, solves this problem by finding a participant that is not behind a NAT box and causing it to become a relay between the two hidden hosts. Rendezvous in the presence of NAT boxes is a current topic of discussion in the IETF.

The problem of session initiation, or rendezvous, may be handled by a set of common ALSs in the future, as an alternative to an application-specific server architecture.

Finally, rendezvous is a process that may need to be better protected from prying eyes. Due to the design of IP and the DNS, rendezvous uses public packet headers. The original DNS did not catalog services on a host, but just hosts themselves; services on a host are identified by "well known ports" that are statically assigned to applications. The well-known port visible in every packet reveals what application it is a part of. This allows ISPs to monitor what applications their users are using, block certain applications or deflect them to servers of their choice, and so on.

A recent extension to the DNS [RFC2782] allows the registration of a service on a host, not just a host; looking up a service name returns a port as well as an IP address. This could be exploited to modify the balance of power in the tussle world, by returning a random port number unique to the host rather than a universal well-known port. If the Internet worked this way, the balance of attack and defense around ports would totally change. A firewall could no longer easily block or allow specific applications for all hosts by blocking port numbers. On the other hand, port scans would be must harder to exploit, since finding an active port would give no hint as to what application was behind it. Since server ports could be picked from the full 16-bit port space, port scans would be very inefficient, in any case. ISPs could no longer track or block application usage. This might make network engineering much harder, since one could not track the growth of specific applications. On the other hand, ISPs might be more motivated to offer different QoS as part of service stratification, as opposed

to blocking or enabling different applications. Finally, it would be straightforward for several machines behind a NAT box to offer the same service, since they would not have to "share" the same well-known port.

4.2.5.2 Data visibility

There was little encryption of data in the early days of the Internet. This raised the risk of unwelcome disclosure but made "benign peeking" possible, for debugging, delegation of function to servers inside the network, and so on. With increasing concerns about disclosure, some applications have incorporated application-specific mechanisms like SSH to provide encryption under specific circumstances. And application-independent approaches like SSL are quite popular today, even if encryption is not the norm in the Web.

Some application designers have concluded that the decisions about the need for disclosure and integrity controls are not a part of the application, but a consequence of the relationship between the parties, the sensitivity of the information, and so on. In this view, the decision to use encryption should be handed off to a lower-level mechanism such as IPSec or VPN tunnels. This may be satisfactory in cases where both parties have compatible and easy means to activate such tool, but it may fail on grounds of complexity and difficulty of use.

Doing encryption right, including key distribution and protection, is a specialty that many application designers are not trained for. It is good that encryption has reached a stage of maturity that makes it available as a stand-alone system that application designers can incorporate. However, it is not clear to what extent encryption can always be extracted from the application. If an application wants to encrypt some but not all of a communication, for example, the lower-layer schemes like IPsec and SSL will not be useful.

An important tussle is the relationship between hidden content and content filtering. Encrypting the contents of messages limits the filtering that a third party can do. Among trusting parties, this limitation may be exactly what is desired—for example to sidestep government censorship. But among parties that don't trust each other totally, filtering may be desirable. Users may want to have their email scanned for viruses, and adults may want to filter their children's communication to prevent the receipt of objectionable material. Since the wishes of the different parties to the communication may differ, it is complex to sort out how the application should actually function in these cases. It is a question of who is in charge, how that fact is expressed, how the application responds to it, and whether the resulting behavior can be seen and understood by all the parties, or whether parts of the action are hidden.

4.2.5.3 Denial of service

Denial of service (DoS) attacks and their more aggressive relative, *distributed denial of service* (DDoS) attacks, attempt to overload a machine to the point where it cannot provide service to legitimate users. Many of the attacks today have focused on lower layers, such as TCP, and attacks on the application should be considered in that context.

The nature of most attacks on TCP is not just to overload the processor, but to exhaust some limited resource, such as the storage for TCP connection state information. The goal is to trick the TCP software into expending effort to set up needless state, which is then maintained for some time to the exclusion of legitimate requests. One of the factors that make attacks at the TCP level worse is that TCP has to set up a connection descriptor on receipt of the first packet, before it knows anything about the identity of the sender or even if the sender's IP address is legitimate. Many attacks at the TCP level involve false source addresses.

One way that these problems might be solved is to let the application provide hints to the TCP layer about what source IP addresses are more or less likely to be valid. The application might keep track, for example, of what IP addresses had ever been seen before, and instruct the TCP layer to give them

higher priority in times of overload. This idea might be helpful, but would require a redesign of the lower levels or the preprocessing of incoming traffic at the packet level before it even reaches the host running the application. This option is not easy to exploit today, but it is worth keeping in mind since it implies something about what information the application may want to store for later use.

If the TCP attack problems can be solved, we can expect that DOS attacks will be directed at the applications themselves. How might an application be made more resistant to these sorts of attacks? In general, the same reasoning applies—avoid a design that requires the application code to invest much effort or create much state until it is sure that it is talking to a legitimate partner. The Web actually has a tool that is helpful here, although some users view it with suspicion: cookies. Cookies allow a user to show that the other end previously knew him, and to send the credential in the first message sent. It is worth thinking about how mechanisms such as cookies can be made robust to forgery, replication and theft.

Another class of defense is the so-called "puzzle": a challenge sent back to the client that the client must solve and return before the server will take any significant action. The idea is to demand enough effort of the attacker that he runs out of resources before the server does. The application designer has an advantage here over the TCP layer. TCP must generate some sort of response upon receipt of the first packet, but the application will not receive an incoming message until the TCP connection is established, which implies that the source address is valid. Even in a DDoS attack that uses many machines to overload the target, the source IP addresses in each of the attacks must be valid, so many attacks from the same machine can be detected as such. Of course, puzzles and related mechanisms have the effect of adding round trip delays in the connection initiation, which is contrary to most design objectives.

Another theoretical means of protection is to make a sender include in the first message a micro-payment that the recipient can refund if the sender proves legitimate. This implies a framework for micro-payments that we lack today, but it is an interesting speculation.

4.2.5.4 Defense in depth

One way to an overloaded server is to permit the server to "outsource" protection to other machines. This is an example of "defense in depth", applied to the problem of denial of service. If the incoming load of requests can be pre-processed or spread out among a number of machines, a DoS attack may be diffused and rendered ineffective.

Caches and replicated content provide a form of defense-in-depth. Since the content is replicated in many places, an attack on any single machine may disrupt a subset of users but will not disable the overall service. An application designer should consider which aspects of an application need to be centralized, which can be replicated, and how preprocessing or preliminary interaction with the source can help separate attacks from legitimate requests.

Of course, if there is a central server in the application design, it can still be vulnerable to a direct attack if the attacker can bypass the first line of defense. If the IP address of the server is known, then only low-level restrictions on routing can prevent a potential attacker from sending directly to the server. At this point, the options becomes more complex: either the server only accepts connections that come indirectly from one of the pre-processors, or routing is restricted so that all packets have to pass through some checkpoint where legitimate packets can be distinguished in some way. There are proposals to create a packet-level mechanism that works this way, for example I3 [Stoica2002]. But for the moment, the application designer is on his own.

4.2.5.5 Attack amplification

One vulnerability to avoid if possible is a design in which one node in the application can be subverted and turned against other nodes. In particular, it is critical to prevent an attacker from using a node as an amplifier for an attack, so that a successful attack on one node becomes a subsequent attack on

many. In general, this requires an analysis of application level behavior, to detect places where incoming requests from one node turn into outgoing requests to other nodes. Mailing lists suggest the potential for amplification, as do any other sort of "one to many" patterns. Email attacks that use the address book of a victim to pass themselves on to others are a real concern today, and lead to analysis of attacks in terms of propagating epidemics. One class of protection, which may have as much to do with implementation as design, is to require that a human be in the loop for actions that seem to imply amplification. This will have a slowing effect, as well as perhaps detecting the attack. Limits on outgoing rates can also help, provided they can be set up so they do not interfere with normal operation.

4.2.6 Performance adaptation.

An application designer must take into account that users may attempt to use the application under very different circumstances, in terms of performance and computational capability. Some of these differences may seem to be outside the scope of this document—whether the user has a small display or a large one, a keyboard or a speech input, a powerful processor or a feeble one, and so on. But some performance issues are directly related to the network—whether the user is on a fast or slow connection with high or low latency, and so on. Some of the issues above relate to topics of interest here.

4.2.6.1 Caching and pre-fetching

If there is a high-latency, slow, or congested link along the path from the source to the destination, the application may benefit if content can be pre-staged or cached near the user. Then the user experience can avoid these impairments. This approach makes less sense for applications that require real-time interaction, access to real time data or to another person; they make more sense with information that can be cached or with human communication, like email, that can be staged.

To implement a caching or pre-staging approach, several things are necessary:

- The application must support some sort of caching or pre-fetching in its architecture.
- There must be a set of servers positioned to perform this function.
- The application must be able to figure out the right locations for these servers, and which server to use for a given request.

The Internet email system has a full relay architecture built into it, but the design is not focused on the goal of staging the email "close to" the recipient. In fact, the POP server can be, and often is, at a great distance from the user. The email design was focused more on reliability, the management of the server, and do on. The Web offers examples that more directly address performance and locality. Many access providers have web content caches in their sites that can serve up web content without having to retrieve it from across the net. The firm Akamai has made a business of pre-positioning content near users, and dynamically computing which of the various Akamai servers is the best, in terms of performance, to provide the content.

4.2.6.2 Reformatting of content

If the user's computer is particularly limited in processing capacity, display capability, and so on, and/or if this computer is at the end of a particularly constricted communication link, the application may be designed to permit the pre-processing or reformatting of incoming messages before they are sent over that slow link to the user. This concept is equally applicable to real time applications and to those where content can be cached or pre-staged. Examples of preprocessing include reducing the resolution of a video image, mixing several audio streams to produce a single one, reformatting a web page for a small display, and stripping attachments off of email and truncating long messages.

If the set of reformatting operations has been standardized, they can be implemented at the source for the benefit of the destination. But since new devices are introduced from time to time, there is a benefit of allowing new sorts of conversions to be implemented by placing them in new servers designed for the purpose. This approach, in the extreme, allows an application to extend beyond the Internet itself, to devices and end-nodes on different sorts of networks. In this case the relay point reformats the whole interaction, not just the data.

4.2.7 Economics

The common economic model of the Internet is a simple one: the user pays a single access fee (e.g. a monthly flat rate) and expects in return to get access to all the applications and services of the network. The user does not expect to pay extra for email, Web access, and so on. This is not totally true, of course; users understand that they have to pay for things like Web hosting, some third-party email services, and some content. But access to most applications comes "for free" once the user signs up for the Internet.

This bundled approach makes a lot of sense as a pricing strategy in the market—the consumer might not tolerate a plethora of small bills for specific services. On the other hand, it raises a critical question: if parts of an application are running on ALSs, as opposed to simply running on the end-nodes, who is providing those services, and what is their compensation?

The DNS is seen as so important that all ISPs just support it as part of their core business. Similarly, email is seen as so central to the net that essentially all ISPs provide POP and SMTP servers as a basic cost of doing business. But beyond that, there is little clarity or consistency in the economic structure of applications. ISPs appear to provide Web proxies both because they improve the user experience and because they save the ISP cost by reducing traffic on the trunk into the Internet. Akamai servers are paid for by the content providers, both to improve the user experience and to reduce the demand on (and thus the capacity and cost of) their primary servers.

For more recent applications, where there is no history of free use, fees are starting to emerge. Internet telephony or VoIP requires rather costly servers to transfer calls into the circuit-switched telephone systems. As a result, most VoIP services, such as NetToPhone or Vonage, have either a monthly or a per-minute fee. It will be interesting to see if SIP servers become part of the infrastructure that ISPs provide as part of their base service, or come to be a service that requires an incremental fee.

Games provide a number of examples of economic models. Some game developers sell the software and run a server as a part of the product, others give away the software but require a subscription to the service, and others use an informal scheme in which some of the players become temporary servers, thus removing the need for the game provider to provide any servers to support the game.

In this context, the emergence of the peer-to-peer systems is significant. P2P systems are based on a different economic model, which is a barter system. Nobody pays anyone else in money; each party just contributes part of the service. This is actually a very good system in some cases, although its use for the propagation of copyrighted content has given it a bad name.

4.2.7.1 Design for making money

Most Internet applications do not have a built-in model of cost recovery, commercialization, or profit. A catalog of the economic motivations that drive application providers on the network reveals a variety of approaches to making money.

- Many Web sites offer free content in order to sell advertising.
- Some proprietary applications give the "viewer" away in order to generate sales for servers.
- Some sites charge direct subscription fees or sell individual units of content for a fee (including iTunes and the New York Times archives.)

It is not unreasonable for service and content providers to make money, notwithstanding the Internet history of "free" applications. This reality begs an important question—should application design take into account the economics of the service, or is the business/economic context something that can just be wrapped around the protocols after they are done?

Building a complete model of pricing and billing into an application's architecture is almost certainly a very bad idea. First, there will always be cases where the application is offered for free—for example inside a corporation to its employees. Second, marketing departments will always want to invent new sorts of pricing models—fixed vs. incremental payment for service, site licenses, and so on. But are there "pricing building blocks" that might be built into an application that would facilitate commercialization of applications? This topic has received very little attention, and what attention it has received has been hostile. There is a sentiment, quite possibly justified, that if it is easy to bill for an application, then providers will do so, which will erode the "bundled pricing" model that seems to work so well for the Internet.

Despite these concerns, it is not wise simply to ignore this set of issues. By doing so, designers lose the opportunity to shape how the economics of an application matures. Specifically, if there are no "economic hooks" in the protocols, service providers may try to exploit other features of the protocol to change the economic landscape, to the detriment of overall operations. Here is one example, hard to prove or document but rumored none the less. Some web hosting providers, since they are paid based on the number of hits, may be marking static web pages as "uncachable" to keep downstream providers from holding a cached copy. This phenomenon represents a tussle between two providers with different economic motivations—one wants to reduce the number of hits to the upstream copy, one wants to increase it. In this case, a correction might be to take information such as "uncachable" and imbed it into the data itself in a way that cannot be modified without detection, so that the setting represents the opinion of the creator, and not of an intermediate server. But it may take more experience than most of us have to think of these sorts of things in advance.

One specific consideration is direction of value flow. The phone system has two sorts of long-distance calls, those where the sender pays, and those, the "800" calls where the receiver pays. One of the problems with the Internet is that one cannot tell which way the value flows just by looking at the flow of packets. But knowing whether the sender or the receiver is prepared to pay for an action might help sort out a number of billing problems. Note that specifying which way value flows in no way sets the price. That can be done in a separate transaction.

4.2.8 User empowerment—tilting the playing field

The list of possible ALS functions in Section 4.2.2 reminds us that application-layer servers in the network represent a rich space in which the tussle of interests will be carried out. The application designer, by crafting the server architecture of the application, is crafting the tussle space that will emerge as the application emerges. Many network designers and application designers share a value that has been called user empowerment, or freedom of choice and action, or an open marketplace. All of these terms try to capture the idea that the Internet is a place where users can run the applications they choose and pick the providers on whom they will depend; it is not a place where monopolists gain the power to impose restrictions on user action. Of course, this is a tussle space, not a given right, but many designers see the individual user as potentially at the mercy of "big business", or "repressive government", and tend to make design choices that favor the end-user when they can. We need techniques to tilt the playing field toward the end-user as we design a server architecture.

For the application designer, user empowerment is the ability of the user to pick the server(s) to use and to avoid the server(s) he chooses to avoid. Implementation of that empowerment is governed by a set of simple rules concerning what is visible, hidden, and protected in messages.

- That which is encrypted (or missing) cannot be seen.

- That which is signed cannot be changed (without being detected).
- That which cannot be seen cannot be acted on.
- Aggregated content whose aggregation boundaries cannot be seen cannot be treated selectively.

Consider a simple example—a user trying to send a piece of mail. Normally, the user picks the SMTP server he wants to use. Giving that choice to the user seems right to most of us. But recently, there have been cases where an access ISP (for example, a provider of residential broadband service) prevents the user from connecting to any SMTP server but the one the ISP provides. There may be claims that this is for "reliability" or some such, but the practical reason seems to be to restrict the form of the "from" email address being used to only those provided by the ISP. By doing this, users are prevented from using a third party email service or their corporate email address, and they may be persuaded to upgrade their service to a more expensive version. One response by a crafty user is to use an SSL connection on a different port rather than raw SMTP, to prevent the ISP from seeing what is going on and blocking it. Of course, the ISP could block SSL connections, which leads to further escalation. There are many applications today that are being designed to work "over the Web", either as an option or as the only transport mode. This may be seen simply as a demonstration of horrid design taste, or as a crafty way to commingle this application traffic with a class of traffic (http) so important that any ISP or firewall more or less has to let it through. By encrypting and aggregating traffic, the user can attempt to impose his choices of servers, despite the motivations of his providers.

All this has been worked out for email in an ad hoc manner, long after the basic email protocols were designed. But any application designer today should think through how this might work out for his application in the future.

- Should the application have a native "encrypted mode" that the sending application can easily trigger? If so, how are the keys managed? What is the relation between the key that links an end-point to a server and a key that links end-points?
- Are there parts of the message that a server may want to change (for good reasons), and other parts that ought to be signed to prevent changes?
- Should the application have a "Web mode" or other aggregated mode that commingles the traffic with that of other applications?
- How does the user "get started" in using this application? How does he find the first servers he needs? Does the access ISP provide the server (as it does the DNS), is there is a central service run by a third party (such as the application provider), or does the user manually configure his software to select a server of his choice, perhaps using information that was advertised on the Web (as happens with some email services)? Could this process of selection be automated as part of the application design?

4.2.8.1 Receiver empowerment

All the discussion to this point has been from the viewpoint of the sender. To first order, it is the sender that picks the recipient. But of course, this is too simplistic. Consider mail: the receiver picks his "incoming" mail server and configures the DNS so that the sending server connects to this server. Consider the web: queries to URLs can be deflected by the receiving end to one or another server in a number of ways. The user can be given a different URL based on where he is coming from, or the DNS that defines the URL can be programmed to give different answers based on the current circumstances. Again, most examples of this today are being retrofitted onto applications where the idea was originally missing, but a designer today ought to consider what options should be installed in the application itself.

There are also proposals to control routing at the network level to meet the needs of receivers—schemes like I3. [Stoica2002]

4.2.9 Conclusions

4.2.9.1 Application design techniques/goals

- Try to sort out the motivations of the parties that may participate in the application.
- Consider the importance of user empowerment: Let the users pick who they want to communicate with and what servers they use.
- Use encryption as a tool to control what can be seen, what can be changed, and what is hidden. Think about the needs of trusting and wary users when making these choices. Even trusting users may benefit from an application architecture that involves servers, which may imply revealing certain parts of the communication. Consider if aggressive use of information hiding may cause the application to be banned in some contexts.
- Design for the life cycle of an application. Design so servers are not needed with small numbers of users, but can be added as application grows up. Consider using a peer-to-peer architecture as a stage of growth and a fallback mode, not as an immutable feature of the design.
- Include full application entity information in the messages themselves, rather than implying part of it implicitly by the Internet destination, so that servers can participate in the mechanism. Encryption can hide this information from prying eyes.
- Do not create a new form of spam in a new application. There is no reason to make it easy for unwelcome users to intrude on others, but be aware that they may have a strong economic incentive to do so.
- Do not provide an unwitting tool for attack amplification.
- Include a concept of identity, and consider giving the receiving user control over how restrictive he wants to be in what he accepts. Sometimes, anonymous communication is socially or economically valuable.

4.2.9.2 Some useful common services

Most of the common services listed here have a strong social component, and will be the target of tussle and debate. We do not mean to minimize any of that, but to invite the debate in an orderly and coherent manner by calling for the services.

- A service to manage identity and support cross-application reputation management, non-repudiation of identity etc.
- Naming services for different sorts of entities, such as SIP, URIs and the Handle System.
- RFC2872 service lookup to avoid well-known ports.
- Encryption layers that provide different sorts of information hiding and signing.

4.2.9.3 Possible changes to the network architecture

Provide:

- New modes of routing to support applications.
- Routing to an identity that is not an IP address, to protect the address and control the traffic to the destination.
- A means to link application-level identity to packet-level identity, so that packet-level filtering and routing can be controlled by receiver specification.

- A service that provides abstracted information about network topology and performance, to assist application with locating servers within the application.

4.3 Mobility

The Internet architecture defines communication between network attachment points, where each such point is labeled by an IP address. In this abstraction, "mobility" refers to an end system or network that changes its point of attachment dynamically. The architectural objective of (IP) mobility support is to provide unbroken application communication when the IP address changes.

Note that the point of attachment might have changed because the end system physically moved, because some segment of the network was renumbered, because the user got a new temporary IP address from a DHCP server, or because the user switched ISPs. We can subsume all these in 'generalized mobility".

The current Internet architecture assumes that the points of attachment, and therefore the IP addresses, are normally an invariant during the lifetime of application communications between end systems. The Internet community has developed additional mechanisms, in particular Mobile IP [MobileIP02], to handle mobility as an exception. Mobile IP assigns an invariant IP address ("home agent address") to a mobile node, and then forwards packets via encapsulation. It implies extra complexity and overhead

In reviewing this architecture, the NewArch project considered an alternative perspective: suppose mobility were architected as the normal case? The project designed a prototype of such a protocol in the M-FARA architecture (See Section 3.1.5). It involves the necessary use of rendezvous agents within the network for mobility.

The consequence of this new perspective seemed to be that the overall design might be somewhat simpler in principle, but that the overhead cost for mobility would be paid even in the non-mobile case. Even though mobility is expected to become increasingly important, we expect that the majority of Internet service points will remain statically connected to the network. Universal mobility therefore did not seem to be a good trade off; we concluded that the current approach, treating mobility as an exception, is probably the wisest engineering compromise.

4.4 References

[CWSB02] D. Clark, J. Wroclawski, K. Sollins, and R. Braden, *Tussle in Cyberspace – Defining Tomorrow's Internet*, Proc. ACM SIGCOMM 2002, August 2002.

[Gritter2001] M. Gritter, D. R. Cheriton, *An Architecture for Content Routing Support in the Internet*, Usenix Symposium on Internet Technologies and Systems, March 2001.

[MobileIP02] C. Perkins, Ed., *IP Mobility Support for IPv4*, Internet RFC 3344, August 2002.

[RFC2782] A. Gulbrandsen, P. Vixie, and L. Esibov, *A DNS RR for specifying the location of services (DNS SRV),* Internet RC 2782, February 2000.

[SIP] J. Rosenberg, et. al., *SIP: Session Initiation Protocol,* Internet RFC 3261, June 2002.

[Stoica2002] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, S. Surana, *Internet Indirection Infrastructure*, Proc. ACM SIGCOMM 2002, August 2002.

[URI] T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifiers (URI): Generic Syntax,* Internet RFC 2396, August 1998.

5 Appendix A: Architectural requirements

To guide its efforts, the NewArch project first examined the high-level requirements on the Internet architecture (IA).

5.1 Original primary requirements

Four of the original Internet architecture requirements -- multiplexing, survivability, service generality, and diverse subnet technologies -- were listed in Clark's 1988 paper as primary.

(a) Multiplexing

Clark identified "multiplexed utilization of existing interconnected networks" as the primary goal of the architecture. Multiplexing is necessary for economical communication of multiple logically-independent data streams across a common transmission medium.

The interconnection of existing networks was required because it was not feasible or desirable to re-engineer existing network technologies, including LANs, terrestrial links, wireless links, and satellite links, for example, into a single unified link layer. Thus, the Internet must be a network of (sub-) networks.

(b) Survivability (robustness)

The Internet must be inherently robust against the failure of gateways (routers) and other network components. Within the military context of early Internet research, this goal was called "survivability": communication must continue as long as there is any possible path.

This requirement implies that the network must be able to dynamically adapt to failures of network hardware or software. It also favors the use of network protocols that are in some sense "self healing".

(c) Service generality

This goal was to support the widest possible range of applications, by supporting a variety of types of service at the transport level. Services might be distinguished by speed, latency, or reliability, for example. Service types might include virtual circuit service, which provides reliable, full-duplex byte streams, and also datagram service, which delivers individual packets with no guarantees of reliability or ordering. The requirement for datagram service was motivated by early ARPAnet experiments with packet speech (using IMP Type 3 messages).

Another possible type of service would be isochronous -- data arriving on a fixed time cycle. Isochronous service was never a requirement for the Internet, possibly because its designers believed that buffering cures all ills.

(d) Diverse subnet technologies

The Internet has to interconnect subnets that use a variety of technologies with a wide range of characteristics. They vary in geographical extent and in performance parameters like bandwidth, delay, error characteristics, and MTU (maximum transmission unit). They include point-to-point networks and NBMA (non-broadcast multi-access) technologies. Support for diverse technologies is required for universality, and it also enables the evolution of networking technologies without updating all network elements simultaneously.

The fundamental conclusion from this requirement is that coping with heterogeneity is fundamental to the Internet.

5.2 Secondary and later requirements

We now list five additional requirements, which were either considered secondary in the early days or were introduced later.

(e) Scalability

Although "the network must be scalable" is a mantra, network scalability is seldom defined with any precision. The major dimensions of growth in the Internet are (1) the number of end systems N, (2) the maximum bandwidth Bmax, and (3) the maximum end-to-end delay Dmax.

The most noticeable scaling issue in the Internet has been the continued exponential growth of N with time, resulting in repeated growth crises for the network infrastructure.

The criticism that "protocol X does not scale" has often been used to mean that network resources scale in direct proportion to N. This is not N squared, for example; in most other circumstances O(N) scaling would be considered ideal. However, since N in the Internet increases exponentially with time, O(N) resources also increase exponentially with time, which is awkward. If the scaling law for Internet resources can be made O(log(N)), then the total resources in the Internet will increase only linearly with time.

The rate of growth of Bmax has been a surprise, but on the whole the Internet protocols have handled this increase without significant problems.

The delay Dmax affects network dynamics through the delay-bandwidth product Bmax*Dmax. Growth in Bmax has created significant growth in this product; what was once a problem only for satellite links (large Dmax) is now a problem for terrestrial links (large Bmax). Some advances in transport protocol mechanisms have been forced as a result

(f) Distributed management

After the cutover to the Internet protocols in 1983, there was never a time when the entire Internet was under a single management, although the core of the original Internet was built and operated by a single contractor (BBN). The growth of NSFnet in the mid 1980s introduced a large number of administrative domains and a hierarchical topology. Multi-organization management became even more important when the Internet became commercial.

Another aspect of the distributed management requirement is that the network should be "plug and play"; it should be possible to build a larger internetwork by simply interconnecting smaller aggregates of nodes, without a centralized administration of the network.

(g) Security

The network hardware and software needs to support the common security functions such as privacy, integrity, and authentication. Security must be applicable to both the network infrastructure and to the end systems.

(h) Mobility

Mobility, in which a host changes its point of attachment dynamically, must be supported. Since changing the point of attachment implies changing the network address, host renumbering (re-addressing) creates some of the same technical problems as mobility.

(i) Capacity allocation

Although the original list included types of service, there was no architectural requirement for a capability to allocate capacity "fairly". The original Internet designers certainly believed in fairness as a social good, but that objective was subsidiary to the goals of survivability and of diverse subnet technologies.

On the other hand, due to the military origins of the Internet, an early goal was to support deliberate unfairness through precedence bits in the IPV4 header. Commercialization of the Internet created a need to explicitly allocate capacity unfairly under the influence of policy and/or price. Today's ISPs want to be able to sell different grades of service for different prices.

5.3 Non-requirements

There are also some important non-requirements for the Internet architecture -- accounting, ease of host attachment, and cost-effectiveness.

(a) Accounting

So far the Internet has not adopted a requirement for usage accounting or an architected mechanism for money flow. Whether or not this is a problem depends very much on your viewpoint. Economists tout the potential benefits of better resource allocation and competition as the result of direct cost feedback for resource usage. Others believe that charging for the network will distort usage and have other undesirable technical and social results.

(b) Ease of host attachment

Originally, there was a large overhead in attaching a machine to the network. It was recognized that this was a hindrance in the long run, especially as the network scaled upwards. Industry has addressed this problem and now attachment is not a significant issue. Every system comes with an Internet protocol stack and hardware and software for network access.

(c) Cost-effectiveness

In the early days there was considerable technical controversy over the seemingly large overhead (20 bytes for IP alone, or 40 bytes for IP and TCP) of Internet packet headers. However, cost-effectiveness was not a military requirement, nor was it important during the 1980s when the government was supporting academic networking. By the time the Internet became commercial, the rapid technology advances in computation power and bandwidth made the packet overhead issue largely a non-issue. The exception is restricted bandwidth in the "last mile", in which case TCP compression has been effective as a localized engineering optimization.

6 Appendix B: Architectural principles, old and new

This list attempts to parse the Internet architecture (IA) into logically independent components. It is certainly not a unique decomposition of the concepts, and a different decomposition might reveal additional insights. There may also be some disagreement on where the boundary falls between architecture and technology. Despite these limitations, this list should be useful for understanding the structure and scope of the architecture problem. This appendix assumes familiarity with the Internet protocol suite.

Some may disagree with our categorization of specific architectural features as "current" or not.

For example, this document considers int-serv and diff-serv to be part of the current architecture, while NATs, firewalls, and VPNs are considered to be outside the architecture. These choices may seem exactly backwards based upon real-world deployment, but they reflect our prejudices about what is "architected" and what is an ad hoc feature or a distortion of the architecture.

Section 6.1 lists the principles of the current architecture. Section 6.2 contains a parallel list that reexamines each of the current principles from Section 6.1 in the light of a possible new architecture. The discussion in the body of this report provides a detailed justification for some of the points summarized in Section 6.2.

We begin by noting an important meta-principle that impacts many technical decisions in the Internet architecture: "Avoid unwise optimizations":

> ***Internet design should favor universality, adaptability, and evolvability over efficiency or, in some cases, even functionality.***

6.1 Principles of current Internet architecture

We attempt to list the principle categories in very rough order, starting with the most fundamental. We further divide the list into two parts, primary and secondary. The primary principle categories establish the general architectural framework, while secondary categories fill in particular details. The specific principles are displayed in ***bold italics*** in the following discussion.

Primary Principles	Secondary Principles
1. Multiplexing: only Packets	12. Distributed Control
2. Transparency	13. Global Routing Computation
3. Universal Connectivity	14. Regions
4. Immediate Delivery	15. Mobility
5. End-to-End	16. Security
6. Loose Semantics	17. Resource Allocation
7. Subnet Heterogeneity	18. Minimum Dependency
8. Common Bearer Service	
9. Connectionless Network	
10. Global Addressing	
11. Protocol Layering	

6.1.1 Multiplexing: only packets

The Internet is based upon packet switching. This fundamental principle has implications for much of the rest of the architecture. For example, packets are the unit of data transmission across the network service interface to end systems (see 6.1.8), and packets must be supported by every subnet technology (see 6.1.7).

> ***The current Internet uses variable-length packets as the universal approach to multiplexing independent data streams.***

The advantages of packets were clear to the Internet designers. Packets provide a universal unit for error detection, error recovery, and multiplexing, and a granularity can be chosen that provides acceptable multiplexing delays at realistic data rates. Variable length was also an easy decision; a fixed packet length would have introduced an unnecessary design parameter that could not be optimized for a wide range of transmission media, and it would have created unnecessary fragmentation problems.

The Internet designers explicitly rejected *message switching* as an alternative to packet switching. A message is essentially an application data unit (ADU) whose size is set by the end applications (and is typically much larger than a single packet).

6.1.2 Transparency

The current Internet provides a kind of *syntactic* transparency—the behavior of the network is defined in terms of packet formats, with the expectation that packets are not modified in transit.

> ***In the absence of transmission errors, user data that is delivered to the intended receiver is delivered without modification.***

What comes out is what goes in. This basic "don't mess with my data!" principle is being threatened in today's Internet. For example, some web caches modify transmitted HTML data to attach advertisements, while NATs and firewalls may modify addresses within user data, for example, changing addresses in FTP commands.

6.1.3 Universal connectivity

Universal connectivity is the normal state; a host can send data directly to any other host, except when explicitly prohibited by a third party, e.g., by a firewall.

A variety of engineering techniques -- policy-based constraints on route announcements, constrained topology, firewalls, and VPNs – are used to limit connectivity. The usual reason for these connectivity constraints is to provide some (limited) security.

6.1.4 Immediate delivery

(a) Connectivity is continuous.

(b) In the absence of failure or overload, data is delivered immediately.

There should be no indefinite delays in packet delivery. The original architecture chose to exclude intermittent connectivity ("delay-tolerant networking" [Fall2003]), requiring that multiplexing rules preclude indefinite queuing delays.

6.1.5 End-to-End principles

The end-to-end principles, which resulted from the "end-to-end arguments" [SRC81], are fundamental to the Internet architecture although they have been partially eroded [Blumen01]. The original description of these arguments from 1981 is the closest thing that exists to a sacred text for the IA. The end-to-end arguments embody architectural reasoning about the appropriate location for functions -- the network vs. the end nodes.

(a) Generality: the network is built with no knowledge of, or support for, any specific application or class of applications.

In particular, the Internet designers were motivated by a desire to support realtime streaming media as well as connection-oriented applications like email and FTP.

(b) Robustness: The end nodes are responsible for communication functions that can be entirely accomplished by those end nodes.

This is the famous "*dumb network, smart end system*" principle. It arose in part from the relatively high cost of computer circuitry in the 1970s and the resulting desire to make the network nodes as simple as possible internally, and in part from the following:

(c) Fatesharing: State that is specific to a particular data flow between communicating end nodes is maintained in those end nodes. In case of a crash, the loss of such state will be coincident with loss of the communicating applications.

Following are three example applications of these principles:

- *Reliable Data Delivery*: End-to-end reliable delivery of data can be accomplished entirely by the end nodes using acknowledgments and retransmissions. Hence, responsibility for reliable delivery rests ultimately with the end nodes.
- *Network Buffering*: Speed variations between end nodes can be entire handled by end-node buffering and flow control. Hence, the network has no internal buffering mechanism to deal with end-node speed variations (although internal buffering is needed for end-to-end congestion control).

- *Format Conversion*: The end nodes can perform any required format conversions, so the network has no architected mechanisms for this function.

6.1.6 Loose semantics

(a) The IA contains no careful definition of the end-to-end semantics of data carriage in the Internet. The transparency principle (6.1.2) implies that what goes in comes out; deviation from this is ad hoc.

(b) The Internet has no model of its own performance. A host that needs to determine the performance of a path must measure it for itself.

This fuzzy definition of the communication service delivered by the Internet was not an oversight. In fact, it has been a key component in the early success of the Internet, enabling technology evolution, adaptation to extreme growth, and satisfaction of new service requirements.

6.1.7 Subnet heterogeneity

(a) The Internet makes minimal assumptions about subnet (i.e., link layer) functionality, in order to operate over the broadest possible range of subnet technologies.

In this principle, the phrase "minimum assumptions" does not mean *no* assumptions. At the minimum, a subnet technology used in the Internet must support the delivery of packets (see 6.1.1), i.e., it must carry a byte stream and support synchronization of bytes and packets. In fact, the current Internet architecture makes additional demands on the subnet, for example:

- TCP performs badly if the subnet packet loss exceeds a few percent; the solution has been to enhance the underlying subnet reliability when needed.
- The requirement for QoS in the Internet [IntServ94] implies QoS mechanisms within subnets.
- Subnet technology is allowed to be sufficiently packet-aware to "mess" with packets, e.g. to lose data on packet boundaries, or to reorder packets.
- The subnet must provide some mechanism to map IP addresses into link-layer addresses, e.g., the Address Resolution Protocol (ARP).

The accommodation of diverse subnet technologies is a bedrock IA principle, but it is sometimes threatened by demands that the Internet make effective use of enhanced functionality in subnets or that it use particular subnets more efficiently. Optimizations for particular subnet technologies may be harmful if they reduce future adaptability of the network as a whole (see the "Avoid unwise optimizations" meta-principle at the beginning of Section 6). Also note that subnet-specific optimization may be an exercise in futility in a world of exponential growth of processing power and network bandwidth.

6.1.8 Common bearer service

(a) The Internet provides a connectionless service" end-to-end.

To incorporate heterogeneous subnet technologies, the architecture defines a standard end-to-end network-layer service that has been called the "common bearer service" [CSTB1994, pg. 47]. It is implemented using the IP protocol. Furthermore, this service is connectionless; no setup is required from a host to send an IP packet (with best-effort service). Each IP packet sent by a host carries topologically-based addressing information (also called a "locator") to control packet delivery.

(b) The common bearer service provides at least the minimal common service, best-effort.

The service model is loosely defined as "*best effort*": packets may be dropped, duplicated, or reordered by the network. The services of reliable delivery, duplicate detection, and reordering are performed by the end systems, because they can be, in accordance with the End-to-End principle (see 6.1.5(b)).

The common bearer service does not try to hide diverse subnet characteristics from the end system. The architecture assumes that the end system will make whatever service enhancements are required by the particular set of subnets along the path and will adapt to the performance and other path characteristics that arise. That is, the subnet diversity is, in effect, passed up through the IP layer to the higher layers that are implemented (only) in end systems. This may be a distinct principle that is not quite necessitated by the end-to-end principles; it states (assumes, as an article of faith) that adaptation to diverse subnet characteristics *can* be done by the end systems. That it *should* be done by end systems follows from the End-to-End principles (see 6.1.5(b)).

(c) There is no access protocol for end systems (hosts)

The X.25 protocol suite used a common carrier model, in which the internal network protocols are decoupled from the access protocol used by hosts. The network end of each access link is an interface box called a DCE, while a host is called a DTE. The Internet did not use this model. Internet hosts use the access protocol of the subnet to which they are attached plus the common bearer service, which extends into the hosts (hence, it is end-to-end).

The IA makes two important exceptions to the connectionless service of (a):

(d) Source routing is supported as an escape when the default hop/hop routing computation does not provide a useful or desirable route, although it is not expected to be used normally.

(e) Flow-specific forwarding context may be established when users request network services that require such context (e.g., for explicit routing or QoS).

6.1.9 Connectionless network mechanism

The (inter-) network packet delivery mechanism is connectionless.

Packets are forwarded end-to-end through each subnet in the path, by connectionless IP packet switches called *routers*. The basic best-effort service uses no per-flow state ("context") in routers; however, routers contain routing state in the Forwarding Information Base (FIB).

Note that a connectionless end-to-end service does not necessarily imply a connectionless mechanism within the network. The ARPAnet provides a well-known counter-example: the ARPAnet provided an end-to-end connectionless (but reliable) packet delivery service, but implemented it internally using a connection-oriented mechanism. This approach was designed to perform allocation of a scarce network resource, reassembly buffers. The ARPAnet used connection setup between network DCEs but connectionless service in intermediate packet switches.

On the other hand, technological limitations in the 1970s made it desirable to minimize the complexity of packet switches. The ARPAnet design also violated 6.1.5(a), since it did not support streaming media. These considerations led to the connectionless network design of the Internet.

An Internet packet header is unaltered in transit (except for the TTL field and perhaps the TOS bits), and packet boundaries are preserved. This invariance is consistent with the global addressing principle (6.1.10) and the End-to-End principles (6.1.5), but it is not a formal requirement of the architecture. On the other hand, TCP header compression and other lossless compression techniques may modify the network payload across one or more hops while preserving the end-to-end payload.

6.1.10 Global addressing

(a) The Internet uses a single global address space to identify the network attachment point(s) of each node. Packet forwarding decisions are based upon these addresses.

The dependence upon a single global address space is fundamental to the IA. The IP address space provides a convenient globally defined space of *locators*. In practice, NAT boxes have eroded the global addressing principle. There is no global routing across NAT boxes, and soft state setup (a context for address mapping) is used to permit cross-region forwarding. NAT boxes depend upon the existence of a network core with global addresses (locators).

(b) IP addresses are also overloaded as end-system identifiers.

The overloading of the IP address has been valuable for (weak) authentication of the sender.

6.1.11 Protocol layering

Internet protocols are defined using layered abstraction. Layering is realized using a last-on, first-off "stack" of protocol headers on each packet.

Layer N presents a service to layer N+1 and constructs this service using the services of layer N-1. Layering provides a powerful model for building complex protocol systems; it provides modularity, abstraction, and information hiding. However, the strict layering model is often violated in today's Internet, so that the service provided by layer N depends upon information at layer N+1, although strictly it should be independent of layer N+1.

The other limitation of the layering model is that new functionality often leads to the definition of new sub-layers; for example: TLS is sublayer 4+, IPsec is sublayer 3+, and MPLS is sublayer 2+.

6.1.12 Distributed control

There must be no single point of failure in the network control mechanism, i.e., the network control mechanisms must be fully distributed.

This principle is required for robustness.

6.1.13 Global routing computation

The Internet performs a hierarchical, globally consistent routing computation to support loop-free hop-by-hop forwarding of packets based on the destination address.

Within the region of the Internet in which there is a single global network address space, there is also a global routing computation.

6.1.14 Regions

The Internet supports administrative regions (routing domains) for routing and limited policy control. A two-level hierarchical routing computation is performed, within and among regions

The concept of region in the IA is weak. In fact, early Internet research included no concept of administrative regions (and there was only one "provider", BBN). However, the need to upgrade routing protocols led to the addition of routing regions, and later NSFnet introduced administrative regions as well as policy-based routing to enforce acceptable-use policies. The IA still does not contain mechanisms to ensure that region boundaries are not compromised.

6.1.15 Mobility principles

The IA is optimized for non-mobile operation and uses a special-case mechanism for mobility.

6.1.16 Network security

(a) The Internet has no mechanism to constrain hosts that offer excess traffic. In particular, it contains no defense against DDoS (distributed denial of service) attacks.

(b) The Internet does not include an architectural approach to protect its own elements from attack.

(c) Link encryption is sufficient and efficient for Internet security.

These were the original security principles of the Internet, which were minimal. Link encryption is inadequate for many reasons, including the scale and the heterogeneous administration of the Internet; it is also (not coincidentally) a direct contradiction to the end-to-end principles.

When host attacks using the Internet first appeared, the network designers simply designated the attacks as a host security problem; the network was doing exactly what it was designed to do. However, the emergence of DDoS attacks has shifted the emphasis towards network security mechanisms

6.1.17 Network resource allocation

(a) A transport protocol in an end node is responsible for sensing congestion and slowing its transmission when appropriate.

(b) Transport protocols should be no more aggressive than TCP in the presence of congestion. However, there is no mechanism in the network to enforce this.

(c) Packet transmissions should be clocked by returning acknowledgments. For closed-loop congestion control, the round-trip time (RTT) is the fundamental loop time constant.

(d) The Internet contains sufficient buffering to allow a host to operate a congestion adaptation algorithm with a round-trip of control latency

Prior to 1985, the IA was clueless about congestion control. The four principles (a)-(d) above were introduced by Van Jacobson for TCP congestion control [Jacobson88].

(e) The Internet provides an architected mechanism for explicit Quality of Service.

Although it has not been widely accepted, Integrated Services [BCS94] provides this mechanism. Although it is still under development, differentiated service (diff-serv) provides a mechanism for network resource allocation for aggregated flows.

(f) The IA includes no architected payment flow mechanism.

Such a mechanism would allow service providers to be reimbursed for augmented QoS, for example.

6.1.18 Minimal dependency

(a) A minimum of network services and servers is required to support end-to-end communication.

In particular, two Internet hosts that know each other's addresses are able to communicate without network support (aside from forwarding by routers). This design was intended to provide additional robustness; for example, it allows communication even when the DNS is failing or unavailable. DHCP is a partial compromise of this principle, because it implies that a host may not be able to communicate at all until it contacts a DHCP server, but dynamic address assignment has a big payoff.

(b) The host/network interface is symmetric and there is no specific network access protocol, so that two Internet hosts can communicate directly without an intermediate router.

6.2 Principles for a New Internet Architecture (NIA)

We now repeat the principles list, presenting the conclusions of the study concerning a new architecture, indicated by the prefix "NIA" (New Internet Architecture).

6.2.1 Multiplexing: only packets

NIA: This principle would be unchanged.

The NewArch project considered allowing multiplexing mechanisms other than packets in particular regions of the network. For example, a data flow might be represented in packets in one part of the path but transmitted as a byte stream using FDM or TDM over another part of the path. The conserved end-to-end data unit might be a single byte or an "application data unit" [CT90].

However, the project concluded that allowing more than one representation and converting within the network in an application-independent manner is so complex as to be infeasible. It creates the need for per-flow state in the network, and it does not deliver greatly improved performance or functionality. However, in some circumstances it might be reasonable to do conversion of representations at an application-aware stateful conversion point, which knows what a particular application needs. In particular, an ALF-like architecture [ALF] should only be contemplated if the conversion points are application-aware.

6.2.2 Transparency

NIA: The network will continue to support totally transparency at the wish of both ends, but it may interpose constraints when requested by the end points.

The transparency principle implies that the Internet is purely a communication mechanism, but a number of forces are starting to introduce network-based services in middleboxes. One motivation for such network-based services is security, as described in Section 4.1. Some network-based services operate at the fundamental network layer; these are the most problematic to the future of network transparency. Other services are at the application level; they are "in the application", but "attached to" the network, as described in detail in Section 4.2 on application design.

6.2.3 Universal connectivity

NIA: A new Internet should permit two parties that want to communicate to do so.

But some parties may desire to block communication, and the network should permit and support this.

NIA: The architecture should include a component that can filter traffic based on criteria specified by end nodes and administrators, including possibly DDoS suppression.

6.2.4 Immediate delivery

NIA: This principle would be unchanged.

The NewArch project discussed whether NIA should support delay-tolerant networking explicitly, but concluded that such networks demand a very different set of architectural principles. They are two different systems; one is not an extension of the other. So this study chose to concentrate on a network that is a descendent of the Internet in this respect.

6.2.5 End-to-End principles

The erosion of the end-to-end principles raises the question of whether a fresh design would abandon them in favor of some other approach. We concluded that the end-to-end principles should *not* be abandoned or replaced; they still provide a clear, defensible, utilitarian division of responsibility in the architecture. However they must be stated carefully, and there are exceptions that must be recognized.

Section 6.2.2 stated a more complex requirement for "transparency modulated by trust" than the original "always transparent" model of the original Internet. It is necessary to reconcile this more complex requirement with the end-to-end principles.

> ***NIA: To the extent possible, mechanisms that provide constraints to protect untrusting applications should be designed so that an application (or a transport layer) that assumes a transparent network can continue to work unmodified, unless the constraints are application-specific and specified explicitly as part of the application design.***

A constraint should be designed as a restriction on the set of otherwise acceptable actions, but not as the creation of new behavior.

As a simple example, one obvious form of constraint is total blocking of connectivity. While this stops an application from working by intention, it does not introduce a new failure mode that an application needs to address—loss of connectivity is a reality of any network. A more interesting example of a constraint is a *protocol normalizer* [HKP2001] that detects packets that are in an "unusual" form and transforms them in a way that removes the abnormality but still leaves them conforming to the specification. This form of constraint strips out any attempt to use a malformed packet to trigger some low-level bug, while not requiring the end-node to be aware of the transformation, since the normalizer carefully preserves all aspects of the end-visible specification. Other examples might include a requirement that the sender include an IP option or limit a sending rate, neither of which would require redesign or recoding of the receiving end-

6.2.6 Loose semantics

> ***NIA: The principle of loose semantics 6.1.6(a)) is retained.***

If a new architecture moved away from syntactic transparency and permitted reformatting or recoding of data in transit, there would be a compensatory need to more carefully define the end-to-end semantics. However, since the project decided against allowing non-packet multiplexing (see 6.2.1), loose semantics can be retained.

Another issue is network performance and what is exposed to the hosts. The following alternative to 6.1.6(b) is highly desirable, although it may conflict with the desires of ISPs that do not want to publish their internal performance:

> ***NIA: the network should contain architected tools to measure its performance and make this information available to end nodes.***

6.2.7 Subnet heterogeneity

Since the Internet architecture has been successful, we are in a position to ask that new subnet technologies be tailored to the Internet's needs. This would mean allowing a future Internet to operate over highly diverse technologies while taking advantage of enhanced link-layer functionality where it exists. This would require a careful definition of desirable link layer enhancements; for example, these might include: congestion indication, QoS, self-characterization of performance parameters, or context setup.

6.2.8 Common bearer service

> ***NIA: These principles would be unchanged.***

6.2.9 Connectionless network

A common assumption is that the only two choices are connectionless and connection-oriented, but in fact there is a range of possibilities for forwarding context, ranging from a single global routing computation to per-flow setup of forwarding context.

NIA: The architecture should support a spectrum of contexts for forwarding packets. The network may detect when the establishment of additional context would improve the efficiency of the forwarding process and adapt accordingly.

NIA: Explicit/path/source routing should be a sanctioned forwarding mechanism at the region level.

6.2.10 Global addressing

NIA: Network addresses imply nothing about the identity of the entity at the end point. Entities can change their network attachment addresses without disrupting on-going communication.

NIA: There are high-level name spaces (e.g., the DNS) to allow nodes to name each other.

There have been many proposals to separate the two functions of an IP address by distinguishing the locators at the network layer from identifiers at the transport and higher layers. Such a distinction would enable communications to continue during locator changes resulting from mobility, renumbering, or multihoming. The project discussed three alternatives for addresses (locators) in NIA:

Option 1: NewArch continues to depend upon a single, global address space for locators.

Option 2: The architecture does not depend on globally meaningful network addresses (locators). Within a region of coherent addressing, addresses define network edge attachment points. As data is forwarded through the network, the destination address may be altered or replaced, based on local context or higher-level information in the header.

Option 3: Topologically-significant objects called forwarding directives (FDs) are used at the network layer for packet forwarding and delivery. FDs have uniform syntax but may be subject to different interpretations. Some FDs are meaningful in a global address space; others are meaningful in other contexts. A distinct set of globally unique identifiers name nodes. Addresses imply nothing about the entity at the end point. End systems can change their locators without disrupting ongoing communication.

The NewArch project developed and prototyped the FARA architectural model, based upon Option 3; see Section 3.1 of this report.

6.2.11 Protocol layering

The project began the exploration of Role-Based Architecture (Section 3.2 of this report), which seeks to dethrone layering as a fundamental principle. Much more research is needed to determine whether this is a fruitful direction, however.

6.2.12 Distributed control

NIA: This principle would be unchanged.

This principle is (still) required for robustness.

6.2.13 Global routing computation

If NewArch relaxed the global address space requirement (Section 6.2.10), the requirement for globally consistent routing would be relaxed:

NIA: A routing computation is performed within each region where consistent addresses are used.

6.2.14 Regions

The region notion embodied in a routing domain should be generalized.

NIA: The architecture should include a general notion of regions, to express differing interconnection policies, trust relationships, multiplexing mechanisms, etc. Regions may also support distinct addressing regimes, performing any necessary address mapping as data crosses region boundaries.

NAT boxes and firewalls depend upon their topological location in the network (and the non-existence of bypass paths), but we want to consider such topological control to be an administrative matter, not part of the architecture.

6.2.15 Mobility

NIA: This principle would be unchanged.

The Internet architecture optimizes for the non-mobile case, using a special-case mechanism for mobility. We decided to leave it this way.

6.2.16 Security principles

Internet security is a critical issue in defining tomorrow's network. We reached certain conclusions in this space, including the requirement for *trust-mediated transparency,* described in Section 4.1.11. However, a more general resolution of the security problem is hampered by a more fundamental limitation—a lack of agreement on what is meant by "security", and what problem is to be solved. Security is not a simple goal where "more is better", it is a complex tussle of requirements, defined by deep conflicts of values.

We believe that some security principles are clear at this point. First, the network must be able to defend and recover from attacks on the network itself. In this respect, routers are similar to end-nodes in their security needs—they require a secure communication channel and confidence that they are talking to the intended partners.

NIA: Routers must protect their control communication from attack.

One class of denial of service attack attempts to flood the network with enough traffic that legitimate traffic is blocked. The worst outcome of such an attack is that even the control traffic that might serve to respond to the attack is blocked. So protecting the ability to send control traffic is critical as a building block for further steps in mitigation and recovery.

NIA: Routers, as well as subnet technologies that have internal control state, should provide a priority scheduling or other mechanism to assure that network control traffic can be forwarded during an attack.

More complex and contentious are the issues surrounding network mechanisms to protect the end-nodes from attack. One extreme is that each host is responsible for its own defense. We believe that an approach of "defense in depth" is appropriate, both end-nodes and checkpoints in the net should combine to defend end-nodes from attack. See Section 6.2.2 above.

NIA: The architecture should include a component that enforces/verifies behavior at region boundaries, which include the host-network interface, and the region-region interface.

6.2.17 Resource allocation principles

Congestion Control: The project explored a radical alternative to the existing approach to congestion control: XCP (see Section 3.4). XCP provides more rapid convergence to intended transmission rates, operates effectively over a wider range of latencies and speeds, and provides more direct control of allocation shares. XCP is one of several recent proposals that use resource allocation state in every packet to avoid per-flow state in routers.

Quality of Service: During the 1990's, the research community invested a great deal of effort in mechanism to provide different quality of service (QoS) in the Internet. Some, though not all, of the creators of those mechanisms were convinced that QoS ought to be a part of the required architecture of the Internet. However, the failure of the commercial world to take up these schemes suggests that there is no agreement on this point.

Our consideration of this situation led to the hypothesis that the problem is not that the mechanisms are unsuitable, but that they were not designed taking into account the *economic reality* of the commercial Internet. This lead to the articulation of a meta-principle as follows:

> ***NIA: To insure the economic viability of a future network, architectural principles and mechanism proposals should take into account economic and business considerations as well as technical requirements.***

It is the opinion of many of us that QoS will eventually be a part of the core architecture of a future Internet, but (as in congestion control) the mechanism can be added in a way that does not interact strongly with other architecture requirements.

Direction of value flow: one example of an economically motivated mechanism is the proposal that packets should carry an indication of whether the value flow (e.g. the direction of the payment) is toward or away from the source. In the current Internet, an examination of the packet flow cannot tell whether the sender or the receiver should be responsible for payment, especially if there is an incremental payment for services such as enhanced QoS. Even within our design group, there was resistance to this sort of reasoning, or to including "business structure building blocks" within the architecture. We strongly recommend that these sorts of considerations be the subject of continuing research and discussion, so that a degree of comfort and consensus can emerge over time.

6.2.18 Minimal dependency

> ***NIA: The dependency graph among the components of the architecture must loop-free so the network can restart after a massive failure.***

A useful consequence of minimal dependency is that it avoids logical loops in restarting nodes.

6.3 References

[BCS94] R. Braden, D. Clark, S. Shenker, *Integrated Services in the Internet Architecture*, Internet RFC-1633, June 1994.

[Blumen01] M. Blumenthal and D. Clark, *Rethinking the Design of the Internet: the End-to-End Argument vs. the Brave New World*, Proc. ACM Trans Internet Technology, *1,* 1, August 2001.

[CSTB1994] Computer Science and Telecommunications Board, *Realizing the Information Future: The Internet and Beyond,* National Academy Press, 1994.

[CT90] D. Clark and D. Tennenhouse, *Architectural Considerations for a New Generation of Protocols,* Proc ACM SIGCOMM, Sept 1990.

[Fall2003] K. Fall, *A Delay-Tolerant Network Architecture for Challenged Internets*, Proc. ACM SIGCOMM 2003, Karlsruhe, Germany, 2003.

[HKP2001] M. Handley, C. Kreibich and V. Paxson, *Network Intrusion Detection, Evasion, Traffic Normalization and End-to-End Protocol Semantics*, Proc. USENIX Security Symposium, 2001.

[Jacobson88] V. Jacobson, *Congestion Avoidance and Control*, Proc. ACM SIGCOMM '88, Stanford, CA, August, 1988.

[SRC81] J. Saltzer, D. Reed, and D. Clark, *End-To-End Arguments in System Design,* 2nd International Conf on Dist Systems, Paris France, April 1981. (Also in ACM Transactions in Computer Systems, 2(4): 1984).