Addressing Reality: An Architectural Response to Real-World Demands on the Evolving Internet

David D. Clark
MIT LCS
ddc@lcs.mit.edu

Karen Sollins
MIT LCS
sollins@lcs.mit.edu

John Wroclawski
MIT LCS
jtw@lcs.mit.edu

Ted Faber
USC ISI
faber@isi.edu

ABSTRACT

A system as complex as the Internet can only be designed effectively if it is based on a core set of design principles, or tenets, that identify points in the architecture where there must be common understanding and agreement. The tenets of the original Internet architecture [6] arose as a response to the technical, governmental, and societal environment of internetworking's earliest days, but have remained central to the Internet as it has evolved. In light of the increasing integration of the Internet into the social, economic, and political aspects of our lives, it is worth revisiting the underlying tenets of what is becoming a central element of the world's infrastructure.

This paper examines three key tenets that we believe should guide the evolution of the Internet in its next generation and beyond. They are: design for change, controlled transparency, and the centrality of the tussle space. [8] Our purpose is not to present these ideas as new, but rather to propose that they should be elevated to central tenets of the evolving architecture of the Internet, and explore the ramifications of doing so. The paper first examines the tenets somewhat abstractly, and then in more detail by studying their relation to several design choices needed for a complete architecture. We conclude with a discussion of the relationship between the network architecture and the applications it serves.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design

Keywords

Architectural principles, architecture design, transparency, design for change, tussle, security, application support.

1. INTRODUCTION

This paper is concerned with network architecture, the fundamental tenets that underpin the actual design of the network. Specifically, we are interested in the architecture of the Internet as it evolves into a central component of the world's technical infrastructure. That current Internet architecture is based on fundamental technical and social requirements [6] that we believe are basically sound, but that should be assessed and modified in the light of current experience. The Internet has been undergoing significant technical and philosophical changes for years, and introspection on those changes is overdue.

We believe that the basic challenges facing the network architecture arise from the network's growing place in society. This contrasts with the situation in the past, where the most fundamental questions facing the architecture have been primarily technological and performance-oriented. While adoption of new tenets must clearly continue to be technically sound, and not unnecessarily impede performance improvements, we argue that the single most important change facing the Internet architect is the qualitative change in perception of the Internet's role and importance, and the resulting new requirements and limitations placed on its design.

This change takes many forms. In the Internet today, basic assumptions about trust, autonomy, and shared objectives are being challenged. As more people and institutions use the Internet to interact with one another, they inevitably encounter unknown or even untrustworthy entities. In the face of potential abuse or other malice, it seems clear that future Internet designs need to address *trustworthiness* – a concept that is much broader than traditional security.

As corporations, national governments, and social groups with varying perspective have come to both provide the infrastructure for and make demands upon the Internet, questions of control of resources come into sharper focus. How much control a provider of bandwidth has over control of content, or to what extent corporate or public policy should be reflected in routing choices, were not part of the original Internet design. It seems clear that future designs ignore such questions at their peril.

Beyond questions of control of the network components, questions of how the network as a whole are used no longer play out against a backdrop of shared purpose. Early Internet usage was largely characterized by common goals of free communication for academic pursuits. Now, a wide variety of players use the network, with often contradictory goals. For example, there are content distributors who do not assert property rights, like free software providers, and content distributors who assert restrictive property rights. Their fundamental goals are different.

To address these issues, we propose elevating three concepts to the level of central guiding tenets. These are *the primacy of design for change, controlled transparency,* and *isolation of conflicts of interest.*

ACM SIGCOMM 2003 Workshops, August 25&27, 2003, Karlsruhe, Germany.

Design for change encourages a network designed for flexibility, elegance and simplicity whenever possible, because such systems endure long after over-optimized and over-specified systems have vanished. Controlled transparency allows network entities to regulate what interaction they tolerate based on their trust of the questioner. Lastly, a network that supports conflicts of interest is designed to accommodate tussles [8] within the system elements.

We proceed by discussing these three core tenets in more detail in Section 2. In Section 3, we will consider a set of architectural choices, examining them in the light of the preceding discussion. Section 4 takes the applications' perspective, and in Section 5 we revisit the topic of state in the network, in light of our discussions.

2. Tenets

In 1984, Saltzer, Reed and Clark [23] first enunciated the end-to-end arguments. These are a series of examples of choices of placement of functionality in distributed situations, either what one might call distributed systems or networks. The following discussions, each in its own way, represent a continuation of reflections on the end-to-end arguments and where to place functionality.

2.1 Design for change

The Internet architects understood from the beginning that the network had to be designed with generality in mind. The computer industry evolves rapidly, and the Internet was intended to hook computers together, so the Internet would have to evolve as well. The generality designed into the Internet has allowed it to support applications that were never contemplated when the design was first conceived, and to embrace new technologies and modalities with drastic differences in performance and behavioral characteristics including such "disruptive" advances as mobility. The Internet design has survived technology speedups of many orders of magnitude, and similar factors in size, as well as more functional changes.

This success implies that as we evolve the Internet, we must continue to *design for change*. Carpenter [4] was the first to call out "constant change" as the primary characteristic for which a network architecture must be designed. Many, if not all, designers recognize that we must resist point solutions, momentary optimizations that return to haunt us, and other forms of short-sighted thinking, as detailed design decisions are made.

What is less broadly acknowledged is that as any technical infrastructure becomes established and embedded, and the natural forces opposing change become dominant, it becomes necessary to take *explicit, architected action* to preserve the ability to change, evolve and advance the technology. A corollary of this is that this action will often involve *sacrifices in other dimensions*, such as performance and efficiency. Unfortunately, these sacrifices often result in short-term costs being incurred to preserve a longer-term, less concrete benefit. The challenge facing the network architect is to arrive at an overall architecture that both encourages protocol designers to preserve generality and evolvability, and minimizes the costs of doing so.[1]

This is at first glance quite a high-level statement, but its implications can be very concrete. For example, one of the most successful aspects of the Internet, in terms of designing for change, is the use of packets. Packets optimize for flexibility and support for diverse service requirements, not for any single application. This approach has been successful in fostering innovation, as the flow of new applications can testify. Additionally, packets have proved their utility as a unified representation that "works over anything". Almost any communications technology can be made to carry packets in some way or other.

Critics will, often correctly, object that packets are not an optimal way to use one or another underlying medium. However, this begs the question of whether "optimization" should be the dominant goal, and how the tradeoffs among optimization, generality, and evolvability should be captured and evaluated. As the Internet has grown up and proven itself, we have seen that different media platforms have evolved to carry packets more efficiently, e.g. time-division wireless such as 802.11. This suggests that within limits (see the discussion of aggregation in Section 3.1), the technologies can morph to carry packets efficiently; the architecture need not morph to match the technology of the day.

2.2 Controlled transparency and trust

The Internet is often described as "transparent", by which is meant that what goes in comes out. The network (allegedly) does not perform internal transformations, processing or other actions on the data.[2] This model has served the Internet well. It is easy to understand, it does not "get in the way" of applications trying to do something new, and for these reasons it has fostered the development of new applications. Transparency was another aspect of design for change, and it was a very successful approach.

In the early days of the Internet, transparency was seen as an unalloyed good. However, the real world tells us that today users want to be protected from some classes of traffic. Traffic directed towards a host is often hostile, and users want protection in the network, not just within the end-nodes. Devices such as firewalls are popular, even though they "break" the transparency of the network. We thus observe that the role of transparency in a future network is more complex than, say, the role of packets. Transparency is good when considering design for change, but bad when we consider security.

Firewalls are only an approximation to what the users want. They block classes of traffic based on where in the network the traffic is coming from. This type of control is crude and

[1] It is worth noting that generality preserved through the "millions of options" approach to extensibility often does *not* meet this test, because the implementation and validation overhead and the interoperability problems of the approach may outweigh any benefit to be gained.

[2] This view of transparency is consistent with the specification of the IP protocol [20], in which the concern is with delivering the bits as transmitted to the receiver. In contrast, some authors [4] consider transparency only as address transparency.

imperfect. It can only prevent certain classes of attach from "outside"; it does not prevent attacks such as DDoS that masquerade as messages to legitimate service ports. Conversely, firewalls do nothing to control attacks from "inside"; caused for example by disgruntled employees. At the same time, some users who trust each other may be on far sides of the network. This leads to increasingly complex structure and behavior, as VPNs and other methods to extend trust boundaries evolve.

We claim that instead shared trust should *implicitly modulate* the degree of network transparency at a basic, architectural level. Ideally, when a set of users that trust each other communicate, the network will appear as transparent as ever. When users that do not trust each other communicate, the network will constrain what is sent to only that which each end is prepared to tolerate. This suggests that a design tenet for the Internet of tomorrow should be *controlled transparency,* often trust-regulated, which is more complex than the simple transparency of today, but addresses both the goals of design for change and security.

One venerable principle of Internet design that has much in common with transparency is the *end-to-end arguments*, which call for function to be implemented at the edge of the network rather than in the core of the network when this is practical. A network that is transparent, as we define it above, does not involve itself with the data being transferred, so it conforms to the end-to-end design. Devices like firewalls, which move function into the net, begin to violate the end-to-end design. So one challenge, discussed below, is how to marry the benefits of the end-to-end approach with controlled transparency.

2.3 Conflicts of interest

The observation that many users do not trust each other is closely related to another observation—many users of the Internet have interests that are adverse to each other. Users want private communication; governments want to wiretap their conversations. Users want to exchange music; rights holders want to prevent this. Users want to send what they please; Internet service providers (some of them) want to regulate what they do, either as a matter of policy (employees should not look at pornographic material at work) or as a means to segregate high-value users (users who want a server at their residence have to purchase a higher-price broadband service.) These sorts of contention are intrinsic to the Internet; they cannot be avoided by making a design decision. The paper by Clark et al. [8] called this ongoing process of contention *tussle*, and proposed that network designers should make their goal designing *tussle spaces*, instead of thinking that they going to resolve tussles. Once one takes this viewpoint, one may be in a position to bias the tussle by the shape of the tussle space, which is a more subtle but more realistic form of control over the future. The paper also identified some design tenets, in particular the isolation of tussle-spaces so that unrelated tussles do not spill over into one another.

3. Getting more specific

In this section we discuss a selected set of design features that we believe should be at the core of a new generation Internet. Each reflects one or more of the above general design tenets.

3.1 Making the best of packets

Questions about the net value of packets have been with us from the beginning of networking. We conclude that use of packets, or more generally the provision for universal fine-grained statistical multiplexing as a core architectural tenet, is a good idea that has passed the test of time. Even if one could design a network from scratch today, the fine-grained multiplexing implied by packets would be the way to go.

Packet-switched networks are motivated by the fact that the ability to multiplex traffic from many of sources in a fine-grained way is critical when traffic is bursty and intermittent. [9], [28] As they have since remote login was first deployed, applications continue to display this same bursty behavior. While there are persistent predictions that the next application will not fit this model and will require building a network with constant-capacity circuits, this outcome has not materialized. Statistical multiplexing as provided by packets is the way we can give a user access to peak capacity on demand without dedicating this capacity to that user full time.

There is a spectrum of service commitments that a network might make to an application, from totally guaranteed bandwidth to a service based on mixing lots of different sources and provisioning to make sure that there is enough capacity most of the time—this is the "best-effort" service of the Internet. Packets (with some mechanism added) can do all of these. We may debate whether we need to augment the best-effort model, e.g with some sort of QoS. This does not invalidate the basic need to deal with bursty traffic or the decision to use packets as the unit of multiplexing.

We observe that the use of packets is an example of both design for change and tussle-space design. The provision of fine-grained multiplexing allows for effective, unpredictable uses of network resources. By allowing for a spectrum of service commitments, the packet approach defines access to those services as a controlled tussle-space.

However, this does not mean that the (architectural) status quo is correct. One aspect of the packet architecture that is missing from the Internet today is any recognition of *aggregates* of packets. By multiplexing lots of sources, service providers make much better use of trunks; this is what makes the economics of the Internet work. Going back to our first tenets, it is packets that support the goal of design for change, but it is the properties of aggregates of packets that support economic viability. Aggregates of packet flows have significantly different statistical properties than the packet flows that compose them—in general they have smaller short-term fluctuations in rate, so they can make significantly more efficient use of fixed capacity trunks. Different mechanisms for the same problem may be preferable at the edge and in the center. [2] The simplest form of an aggregate is implicit—just all the traffic going over a trunk. But aggregates are also a natural unit of traffic engineering, and may be an equally natural unit for on-demand bandwidth acquisition or QoS control. This has obvious implications for the architecture. Consider the network operator wishing to identify all the traffic going between city-pairs and manage the routing of these aggregates to balance the load on trunks. This objective requires that the city-pair aggregate be explicitly identified in the routing. The current Internet architecture has no way to name or manage aggregates, so lower level mechanisms such as MPLS are used to name these entities. This leads to duplicated

function and potential conflicts and instabilities between competing algorithms operating at different protocol layers.

Instead, the architecture should include some means to name and reason about aggregates as fundamental, first class objects, so that important objectives such as traffic engineering can be expressed within the architecture, rather than outside it. This requirement does not imply that the architecture should mandate the basis for forming the aggregate—there can be many reasons for doing this. This requirement only means that the architecture should understand the concept of an aggregate, and should be able to perform actions on it such as routing.[3]

Because the center of the network deals with aggregates, it is apparent that two sorts of network technology will evolve. Near the edge of the network, where total speeds are lower and the degree of aggregation is low, technologies will be used that can multiplex packets and that can deal with statistical fluctuations in traffic. Near the center of the network, where the speeds are high and the degree of aggregation is high, technologies are emerging that deal with aggregates. [22], [26] This pattern is clearly emerging today. However, much of the power and flexibility of the approach is lost because the boundary between the two zones is unnecessarily rigid in the currently evolving approach.

It is important to notice that the inclusion of aggregates in the architecture changes the nature of the tussle-space over service access, providing the "clients" with a new set of handles on that tussle-space.

3.2 Transparency, security and end-to-end design

In the 1970s, the security community had a clear idea of the security problem to be solved and how to solve it. [12] Their formulation of the problem had three parts: unauthorized disclosure, loss of data integrity, and denial of service. Security was framed in the context of an isolated machine, and the proposed approach was a "security kernel" that mediated all reads and writes to data. Most of the attention was directed to disclosure and integrity—denial of service was considered too hard to handle in a systematic way. Attacks on systems were viewed as one means to achieve disclosure or loss of integrity, and were treated in that context.

Managing disclosure and integrity in a networked world is best organized by separating host security from the problem of protecting data "in the network". The network problem is actually fairly easy to solve, since data can be encrypted end to end, if it need not be modified in transit[4]. So an "end-to-end" approach to disclosure and integrity control protects data in the network. At the same time, protection from disclosure and data corruption within the host is (sometimes) easier, because we no longer run multi-user systems with different users running unrelated programs on the same machine. We usually can afford to put mutually suspicious users on separate machines, and limit multi-user interaction on a single machine to *servers*, where the high-level security problems manifests at the application level. Between end-nodes, there is a varying *spectrum* of problems to solve. When two end-points trust each other, it is appropriate and efficient to adapt an end-to-end approach to integrity and disclosure. But among non-trusting machines, the focus shifts to a different set of problems—preventing one machine from attacking another, and providing assurance (e.g. via a third party) that both end of a transaction can be held accountable. The security tools of choice depend on where in the spectrum of trust-regulated transparency the end-points lie.

This approach to disclosure and integrity issues shifts attention to denial of service and attacks on the end-nodes. Attacks on end nodes are facilitated by pure transparency. Pure transparency can be made secure in principle—it requires that all end-points be perfect protectors of their own facilities, and dedicate their resources to protecting themselves. However, this means that hostile traffic can consume network resources at will, since it will only be detected at the end-point. In practice, the world is not going in this direction. Observation of the real world suggests that people want a security architecture that involves points of protection "within" the network. So network designers are now putting technology into the network, not for the classic disclosure and integrity control, but to fight system attacks and denial of service attacks.

While perfect host security might reduce the call for controls "in the net" to counter attacks on end points, protection within the network is here to stay once we consider problems of theft of network service, denial of service by flooding, and other attacks on the network itself. But it is important to remember that these two problems (host vs. network security) have different implications for the protection mechanisms.

As we add new security services to the network, it is important to think carefully about what they do to transparency. The desirable ultimate outcome would be a universal *model* for this function, independent of specific implementation. A simple starting point might be that security checkpoints should either appear totally transparent (as if they are not there), or block traffic totally, so that they mimic a network failure (which applications deal with today anyway). A simple augment would be that when security considerations permit, some sort of signal to the end-point should be defined so that the users know what failure occurred.

As attention has shifted to denial of service, that problem is being broken into parts that can be separately understood and managed. So a framework for addressing denial of service is emerging.

[3] Some calls for abandoning packets (e.g. in the center of the net) [1] are a result of not having aggregates as part of the architecture. For example, some high-speed optical technologies cannot switch fast enough to forward individual packets, implying that some other unit of multiplexing is needed in the middle of the network. If the aggregate were properly designed, these technologies could be designed to deal with them, not with packets. It is then a matter of optimization whether the packet headers should be removed for efficiency and reconstituted or just sent for greater simplicity, assuming they are recoverable.

[4] If the design of the application *does* require that the data be processed while it is in transit across the network, this implies that the end-node have decided to trust those intermediate nodes, which becomes part of the security equation.

1. Routers and similar devices inside the network can use encryption and other means to protect themselves from attack, just as hosts use end-to-end techniques. Note this implies a degree of trust among the routers so connected. (The problem with this approach is the possible increase in the need for manual configuration of routers, which adds to operational problems.) [15]
2. Routers and other devices that allocate capacity can use priority or multiple queues to ensure that key network messages can be forwarded during an attack, and abusive traffic can consume only a share of network capacity, assuming the masquerading problem is solved.
3. Devices that constrain or verify behavior will be positioned at region boundaries, at points where the trust assumptions change and where usage limits are to be imposed.

Attacks on end nodes are thus managed using a mix of traditional end-node security and filtering of unwelcome traffic (traffic from un-trustworthy nodes) upstream from the host.

One result of this framework is that different parts of the network can see into the protected areas with varying clarity. To completely untrusted sources, the area is opaque; as sources demonstrate more trustworthiness, more services and endpoints become visible; completely trusted external sources or internal sources can see all services. This controlled transparency is an emerging model for protecting networks and services.

This general approach is coming into focus, but has not yet evolved into an appropriate network security architecture. There are no developed proposals for the expected division of responsibility between the end-node and the network, and no frameworks that let a system operator verify that the sum of the mechanisms in place lead to a consistent level of security. So there is much work to be done in this space. However, there are some conclusions that can be drawn.

First, protection points will be stateful. We can perhaps learn from today's experience with firewalls and NAT how to think about this state. Second, different parties (e.g. the operator of the end-node and operator of the network) may *disagree* over the desired security policy in these devices. This is a tussle space. A second tussle space is the conflict between the end-user desire for confidentiality and virtually all governments' desire for lawful intercept. An interesting example of the tenet of separating tussle spaces was the US government's willingness, at one point in the debate over use of encryption technology, to let it be used for integrity but not for confidentiality. Mechanisms that tangled the two prevented the deployment of improved integrity (which may have been a tactical move by one side in the tussle.)

3.3 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. It provides a weak foundation for theorists to "reason about" the Internet, but as a practical matter it has benefited most applications by forcing them to be much more robust to varying performance and technical changes than they would otherwise have been.

As the Internet matures, it is useful to ask whether new forces challenge this perspective, and if so, whether the benefits of the perspective continue to outweigh its disadvantages. We suggest that the primacy of weak semantics is indeed 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. In essence, this is a network externality effect: within the space of a "weak" semantics that explicitly or implicitly tolerates alternatives, behaviors that do not provide significant benefit, or 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. The key point is the positive feedback loop: as these behaviors become less and less common, developers start to optimize for the case in which they do not exist, causing the system to tolerate them less. Over time, the de facto system specification becomes more precise, and more minimal.

Many examples of this phenomenon are demonstrated in today's Internet. Some are fairly obvious: over time the limited use of IP options has led to the development of router fast-paths that do not support them, rendering options less and less useful. In effect, IP options are being removed from the specification.

It is interesting to note that the same force can remove aspects of a specification that at first appear much more fundamental. Consider IP source addresses. Although the IP specification would appear to demand that source addresses are to be carried end-to-end, most common uses of IP do not actually depend on this. As a result, the requirement has turned to an assumption, which now is becoming weaker and weaker. With the advent of NAT boxes, packet rewriting gateways, and the like, it is now decidedly unwise to assume that the source address in a packet has any relation to the end system sending the data.

We offer two observations. The first is that this phenomenon offers both cost and benefit. The benefit is that as the de-facto system semantics becomes more precise and minimal, it becomes more and more possible to optimize implementations, and more and more possible to depend on precise details of the architecture's behavior. This leads over time to higher performance and greater efficiency for existing applications.

The obvious cost, however, is ever-increasing loss of the flexibility and evolvability that is a hallmark of systems with weak semantics. Since, as we have argued above, this progression is both natural and powerful, the system architect that desires to limit its more negative effects should expect that explicit action is needed to do so.

Unfortunately, the range of options available to slow this progression is not entirely clear, and deserves further research. Strategies employed in the past range from formal coverage and regression tests of implementations (which can help

prevent atrophy of little-used capabilities, but effectively remove much of the underspecification) to building in "semantic escape paths" such as IP options, and then *using these paths for at least one critical function* to keep them viable. Less technical forces also play a role. For example, the maintenance of a healthy application development and research community serves to continually throw diversity into the mix, balancing the drive toward homogeneity.

A second force increasingly acting against the existence of weak semantics is 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". This is a direct challenge to the weak semantics paradigm and its concomittant benefits.

We consider ways to balance concerns about security with the perceived benefits of weak semantics described above.[5] A first question is why "weak semantics" is seen as problematic to secure systems. One plausible conclusion is that precise and minimal system semantics, by offering more direction to the developer, 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. This is particularly relevant because it is widely recognized that many networked system security weaknesses are not due to mis-specification, but rather to mis-implementation - problems such as buffer overflows, unhandled bit combinations, and the like. [5] [19]

This concern can be alleviated in two ways. The first is to use implementation techniques, such as higher level programming methods or formal verification methods that are less susceptable to attack through mis-implementation. Of these, high-level approaches such as formal protocol verification are somewhat problematic in the presence of weak semantics, but a substantial percentage of lower-level security holes could be avoided without threat to the benefits of weak semantics through stronger software engineering methodology.

A second approach is to protect against mis-implemented or buggy code through the use of a separate, presumably simpler, more understandable, or more reliable filtering device. The most common of these is the firewall.[6] Unfortunately, the binary, all or nothing nature of firewall protection contributes directly to the problems we hope to avoid here - current best practice is to disable all paths through the firewall that are not specifically required.

A better approach may be through the use of *protocol normalizers*. [14] Here, some flexibility is preserved. The device, rather than simply restricting actions it does not expect, attempts to *normalize* them by ensuring that use of the protocol falls within understood boundaries, and actively reshaping it to do so if not. This more active and permissive approach contrasts with the firewall's passive but restrictive model. In theory, a correctly matched normalizer - end-system pair will preserve significant flexibility while remaining nearly as protected against mis-implementation security holes as is a firewalled system. In practice, it seems likely that the increased complexity of the normalizer approach, with corresponding increased risks, must be traded off against the additional flexibility. Our conclusion is that this strategy merits further research.

To consider this situation further, it is useful to recognize that there are two separable benefits to weak semantics. One is to allow flexibility of implementation. The other is to allow for growth and evolution in functionality of the protocol.

The protocol normalizer example above is most suited to preserving benefits of the first sort. The trick lies in identifying classes of underspecification that are potentially exploitable in specific environments, and then mitigating their effects through normalization.

Simultaneously maintaining functional flexibility and security in the presence of "evolution" underspecifications is more difficult. The reason for this is that allowing for evolution often means specifying that unknown or unexpected protocol actions must be ignored - "be conservative in what you send; be liberal in what you receive." This idea is in *direct* conflict with the security-motivated principle that unanticipated protocol events or data objects should be rejected rather than permitted. Reconciling these two perspectives successfully requires moving away from either simple endpoint - instead making a sophisticated, probably runtime decision about which events to permit and which events to reject, based on more complex analysis of the surrounding environment and constraints. This too is a subject for future research.

Despite the challenges posed to the weak semantics paradigm by the forces described above, the implementation and systemic benefits of the paradigm remain substantial. However, that is the most we can say. System architects today lack the tools to rigorously reason about *how* substantial these benefits are, or what the cost tradeoffs might be. At this point, the ability to select a minimal set of semantics with just the right degree of weakness or strength of the definition remains an art. Success in this regard is a mark of the best systems designers, and the process by which the skill is taught and learned is not well understood. We suggest that this question is ripe for deeper study, and successful research results in this area underpin the development of a more scientific approach to our goal of systemic design for change.

3.4 Naming and addressing

Aside from packets, nothing may be as fundamental to the Internet as its idea of host addresses, naming, and routing. Since the original design of the Internet, there has been much informal debate about how identity and addressing should be handled. In the Internet, the IP address serves both to locate the end-point within the network, and to provide a (weak) identity check. TCP includes the IP address in the pseudo-header to preserve the integrity of the end-point association. DNS gives out an IP address when presented with a name, with

---

[5] The authors thank Mark Handley and Steve Bellovin for a useful email exchange that led directly to the points made in this section

[6] "The primary purpose of firewalls has always been to shield buggy code from bad guys." Steve Bellovin, private communication.

the implication that the service at that location is actually the one desired. There is no other means to check that the service at the end-point is the intended one unless there is some shared application-level validation such as a login or exchange of secrets.

3.4.1 Separation of identification and location

The proposal has been made many times that identification and location should be separated. We believe that this is a desirable goal, and should be a part of a future Internet architecture. But this goal raises some interesting challenges and a detailed design is necessary to expose all of them.

One consequence of mixing location (address) and identity is that since users do not want to lose their identity, they desire to keep the same address, even when they move. The telephone system mixes location and identity in a serious manner, since phone numbers (rather than something like DNS names) are printed on stationary and advertisements, and are remembered by parties that want to call each other. So government regulation has come to bear on the phone system to make numbers portable; in the United States, 800 numbers are portable today, local number portability and cellular number portability are coming. What this implies in practice is that in an example such as this phone numbers less and less express location, and a lookup in a flat database is required inside the phone system to translate the phone number dialed to the internal location information.

In contrast, the Internet still maintains some efficiency in the use of IP addresses to express location. The Internet has higher level naming mechanisms such as the DNS, and in general, users are expected to change their addresses when they move, so addresses can still actually be used as addresses, rather than a lame form of identity. This approach is the correct way to go. For scaling reasons, routers cannot do a lookup for each packet in a flat database of entries of ever increasing size, and if IP addresses cease to express location, we will have to invent a lower level identifier that does. A future architecture should have an efficient means of expressing location, where efficient means that it meets the needs of the routers.

Some proposals for the use of IPv6 have missed this point, and describe a scheme where devices are manufactured with their address built-in. This view makes an address into an identity rather than a location, and should be resisted.

In arguing for separation of location and identity, it is important to note that location changes for all sorts of reasons. A common example of changing location is host mobility. This problem has received a great deal of attention, but there are many other reasons that the location of an end-node may change. First, the whole network to which the host is attached may move. Networks are appearing in cars, trains, planes, and people are starting to carry networks on their person. Networks like these may change their attachment point to the Internet frequently. Second, a host or network may be multi-homed, and may change the path it uses from among the ones available. Third, the operator of a host or a network may change its service provider. If we accept the goal of efficient representation of location, all of these actions will imply that the address changes. So change of location implies that address dynamics will be much more pervasive, and all hosts, not just mobile hosts, will have to deal with address management issues. Also note that there are tussle spaces within this design. Consumers tussle with providers over customer lock-in. Providers try to capture customers, and customers try to preserve the easy ability to change providers. If identity gets linked to address, and the address is linked to the provider, then one cannot change providers without losing ones identity. This problem manifests most often at a higher level, when people who change ISPs discover that they have to get a new email identity.

3.4.2 Identities and translation

If we no longer use IP addresses as the identities for end-points, we are left with three key issues to address:

- **Scope**: do all end-points need identifiers?
- **Identifier assignment:** How are the identifiers chosen? Must they be unique over the whole Internet? Can they be reused over time?
- **Identifier translation**: When and how are the identifiers translated?

As the discussion above suggests, there are two key functions we expect of identifiers: *verification* and *resolution*. These are very different from each other. As Moskowitz [18] has proposed, one can create public-key pairs for verification or authentication that have no relationship to location or accessing the end-points. Other sorts of names or identifiers, such as DNS names or URNs (Uniform Resource Names) [10], [25] might be resolved to addresses or URLs (Uniform Resource Locators). One can create a single mechanism to provide both of these functions, but we suspect that part of the negative experience of the past was that each is a potential tussle space. Hence, if a single mechanism provides both functions, the tussles from one space overflow into the other.

If we believe that IP addresses should not provide either of these functions, we need to invent one or possibly two identifier spaces. Let us consider each namespace separately. In terms of verification, each end-point in a pair or group of communicating end-points may need to verify the identity of the other(s), but this requirement only applies to the potential set of communicating entities, not the whole world. Furthermore, there may be many end-points that an end-point specifically does not want to be able to verify, or that do not want to be verifiable. We do not want to deny anonymity, only to know when it is occurring. So we can state clearly that, for purposes of verification, there may be many verification communities that do not need to be global, and that may overlap. A global namespace for verification is neither reasonable nor necessarily desirable. Rather we want to make it feasible to have many such identification spaces.

If we consider locating end-points, there are a number of possibilities that one can imagine. There are some cases in which it is critically important that identifiers reach the same end-point regardless of the location or identity of the source end-point. Email is a good example. If we send email to userXYZ@aol.com, from various points around the globe, we want it all to arrive at the same mailbox, although we do not care where that mailbox is. On the other hand, there may be many end-points that never need to be identified. For example, a "known" end-point may hand off some of its responsibilities to subsidiary end-points. These may only be accessible by telling a client their current addresses, which can never be considered permanent identifiers, because those end-points may be mobile. Only for the period of a transaction may that address be useful. We can also easily find situations in which local identifiers (such as "laser printer") is all that is needed for resolution to an address.

Thus, we can make four observations with respect to identification scope and assignment: 1) It is often reasonable to separate the identification functions into verification and resolution; 2) there may be many such identifier spaces; 3) some, but not all, of these identifier spaces need to be resolvable to the same addresses or verifiers from everywhere; 4) there can exist some end-points that have no identifiers that map to locations or verifiers.

With these issues of scope and assignment in mind, we can now explore the question of how these identifiers, whether for verification or addressing, are translated. The situation here is fairly simple, since we have recognized that for both functions there may be multiple identifier spaces and that not all end-points fall into any or even one such space. For those identifier spaces that are intended to be global, in that from anywhere they translate to the same result, some form of global translation mechanism or federation of more local translation mechanisms must be available. But for the others, it can simply be a local matter. The DNS provides us with an example of a global translation mechanism, and it is clear that it is very challenging to make such a large, complex system work well, especially under various engineering design criteria, such as speed, accurate management of updates, and network resource usage. The Grapevine experience [11] provides an interesting example of the sorts of problems that can arise when trying to coordinate such a complex, widely distributed system. There is one more point worth noting with respect to translation mechanisms. There is nothing in all of this that requires that there be one translation mechanism for all time or even at any one time, for any particular scheme. The only requirement is that if, within some scope (global or local) identifiers must be mapped in a specific way, that any and all translation schemes meet the same criteria. Translation schemes may evolve; there may be more than one at any given time, and certainly over time. The DNS [16], [17] does not support this approach, but other approaches such as URNs [24] and Active Names [27] do.

We conclude from this that it is important to separate location from the two functions of verification and access, and, to the extent these functions are needed, they can be provided by a multiplicity of services, each meeting the engineering design criteria of their user communities. These criteria derive from both designing for change and the need to cleanly define tussle-spaces.

4. An application perspective on architecture

In the past, network architects have designed the network, and then let the application designers do what they pleased, without much guidance. Today, not only do we know more about how to build nets, we know more about what application designers do. We should take this into account. By giving advice to application designers, and making clear the responsibility of the application layer, we can avoid making the network more complex.

This does not mean we should restrict what the application tries to accomplish. We cannot predict what the next application will be; hence our continued emphasis on *design for change*. The idea here is to give the application designer better advice about how to accomplish his goal, whatever it may be. To understand how we might better guide applications in their design, we need to look at network function from an application perspective.

4.1 Applications, transparency and scope

Applications are selective with respect to the features of the net on which they choose to depend. For example, the Internet email protocol SMTP just depends on a byte stream. A byte stream is a transparent service at some level—the bytes that come out are the bytes that go in, but SMTP takes no heed of the packet abstraction, nor (at a higher level) of the "features" of TCP other than reliability and ordering. So some applications are actually designed to tolerate weaker semantics than the network provides. At the same time, the Internet email service does not use an end to end approach to its design, but is implemented as a series of mail relay servers between the sender and receiver. This illustrates an important duality, which we discuss below.

The design goal of the network and the application are different. The network is designed for the application yet to come. It is optimized for change. Each application exploits what it finds, and has no need of generality beyond that. Application designers cheerfully pick and choose from the generality offered them in favor of convenience and a different kind of generality—the ability to operate over different sorts of infrastructure from the ones the network operator proudly offer, and thus to achieve a broader reach or scope than the Internet itself. So design for change for the network builder implies building for the application that has not yet emerged, while design for change for the application builder implies building for the network that has not yet emerged.

Email has a long history as an application that transcends Internet boundaries. All the networks in the 1986 survey by Quarterman and Hoskins [21] supported email, and that paper illustrates the lengths to which those users went to allow interchange of email between non-IP and IP networks. Email messages were routed at the application level to gateways that were capable of terminating the local network's reliable transport, were able to translate the headers to the destination network's addressing system, and could get the message to appropriate application servers in the destination network. Email connectivity extended beyond the Internet via connection-oriented permanent networks and on-demand telephone relays, as well as now-conventional packet forwarding networks.

Though it is less common to come in contact with it today, email continues to reach further than systems running IP. FidoNet [3] is a network relaying email between bulletin board sites via telephone connections. It has been in operation since the early 1980s and continues to operate. The Fidonet mail routing and transfer protocols are far removed from IP. Mail is routed between geographically-named mail hubs connected via non-dedicated telephone lines using a transfer protocol tuned for efficient use of telephone voice lines. In this respect, the "reach" of the application is broader than the reach of the Internet itself. Its administrators claim that more than 10,000 systems are configured to transfer email via FidoNet.[7] [13]

Architects understand that applications may span more than a single network architecture, but this idea is not viewed as part of the Internet architecture, because it seems as if by definition it is beyond it. This view is wrong. Recognition of this should

[7] Because FidoNet uses explicit route maps, that number is trustworthy.

be explicit, because it will help us understand what the "core" network needs to do.

The generality and uniformity of the network need reach no further than the point at which we wish to support the unknown application. Interconnection at the application level will always be less general but more encompassing than general Internet connection. Respect both approaches.

Consider (again) email, and imagine a mobile device that is designed only for email. Although designers of this device may find full Internet connectivity attractive, they could also use stateful conversion points as the historical systems do. Both approaches have their merits and costs. A conversion point implies a stateful point in the path of the application—email has these already. It implies that the communication path from the device to this conversion point may be outside the architected network (e.g. on the cellular network). But it may imply more efficient use of wireless capacity, or some other benefit.

Sensor networks are an emerging example. These networks tend to have very constrained complexity and power budgets, a constrained set of application goals, novel ways of routing information and so on. These networks may be "attached to" the Internet, and as a result make their applications available across the Internet. But they often will not run the Internet protocols. The interconnection point between the Internet and the sensor network will almost certainly be application-aware.

Any device too specialized to run the full Internet suite can be assumed to be too specialized to be treated as a fully general platform for unknown applications.

Many people might accept this statement implicitly, but might be unwilling to accept it as an architectural constraint. But if we accept it, it provides a clean division between two parts of the network—that part where we preserve the core principles of the Internet design (universal packet carriage, transparency, address-based routing, support for the unknown application) and those parts that use some other principles. Networks of sensors, networks of specialized wireless devices, and other technologies should be encouraged to be cleanly in one camp or the other. Either they run the general Internet suite, or they have a connection point (probably stateful) and run application-specific mechanisms[8]. *This second mode exists today, and we should ask what we need to do to support it; not just deprecate it.*

4.2 Application level framing revisited

The discussion above provides a way to resolve a standing debate about the utility of a proposal called Application Level Framing (ALF).[7] That proposal described an alternate sort of network in which there was no uniform unit of multiplexing such as the packet. The unit of transport was an application layer data unit, or ADU. The network only preserved this aspect of transparency, and was free, inside, to fragment and reassemble the ADU in arbitrary fashion. ADUs could be transported using different modes of multiplexing to match the details of the network technology—packets, cells or what have you. This idea was appealing, but had awkward implications in detail. The original proposal provided transparency at the ADU level—the ADU out was the same as the ADU in. But this view turns out to be over-simplistic. If a multiplexing unit is lost inside a network, it is not always reasonable to discard the whole ADU. We have seen the problems that arise just with IP reassembly—this would be far worse, because the ADU could be much bigger. But if the ADU can be delivered while incomplete or reconstituted based on partial retransmission, this implies that some knowledge of the "hidden" multiplexing units starts to leak out. Different applications might be tolerant of ADUs with missing or corrupted parts, others might demand totally accurate delivery or nothing, and so on. So the ADU conversion point in the network either has to be application-aware, or support a number of "general" modes of error recovery.

We conclude that if ALF is not application-aware, several of our goals will be compromised. First, since ALF error recovery will not lead to a simple form of network transparency, it will force a more careful specification of the ADU transport semantics. Second, it will demand general, stateful transformation points in the network. If, however, the transformation is application-aware, it need only preserve the semantics on which the application depends. The idea of application-aware ALF at the application layer is exactly the same idea as preserving the generality of the network only to the same extent as we preserve application neutrality. This is a valid approach, and should be made explicit.

4.3 Applications and naming

If applications wish to have a reach, or scope, that is larger than the Internet, then it follows that they need to be able to name application end-points in ways that are not restricted to the Internet context. The design of email provides email "addresses" general enough that they can name entities outside the Internet. There are several approaches for doing this, including embedding one kind of name inside another with clever syntax, or building an application-specific naming service that is managed coherently to name points in all the desired networks. But as part of designing an application, the designers must decide how they will name their end-points, and whether the application will include a routing/forwarding function at the application level, as Internet email does. As we discussed earlier in Section 3.4, questions about the universality, persistence, and scope of translations of identifiers, whether at the IP routing, transport, or application layers must be considered carefully, in order to provide the desired functionality balanced against various tussles. The problems are no different in the domain of applications, although the choices may be, since these identifiers may also need to be user or human friendly.

4.4 Giving advice to the application designer

Guidance to the application designer might be captured in the form of a set of questions:

- What is the definition of transparency on which the application depends?
- What is the desired scope of the application?
- What sort of application-level naming and addressing is needed so that the application can have the needed scope?
- Does the application benefit from (or require) relay points?

[8] "Application-specific mechanisms" can include using parts of Internet mechanisms, The important distinction is the sacrifice of broad generality.

- If so, what parts of the data need to be visible for transit? What parts can be encrypted end to end? Are there regions of trust within which the data need not be protected by encryption, and does this make the design of the edge regions easier?
- Does the application depend on end-to-end verification to insure reliable delivery, or on robust relaying? Does the application need to depend on "immediate" bi-directional communication?

By asking these sorts of questions of the application designer, we emphasize what problems the network and the application each have to solve. We also return to the three tenets with which we began this paper, now considering the boundaries between network and application: the impact of designing for change, controlled transparency and negotiation in tussle-spaces. If the application builder wants a greater scope than that of the Internet, we shift to him the necessity of dealing with network technology so specialized that the Internet abstraction cannot easily be made to work over it. (With luck, some other network designer has dealt with the quirks of that technology, so the application designer does not have to be a network specialist.) We remove from the Internet naming conventions the need to name entities that are outside the reach of the Internet.

As we shift this responsibility to the application designer, we might at the same time build tools that support his needs. These might include new sorts of region-boundary services, new naming services, and so on. But by this distinction, we capture the idea that these new services do not need to capture the full generality of the Internet transparency.

5. Revisiting the stateless faith

Internet believers praise the objective of the stateless network, often ignoring the reality. Of course, the network is not stateless; it has routing state at a minimum. But the normal interpretation of "stateless" is that there is no state per "connection", whatever a connection is. One test we apply to see if the network is stateless is whether any sort of setup is required before sending a packet. This is a laudable goal; demanding any sort of setup does add undesirable overhead and complexity to interaction, especially since many interactions are very brief and involve few packets. But it is clear that there is state inside the network that does relate to individual connections, and the quantity of such state is growing. State is particularly related to regions and their boundaries. What is desirable is a general, reusable set of architectural mechanisms to manage this state. To arrive at this mechanism, it is helpful to consider the varying purposes this state serves, as a starting point towards identifying common characteristics that can be abstracted.

State in NAT boxes does address translation. It is created when the first packet arrives, using a preset algorithm, so no explicit setup from the endpoint is required. State in firewalls usually sits at region boundaries. It is not usually per connection, but has per-connection consequences. State is also found at points where network-specific reformatting is performed (e.g. breaking packets into cells), where QoS is implemented, and where headers are compressed. There is also a great deal of application-level state in the network.

Our affirmation above that packets are still a good choice does not mean that the packet design currently used in the Internet - stateless connectionless datagrams - is necessarily the way to go. We need a new set of design principles that reflect this reality: such as end-driven reconstitution of state after a failure, ability to inspect the state to determine why communication is failing (why transparency is missing), and so on. Taken together, these design tenets begin to define a shared, evolvable soft state support subsystem within the Internet architecture.

In particular, the goal of controlled transparency may be best achieved with some careful use of state in the network, so that the first packet in a transfer carries some extra trust-defining information, which is cached and reused on subsequent packets, and which can be reconstituted if it is lost. This sort of zero-round trip soft state may be a relatively benign design compromise.

6. Conclusions

The tenet of design for change is, from one perspective, an argument for architectural minimalism – a small, carefully limited set of global invariants and shared functions, with all else left to more local definition. Once an infrastructure becomes embedded and dominant, any point of global commonality will become almost impossible to change or migrate. Of course, having alternative solutions to every problem is an open door to loss of interoperability, confusion at the application and user level, and so on. But clever designers understand how to install escape paths to avoid being locked in to a bad decision. The prefix to URLs (http, https, etc.) was intended to allow them to name "things" in multiple name spaces without the user having much care about what was happening.

Perhaps the most radical idea from this analysis is that the simple, end-to-end transparency model should be replaced with the more complex idea of controlled transparency. This implies active elements in the network, which in turn implies a tussle over who controls these devices. It also implies that we need to specify what impact these devices have on the semantics on which the applications depend on.

7. ACKNOWLEDGMENTS

This work was funded under two grants from the Department of Defense Advanced Research Projects Agency, F30602-2-0553 (MIT) and F30602-00-1-0540 (ISI).

We would like to thank the other members of the NewArch team: Steve Bellovin, Bob Braden, Noel Chiappa, Aaron Falk, Mark Handley, and James Sterbenz for arguing and challenging every step of the way, and our reviewers for their insightful comments.

8. REFERENCES

[1] I. Baldine, H. Perros, G. Rouskas, D, Stevenson, *JumpStart: A Just-in-Time Signaling Architecture for WDM Burst-Swtiched Networks*, **Proc. Networking 2002**.

[2] Y. Bernet, *The Complementary Roles of RSVP and Differentiated Services in the Full-Service QOS Network*, **IEEE Communications Magazine**, Vol. 38, No. 2, February, 2000.

[3] R. Bush, *FidoNet: technology, tools, and history*, **Communications of the ACM**, Vol. 36, No. 8, August 1993.

[4] B. Carpenter, *Internet Transparency*, **RFC 2775**, February 2000.

[5] W. Cheswick and S. Bellovin, *Firewalls and Internet Security: Repelling the Wily Hacker (1st ed.),* **Addison Wesley**, 1994.

[6] D. D. Clark, *Design Principles of the Internet Architecture,* **Proc. ACM SIGCOMM 1988**, Stanford, CA, USA, August 1988.

[7] D. D. Clark and D. Tennenhouse, *Architectural Consideration for a New Generation of Protocols*, **Proc. ACM SIGCOMM 1990**, Philadelphia, PA, USA, September 1990.

[8] D. D. Clark, J. Wroclawski, K. Sollins, R. Braden, *Tussle in Cyberspace: Defining Tomorrow's Internet*, **Proc. ACM SIGCOMM 2002**, Pittsburgh, PA, USA, August 2002. Also to appear. **IEEE/ACM Transactions on Networks**.

[9] M. E. Crovella, A. Bestavros, *Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes*, **IEEE/ACM Transactions on Networking**, Vol. 5 No. 6, December 1997, pp 835-836.

[10] L. Daigle, D. van Gulik, R. Iannela, P. Faltstrom, *URN Namespace Definition Mechanisms*, **RFC 2611**, June, 1999.

[11] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, *Epidemic Algorithms for Replicated Database Maintenance*, **Proc. Sixth Symposium on Principles of Distributed Computing**, Vancouver, B.C., Canada, August 1987, pages 1-12.

[12] Dept. of Defense, *Trusted Computer Systems Evaluation Criteria* **DoD 5200.28-STD**, August 15, 1983. (Also known as the "Orange Book".)

[13] *FidoNet Web Site*, **http://www.fidonet.org**

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

[15] S. Kent, C. Lynn, K. Seo, *Secure Border Gateway Protocol (S-BGP)*, **IEEE JSAC**, Vol. 18 No. 4, April 2000, pp. 582-592.

[16] P. V. Mockapetris, *Domain names - concepts and facilities,* **RFC 1034**, Nov. 1, 1987.

[17] P. V. Mockapetris, *Domain names - implementation and specification*, **RFC 1035**, Nov. 1, 1987.

[18] R. Moskowitz, unpublished documents on the Host Identity Payload Architecture, 2001. Currently the work only exists as Internet Drafts.

[19] National Research Council, Committee on Information Systems Trustworthiness, *Trust in Cyberspace*, **National Academy Press**, 1999.

[20] J. Postel, ed., *Internet Protocol*, **RFC 791**, Std 5, September 1981.

[21] J. S. Quarterman and J. C. Hoskins, *Notable Computer Networks*, **Communications of the ACM**, Vol. 29 No. 10, ACM, October 1986, pp. 932-970.

[22] C. Qiao and M. Yoo, *Optical Burst Switching (OBS) – A New Paradigm for an Optical Internet*, **Journal of High Speed Networks**, 1999.

[23] J. Saltzer, D. Reed, D. Clark, *End-to-End Arguments in System Design,* **ACM Transactions on Computer Systems,** Vol. 2 No. 4, November 1984, pp. 277-288.

[24] K. Sollins, *Architectural Principles of Uniform Resource Name Resolution*, **RFC 2276**, January, 1998.

[25] K. Sollins and L. Massinter, *Functional Requirements for Uniform Resource Names*, **RFC 1737**, December, 1994.

[26] J. Turner, *Terabit Burst Switching*, **Journal of High Speed Networks**, 1999.

[27] A. Vahdat, M. Dahlin, T. Anderson, A. Aggarwal, *Active Names: Flexible Location and Transport of Wide-Are Resources*, **Proc. 2nd Usenix Symposium on Internet Technologies and Systems**, October, 1999.

[28] W. Willinger, M. Taqqu, R. Sherman, D. Wilson, *Self-Similarity Through High Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level*, **Proc. ACM SIGCOMM 1995**, Cambridge, MA, USA, August 1995, pp. 100-113.