Internet Evolution and the Role of Software Engineering

Pamela Zave

AT&T Laboratories—Research, Florham Park, New Jersey, USA,
pamela@research.att.com,
WWW home page: http://www2.research.att.com/TILDEpamela

Abstract. The classic Internet architecture is a victim of its own success. Having succeeded so well at empowering users and encouraging innovation, it has been made obsolete by explosive growth in users, traffic, applications, and threats. For the past decade, the networking community has been focused on the many deficiencies of the current Internet and the possible paths toward a better future Internet. This paper explains why the Internet is likely to evolve toward multiple application-specific architectures running on multiple virtual networks, rather than having a single architecture. In this context, there is an urgent need for research that starts from the requirements of Internet applications and works downward toward network resources, in addition to the predominantly bottom-up work of the networking community. This paper aims to encourage the software-engineering community to participate in this research by providing a starting point and a broad program of research questions and projects.

1 Introduction

In a recent article on the future of software engineering [1], Mary Shaw identified the Internet as both the dominant force for the last 15 years in shaping the software milieu, and the source of software engineering's greatest coming challenges. The challenges center on software that is widely used by—and often created by—the public, in an environment that is ultra-large scale and has no possibility of central control.

Software research has made huge contributions to Internet applications and middleware [2]. These successes have been enabled by an Internet architecture that has provided a relatively stable and universal interface to lower network layers.

Currently, however, Internet architecture is a controversial subject in the networking community. Peoples' ideas are changing fast, and it is increasingly likely that the future Internet will not have an architecture in the sense that it has had one in the past.

The first goal of this paper is to inform software researchers about current trends in the networking community and their likely effect on the future Internet. Sections 2 and 3 summarize the nominal Internet architecture and the reasons it is inadequate today, both as a characterization of the current Internet and as a basis for meeting projected requirements. Section 4 describes how networking is changing in response to these pressures, and where the current trends are likely to lead.

Sections 3 and 4 show that there is an urgent need for research that starts from the requirements of Internet applications and works downward toward network resources, in addition to the predominantly bottom-up work of the networking community. The second goal of this paper is to encourage the software-engineering community to participate in this research. Section 5 provides a starting point in the form of a well-defined and apparently general architectural unit. Based on this foundation, Section 6 presents a broad program of research questions and projects. The paper concludes that Internet evolution presents exciting and important research challenges that the software-engineering community is best qualified to take up.

2 The "classic" Internet architecture

Although the Internet architecture is not precisely defined, there is broad agreement on the properties and protocols in this section. The Internet is organized into layers, a concept that was brought into operating systems by Dijkstra [3] and then adopted for networking. Usually the Internet is described in five layers, as shown in Figure 1.

The physical layer provides data transmission on various media. A link is a machine-to-machine data connection using a particular medium. Links are partitioned into groups called subnetworks. A machine can belong to (be a link endpoint of) more than one subnetwork, in which case it acts as a gateway between them. The architecture is designed to allow great diversity in the physical and link layers.

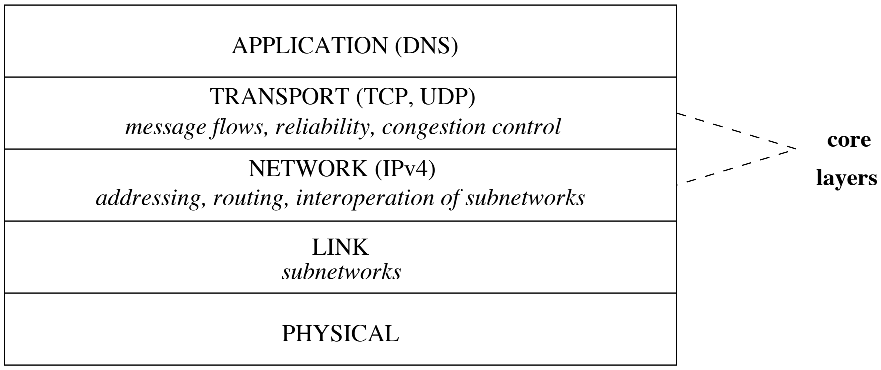
APPLICATION (DNS)
TRANSPORT (TCP, UDP)
message flows, reliability, congestion control
NETWORK (IPv4)
addressing, routing, interoperation of subnetworks
LINK
subnetworks
PHYSICAL
core
layers

Fig. 1. The five layers of the classic Internet architecture.

The network layer is one of the core Internet layers. It is referred to as the "narrow waist" of the architecture because it is defined by the single Internet Protocol (IP), which runs on all subnetworks. Each machine has a unique 32-bit address. The address space is organized as a location hierarchy, which allows global routing with routing tables of reasonable size. Each machine is either a *host* or a *router*; hosts run applications and serve users, while routers deliver packets. IP provides best-effort packet delivery between the machines.

The transport layer is the other core Internet layer. It is defined primarily by the two transport protocols User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). UDP sends and receives individual packets. TCP implements reliable, FIFO, duplicate-free byte streams (two-way connections). TCP also implements congestion control.

In the application layer, domain names as supported by the Domain Name System (DNS) are available as a global mnemonic name space. Although all other functions are the responsibility of the applications themselves, there are many functions that are common to a variety of applications. These functions are sometimes implemented and deployed as shared or re-usable middleware. The functions include:

- management of application-specific name spaces;
- directories for discovery of resources and services;
- endpoint mobility;
- enhanced inter-process communication facilities such as remote method invocation, transactions, multi-point connections, multicast, publish/subscribe event-based communication, and trusted-broker services;
- security (authentication, access control, encryption, protection from denial-of-service attacks);
- protocol conversion and data reformatting;
- buffering and caching;
- monitoring and billing for quality of service.

The classic Internet architecture was strongly shaped by the "end-to-end" principle [4], which says that in a layered system, functions should be moved upward, as close as possible to the applications that use them. If this principle is followed, applications do not incur the costs and complexity of core functions that they do not use, nor do they contend with core functions that conflict with their goals.

Originally TCP was the only IP transport protocol. In the best-known application of the end-to-end principle, TCP was separated from IP, and UDP was added, when the early designers realized that TCP is ill-suited to real-time traffic such as voice [5]. TCP reliability and ordering introduce delays (while lost packets are retransmitted). Voice protocols cannot tolerate delay, and they deal with lost packets by other means such as interpolation.

A different and equally influential principle that is usually confused with the end-to-end principle is the "fate sharing" principle.[1] The fate sharing principle

[1] The confusion arises because the name "end-to-end" actually fits the fate sharing principle better than it fits its own principle.

says that it is acceptable to lose the state information associated with an entity only if, at the same time, the entity itself is lost [5]. In the best-known application of the fate-sharing principle, all TCP functions run in the hosts at the endpoints of TCP connections. Even if there are transient failures in the network, the hosts can continue to communicate without having to reset their states.

3 The real Internet

The classic Internet architecture is a victim of its own success. Having succeeded so well at empowering users and encouraging innovation, it has been made obsolete by explosive growth in users, traffic, applications, and threats.

3.1 Network management and operations

The Internet was not designed with management in mind, yet the administrators of today's networks face critical problems of configuration, traffic engineering, routing policy, and failure diagnosis. Their tools for understanding network traffic are poor, and their mechanisms for controlling network operations do not offer a predictable relationship between cause and effect.

Most of the Internet consists of autonomous subnetworks that are either operated by organizations for their own purposes, or operated by service providers for profit. In this environment, economic incentives work against many important goals [6]. No network operator has the power to provide better end-to-end quality of service (including bandwidth or latency guarantees, high availability, and fast DNS updates) across the boundaries of autonomous subnetworks. Because no network operator has the power to provide it, no operator can charge for it, and so no operator has an incentive to support it. Furthermore, no operator has a strong incentive to implement improved technology for global network management, because improvements yield no benefit until most other autonomous subnetworks have also implemented them.

Internet routing is beginning to have serious problems of scale. The routing table in a typical router now has 300,000 entries, and these must be stored in the fastest, most expensive types of memory to maintain routing speed.[2] There are efforts to move toward a scheme in which a typical routing table has one entry per autonomous system, which points to a router that can route to all the addresses for which that autonomous system is responsible.

3.2 Middleboxes

To understand the issues of Internet software, it is important to consider *middleboxes*. A *middlebox* is a stateful server that lies in the middle of a communication path between endpoints. It is distinguished from hosts, which are the endpoints

[2] One of the main obstacles to acceptance of IPv6 is the high cost of routers able to handle 128-bit IP addresses.

of communication paths. It is also distinguished from routers, which maintain no state concerning particular instances of communication. A middlebox can perform any function desired [7], in contrast to routers, which are constrained to forwarding packets without altering them.

Middleboxes are ubiquitous in the Internet, as will be discussed in the next two subsections. They have always been controversial, partly because of the unfortunate things that have been done with them, and partly because every middlebox violates the principle of fate sharing.

3.3 Network and transport layers

By the early 1990s two deficiencies of the classic Internet architecture had become recognized as major problems:

- The 32-bit IPv4 address space is too small for the demand.
- There is no provision for secure private subnetworks.

The address space was effectively extended by Network Address Translation (NAT), which allows many machines on a private subnetwork to share a single public IP address. Private subnetworks were fenced off by firewalls. Both are implemented by middleboxes, and NAT and firewall functions are often combined in the same middlebox. These middleboxes are embedded so completely in the network (IP) layer that basic understanding of Internet reachability requires taking them into account [8].

NAT boxes and firewalls are everywhere: all business enterprises and Internet service providers have them—maybe even several tiers of them—and they are packaged with most residential broadband offers. Despite their obvious value, NAT and firewalls do enormous harm from the perspective of those who need to build and deploy applications. This is not because their functions or middleboxes in general are intrinsically harmful, because they are not [9]. The harm is done by heedless conflation of the concerns of network, transport, and application layers.

To begin with NAT, Figure 2 shows a typical NAT box on the border between a private subnetwork and the public Internet. Many different private subnetworks re-use the same "private" portion of the IP address space. This means that some of the nodes in the subnetwork have distinct private addresses but share a small number of public IP addresses, which are the addresses of the subnetwork's NAT boxes.

On its public side the NAT box has an IP address and many ports.[3] A public port encodes a private *address, port* pair, so that a node on the subnetwork can communicate on the open Internet. An example of this encoding is shown in Figure 2. Because there are obviously fewer public ports than private *address, port* pairs, most public ports are allocated dynamically by the NAT box and released as soon as possible for re-use.

[3] There are many variations in the way that NAT boxes and firewalls work. This section describes common and representative schemes.

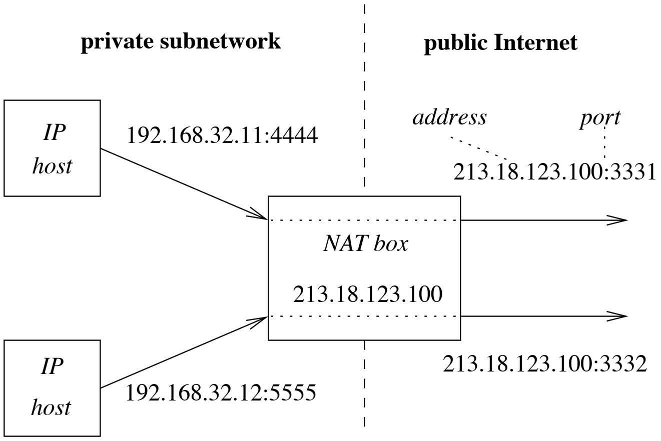

Fig. 2. Source addresses and ports altered by Network Address Translation.

Because of NAT, many machines on private subnetworks do not have persistent public addresses. It is impossible for applications on machines outside the subnetwork to initiate communication with the applications on such a machine, even if this would be useful and firewall security is configured to allow it. In this case, application-layer concerns are sacrificed for network-layer goals.

Port numbers are transport-layer mechanisms used by TCP and UDP to distinguish different sessions or applications at the same node (applications are named by well-known ports). Yet NAT boxes alter port numbers for network-layer reasons. As a result of this conflation of concerns, applications can no longer use them the way they are supposed to be used. For one example, even if a NAT box is configured to leave well-known ports alone, at most one node of the private subnetwork can receive requests at a well-known port. For another example, remote port manipulation by NAT boxes means that an application cannot observe, control, or communicate its own port numbers.

Turning to firewalls, a NAT box often acts as or in conjunction with a firewall. In its default setting, a firewall only allows communication initiated from inside the private subnetwork. A node in the subnetwork sends a message to an IP address and port on the public Internet. Both source and destination *address, port* pairs are stored in the firewall, and only subsequent messages from the public destination to the private source are allowed in.

Peer-to-peer communication can only be established if one endpoint initiates a request and the other endpoint accepts it. Until 2004 it was considered impossible to establish a TCP connection between two peers behind different default firewalls. Even now it only works 85-90% of the time, because the solution is a hack depending on the details of firewall behavior, which vary widely [10].

All of these issues are relatively unimportant for client-server applications such as Web services, at least when the clients are private and the servers are public. The issues are critical, however, for peer-to-peer applications such as voice-over-IP.

There is a large selection of mechanisms to make peer-to-peer applications such as voice-over-IP possible. However, all of them have major flaws or limitations. NAT boxes and firewalls are sometimes configured manually to allow this traffic, but this is too difficult for most people to do, and violates the security policies of large organizations. Clients, application servers, NAT boxes, and firewalls can all cooperate to solve the problems automatically, but this requires a degree of coordination and trust among software components that is rarely feasible. *Ad hoc* mechanisms such as STUN servers[4] exist, but these mechanisms are fragile, and do not work with common types of NAT, types of firewall, and transport protocols.

Finally, when implementors cannot use TCP because of firewalls, they sometimes re-implement the features of TCP that they want on top of UDP.[5] In addition to the waste of effort, this is almost certain to subvert TCP congestion control.

3.4 Middleware and applications

Applications need a broad range of effective mechanisms for security and endpoint mobility. Although is not clear whether these mechanisms should be in networks, middleware, or applications (or all three), it is clear to everyone that current capabilities are inadequate.

Many of the middleware functions listed in Section 2 cannot be provided in accordance with fate-sharing, or cannot be provided as well [11, 12]. For example, the only way to protect endpoints against denial-of-service attacks is to detect malicious packets before they get to the endpoint. So this function must be provided in the network, even though detecting malicious packets often requires maintaining state information that violates the principle of fate-sharing.

Application middleboxes were more controversial ten years ago. By now the philosophical debate seems to be over, with the trend toward “cloud computing” having finalized the legitimacy of servers in the network. Furthermore, fate sharing is less important now because network servers are commonly run on high-availability clusters.

Although the debate is over, a serious problem remains: the Internet architecture does not support application-level middleboxes. Consider, for example, the problem of introducing Web proxies into the paths of HTTP messages. There are many kinds of Web proxy, providing functions such as caching, filtering, aggregating, load-balancing, anonymizing, and reformatting. They serve the interests of diverse stakeholders such as users, parents, employers, Internet service providers, and Web content providers.

[4] A machine behind a NAT box can query a STUN server to find out what address and port the NAT box has currently assigned to the machine.

[5] Because UDP does not require an initial handshake, it is easier to traverse firewalls with UDP.

Figure 3 shows the desired path of an HTTP request, from a browser on an employee's desk through three proxies:

- a privacy proxy desired by the employee, to reduce information leakage and annoyance;
- a filtering proxy desired by the employing enterprise, to block access to some Web sites;
- a caching proxy desired by the Internet service provider, to improve performance and conserve resources.

The choices for implementing this chain by introducing the middleboxes into the path of the HTTP request are not attractive.

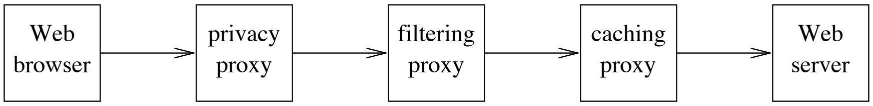

Fig. 3. The path of an HTTP request.

First, a browser or proxy can be configured with the address of the next proxy in the path. This is administratively burdensome and very inflexible. It can also be circumvented by one stakeholder acting against the interest of another stakeholder. For example, why should the privacy proxy (or the Web browser, if there is no privacy proxy) direct requests to the filtering proxy against the user's own interest?

Second, an implementation of IP can look inside messages to find the HTTP requests, and route them in a special way. Finally, a local DNS server can be programmed to answer a query for the IP address of a Web service with the IP address of a local proxy instead. These mechanisms are both messy, burdensome, and inflexible. They break the rules of IP and DNS, respectively, and are only available to some stakeholders.

Because of problems such as these, protocols such as Mobile IP [13] and IP multicast [14] rely on special IP routers, addresses, and routing. In this way their designers avoid the limitations that most application programmers must accept.

3.5 Principles and priorities

The range of problems observed today is not surprising. The Internet was created in simpler times, among a small club of cooperating stakeholders. As the Internet becomes part of more and more aspects of society, it will inevitably be subject to more demands from more stakeholders, and be found deficient in more ways [15].

This final section of the status report turns to impressions of the principles and priorities that have shaped the Internet so far. These impressions are subjective and not completely fair, because the Internet has been shaped by historical, political, and economic forces beyond anyone's control. All of the tendencies, however, are readily apparent in current networking research as well.

The networking community has a good understanding of the demands of scalability and the value of hierarchy in meeting those demands. I would not say it has a good understanding of the end-to-end principle, because it is usually confused with the principle of fate sharing. Performance, availability, and efficiency are the highest priorities by far.

There are some principles and ideas, understood very well by software engineers, that appear to be unfamiliar to the networking community—or at least to large segments of it:

Complexity matters. The trouble with software is that it can do anything, no matter how complex, convoluted, fragile, incomprehensible, and ill-judged. Software engineers understand the cost of such complexity. Because the networking community underestimates the cost of complexity, it pays no attention to one of the most important problems of the current Internet, which is that it is much too difficult to build, deploy, and maintain networked applications.

Separate concerns. "Tussle" describes the ongoing contention among stakeholders with conflicting interests, as they use the mechanisms available to them within the Internet architecture to push toward their conflicting goals [15]. Because the mechanisms they use are always intertwined with many other mechanisms, and each mechanism was designed to address many concerns, the result of tussle is usually to damage efforts to reach unrelated goals.

With the right abstraction, a structure can be both simple and general. Here the best example is the application protocol SIP [16], which is the dominant protocol for IP-based multimedia applications, including voice-over-IP. SIP is currently defined by 142 standards documents totaling many thousands of pages (and counting) [17]. Each new requirement is met with a new mechanism. There seems to be no conception that a protocol based on better abstractions could be both simpler and more general.

Think compositionally: there is no such thing as a unique element. Networking is full of things that are treated as unique, when they are actually just members of a class that need to be considered in relation to one another and often composed in some way. "The stakeholder deciding on the proxies for an HTTP request" is one such example, as described in Section 3.4, and "a communication endpoint" is another. For example, if a proxy blocks a request for some reason, then the proxy is the endpoint of the communication, rather than the IP host to which it is addressed. Compositional thinking will be discussed further in Section 6.

From the viewpoint of Internet users and application programmers, there are requirements that sometimes equal or exceed performance, availability, and efficiency in priority. These include ease of use, correctness, predictability, and modularity. The use of functional modeling and formal reasoning to help meet such requirements is all-but-unknown in the networking community.

4 Internet trends and evolution

4.1 Trends in practical networking

As we would expect, people have found a variety of ways to work around the shortcomings of the Internet as they perceive them. In addition to code workarounds, two of them operate on a larger scale.

First, many new networks have gateways to the Internet but do not run IP. These include mobile telephone networks,[6] and also many other wireless, enterprise, and delay-tolerant networks.

Second and most important from the perspective of software engineering, enterprises deploy what they need in the form of Internet *overlays*. An *overlay* is a custom-built distributed system that runs on top of the core layers of a network, for the purpose of simulating a different or enhanced architecture. In general an overlay consists of user machines, servers, protocols, data structures, and distributed algorithms. In general an overlay can cross the boundaries of autonomous subnetworks.

Many overlays run on the current Internet. Akamai runs a content-delivery overlay providing high availability, low latency, and fast DNS lookup/update [18]. Skype achieved instant popularity as a voice-over-IP service partly because it was the first such peer-to-peer service to penetrate firewalls successfully. Many global enterprises rely on Virtual Private Networks (VPNs), which are overlays, for security.

Returning to the discussion of voice-over-IP in Section 3.3, Skype uses an overlay solution to the firewall problem as follows [19]. Users on private subnetworks initiate TCP connections to servers or designated peers on the open Internet. These servers connect with each other, and all messages between peers on private subnetworks are relayed by these servers. This is a very inefficient way to transmit high-bandwidth real-time voice data, which should travel between endpoints by the shortest possible path [20].

4.2 Trends in networking research

In the networking community, researchers have been working on deficiencies in the Internet architecture for at least a decade.

In the first half of the 2000s, the research climate was dominated by the belief that research is pointless unless its results can be adopted easily within the existing Internet. Attempting to work within this constraint, people realized that the current architecture makes solving some problems impossible. The idea of a "clean slate" architecture became popular, if only as an intellectual exercise that might lead to fresh ideas and a more scientific foundation for research. Under the title "Next Internet," clean-slate research gained funding and attention worldwide. Researchers gave serious attention to how the results of such work might find their way into practice [21, 22].

[6] The public landline telephone network does not run IP either, but it would not be expected to do so, as it preceded IP by a century.

Even before the question of how results could be used, a clean-slate research agenda faces the question of how new ideas can be evaluated. For an experiment to be convincing, the experimental network must be large-scale and open to public use.

The answer to this question, according to widespread opinion in the networking community, is *virtualization*. This refers to technology for supporting, on a real network, a set of independent virtual networks. Virtualization must guarantee each virtual network a specified slice of the underlying network resources, and must also guarantee that applications and data on different virtual networks are isolated from one another. Virtualization can provide a large-scale test-bed for multiple parallel experiments, at reasonable cost [23]. Virtualization technology itself is now a major subject of research.

There are two ways that these research trends could lead to success. One is that a particular “Next Internet” architecture emerges as the leader and is adopted globally as the Internet architecture for the next few decades. The other is that no architecture emerges as the leader, yet virtualization works so well that no single architecture is required. These are the “purist” and “pluralist” outcomes, respectively [23].

How would a pluralist outcome be different from the current situation, in which more and more stakeholders resort to overlays? As envisioned, the pluralist outcome has three major advantages:

- Provided that virtualization comes with the right economic incentives, a virtual network can get a guaranteed allocation of resources end-to-end, so it can offer predictable quality of service to its users.
- Virtualization is intended to expose the capabilities of the underlying hardware as much as possible, so that virtual networks can monitor and exploit them better than is possible now.
- A virtual network is a clean slate, without the obstructions and complications of existing core layers that current overlays must contend with. In allowing applications to get as close to the physical and link layers as necessary, it is a return to the end-to-end principle.

4.3 Prospects for evolution

Concerning virtualization and “Next Internet” investigations, Roscoe expresses a popular viewpoint in regarding the pluralist outcome, not as a consolation prize, but as the preferred alternative [24]. Certainly it seems the more likely result, given that the purist outcome requires the kind of global consensus that is very difficult to achieve in any public concern.

If neither of these outcomes takes hold, then the Internet will certainly continue evolving in ways that make the classic IP architecture less important. There will be more networks that do not use IP, and more overlays designed to avoid various aspects of it. Overlays will become “application-specific networks” because all networking problems, from application programming to traffic engineering and monitoring, are easier if the application is known.

From a top-down viewpoint, all the possible paths of evolution look similar. There is certainly a new openness to clean-slate thinking and application-specific thinking. Hopefully there is also increased appreciation of the richness of networked applications and the importance of fostering innovation by making them easier to build.

5 A pattern for network architecture

This section defines an *overlay* as a unit of network architecture. In [25], Day argues convincingly that this is the only type of architectural unit. There can be many instances of the overlay type, differing in their ranks, scopes, and properties. The choice of the term "overlay" is discussed in Section 5.4.

Day's pattern for overlays is unusually thorough and explicit. The presentation in Sections 5.1 through 5.2 is a much-abbreviated form of it.

Day's pattern is also well-thought-out and appears to be quite general. For this reason, it seems to be an excellent starting point for further research on network architecture.

5.1 Definition of an overlay

An *overlay* is a distributed inter-process communication facility. It has *members*, each of which is a process running on some operating system on some machine. It also has a *name space* and some mechanism for enrollment (either dynamic or by static configuration) by which processes become members and get unique names from the name space. An overlay process belongs to only one overlay. The membership of an overlay is its *scope*.

An overlay offers a communication service to overlays and applications above that use it. From the perspective of the upper overlay (or "overlay" for short), the programming interface to the lower overlay (or "underlay" for short) gives overlay members the abilities to:

- register and authenticate themselves with the underlay;
- provide information to the underlay about their location, *i.e.*, a member of the underlay on the same machine;
- provide information to the underlay about which overlay members have permission to communicate with them;
- authorize expenditures of resources in the underlay;
- request a communication session with another member of the overlay (by overlay name), and receive an identifier for it;
- accept a requested communication session (associated with the requestor's overlay name) and receive an identifier for it (which need not be the same as the requestor's identifier);
- send or receive messages in a session, using its identifier; and
- close a communication session.

As with enrollment, an overlay can give information to an underlay dynamically or by static configuration.

To implement the sessions of the communication service that it offers, an underlay includes mechanisms to:

- maintain directories with the locations and permissions of registered overlay members;
- enforce the permissions and authorizations of the overlay;
- allocate, manage, and release resources, which often include sessions in other underlays below itself;
- route and forward messages, which may have data units different from those of the overlay;
- perform error control and congestion control on transmission of messages; and
- account for resources used by the overlay.

Returning to a focus on a single generic overlay, the members of an overlay are sometimes partitioned into hosts and routers, but this is not a necessary distinction. The *rank* of an overlay is its place in the use hierarchy.

In Day's view all of the classic IP core is one overlay, including IP, TCP, UDP, and DNS.[7] This is consistent with the nature of current non-IP overlays, which contain all these parts. For example, *i3* is a well-known overlay suitable for a variety of purposes [26].

5.2 Private subnetworks

Figure 4 shows an arrangement of overlays suitable for an application that traverses the boundaries of private networks. A and D are application processes. The application (described as an overlay, although it may not export a service for higher applications) must have at least as many processes as participating user machines. The application has a permission scheme in which A can initiate a session with D. All processes are labeled with their names within their overlay, and registered as co-located with the processes directly below them in the diagram.

Below the application is an overlay for the application infrastructure, containing a process on every machine that participates in the application, whether user or server/router. For the session between A and D, this overlay checks permissions and chooses route **a, b, c, d**.

At the lowest rank of the figure there are three overlays, a public network and two private networks. Processes in these overlays represent points of attachment to these networks. As we can see in the figure, processes **b**, *b*, and *b'* are co-located at a machine that serves as a gateway between public and private networks, as are processes **c**, *c*, and *c'*. The arrows show the path of a message from A to D.

This structure provides opportunities for security that do not depend on discriminating arbitrarily against peer-to-peer applications, as firewalls usually do. Security is discussed further in Section 6.4.

Each network overlay need only have a name space large enough for its members, and it is independent of the name space of any other overlay. Thus the

[7] The distinction between TCP and UDP is de-emphasized.

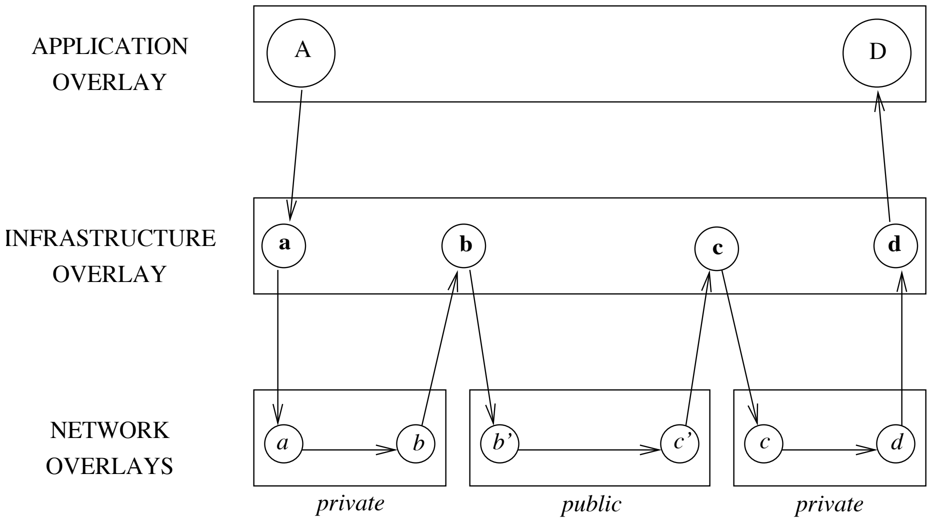

Fig. 4. Interoperating networks are overlays with the same rank.

structure is as good as NAT for managing name spaces of limited size. Application processes refer to each other by names in their own overlay, regardless of where they are located or attached.

5.3 Mobility and multihoming

Mobility is the movement of processes in one overlay with respect to other overlays. Figure 5 is similar to Figure 4, except that the vertical lines show the registered locations of processes. The dashed lines are permanent locations while the dotted lines are transient locations.

Initially **b** is located at *b* in the leftmost network, and **a** can reach **b** in one hop in the infrastructure overlay. Later the machine containing B and **b** moves to the rightmost network, so that it loses its point of attachment *b* and gains a point of attachment *b'*. At this later time, the infrastructure overlay must route messages from **a** to **b** through **c**.

Throughout the movement of this machine, the infrastructure overlay can maintain a session between A and B, because it always has the ability to reach **b** from **a**. This illustrates that the scope of an overlay includes all processes that can reach one another without forwarding in a higher overlay. In contrast, processes *c* and *c'* are not in the same overlay, and cannot reach each other without forwarding in the infrastructure overlay.

Mobility and multihoming are different points on a continuum. With multihoming, a process maintains multiple points of attachment, possibly to different overlays, on a permanent basis. For example, **c** is multihomed to *c* and *c'*. With

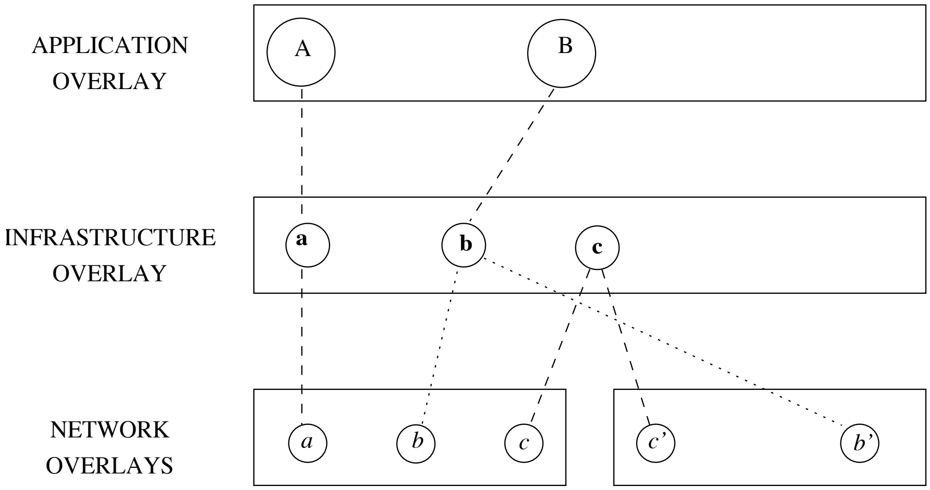

Fig. 5. Mobility is a dynamic relationship between overlays.

mobility, a process may never have two points of attachment at the same time, or have two only transiently.

5.4 Overlay composition

Previous work on overlay composition has proposed two operators for composing overlays, *layering* and *bridging* [27, 28]. *Layering* is the use of one overlay by another overlay of higher rank. *Bridging* is an operator that composes two overlays of the same rank, such as the networks in the bottom ranks of Figures 4 and 5.

As these figures show, bridging is a top-down concept, something that an overlay does with two overlays below it. As an operator joining two overlays of the same rank, bridging is always under-specified and probably a step in the wrong direction. A proper operator would have to specify which are the common elements of bridged overlays, when and how a message is forwarded from one overlay to another, and what service the composed overlays offer to a higher level—all very difficult to do from the bottom up. Such errors are common in networking, due to the chronic absence of specifications (whether formal or rigorous but informal).

I take as a working hypothesis—to be exploited and evaluated—that Day's pattern is the correct unit of architecture. Given its importance, its name is also important. The three candidates are "layer," "overlay," and Day's name Distributed IPC Facility (DIF, where IPC is Inter-Process Communication). "Layer" is not a perfect name because it is too easy to misunderstand it as a rank (as in Figure 4) or as a global structure that all vertical paths must traverse (as in Figure 1, or in cakes). "Overlay" is not a perfect name because of its implication that it must *lie over* something else, which virtualization aims to make less relevant. Acronyms, and especially recursive ones, impede intuition. Because the confusion caused by “layer” seems much more serious than the confusion caused by “overlay,” I have chosen “overlay.”

6 Research challenges, from the top down

Internet evolution is creating interest in a large number of research questions concerning network architecture, many of which should be answered from a top-down, application-centric viewpoint rather than a bottom-up, resource-centric viewpoint. There should be no artificial distinction between “middleware” and “network,” but rather a well-organized hierarchy of abstractions and implementations, problems and solutions.

6.1 Overlay organization

What is the range of functions and cost/performance trade-offs that overlays can provide? Clearly evolution should move in the direction of offering more and better choices to stakeholders and application developers.

How are overlays arranged in a use hierarchy? Presumably there will be overlays with more application services at the top, and overlays with more clever resource management at the bottom.

In an organization of overlays, how many overlays should there be, and what are their scopes? These would seem to be the most important questions for understanding and achieving scalability.

Within an overlay, can the interests of different stakeholders be represented and composed in an orderly manner? Section 6.3 discusses a particular example of such composition.

As with any mechanism for modularity, layering imposes some overhead in terms of extra processes within an operating system, extra message headers, *etc.* How serious is this overhead? Are there techniques for minimizing it when necessary, and can they be used without destroying valuable boundaries?

An Internet of new overlays would require a tremendous amount of software. To what extent can overlay code be parameterized? How can an overlay be composed from smaller re-usable pieces?

Many of these questions are related to large bodies of research in programming languages and software engineering, far too numerous to mention here. New networking research should be drawing on their successful ideas.

6.2 Overlay interaction

Vertically adjacent overlays should interact only through their interfaces. Is the interface template in Section 5.1 enough, or is more inter-overlay communication required? When are vertically adjacent overlays bound together? Can it be delayed until runtime?

What about the possibility of explicit interaction between overlays that are related vertically but not directly adjacent? For example, could a topmost overlay choose the bottom overlay to be used by a middle overlay between them? Could there or should there be any explicit interface between top and bottom overlays?

How does interaction across the interfaces of the use hierarchy represent and compose the interests of different stakeholders? One issue is already handled explicitly: an overlay can account for and charge for resources used by an upper overlay.

Configuration is part of the interface between layers. In the Internet today, configuration at all levels is a huge headache and major source of errors. Can we design overlays to ease the task of configuration?

At a deeper semantic level, what is the specification of an overlay (the service it offers) and what are its assumptions about the overlays it uses? How can formal reasoning be used to verify that an overlay's assumptions and implementation satisfy its specification, and that the specification of an overlay satisfies the assumptions of an overlay that uses it? Are there principles governing the properties of overlays and how they compose in the use hierarchy?

6.3 Communication primitives

Message transmission (UDP), reliable FIFO connections (TCP), and multicast are familiar Internet primitives. Middleware commonly implements remote method invocation and transactions.

It seems that there might be other communication primitives that could be provided as "sessions" in the overlay pattern. Any of the following possibilities, however, might turn out to be minor variants of other primitives, or so intertwined with an application that they cannot be separated as an application-independent service.

An *abstract name* is a name that does not permanently denote a specific overlay process. Several possible primitives are related to abstract names. For one example, a multicast name denotes a set of processes intended to receive all transmissions in some class. Other examples include publish/subscribe event-based communication, message-oriented anycast, and connection-oriented anycast. (In connection-oriented anycast, a connection request can go to any member of an abstractly named anycast group, but once the connection is formed to a particular process it is point-to-point.) The semantics of abstract names has been explored in [29].

In many high-performance applications today, processes migrate from one machine to another. This is similar to mobility but not identical, as a process changes the machine to which it is attached.[8]

The most interesting possibility in this area is support for applications that are not monolithic, but rather are compositions of several applications. The composed applications can be developed independently, and can represent the

[8] Day views migration as a form of multicast. In his view, a post-migration process is related to, but different from, the pre-migration process.

interests of different stakeholders. For example, Figure 3 can be seen as a composition of four applications (three proxies and a Web server). We know that these applications can be developed independently. As explained in Section 3.4, they represent the interests of four different stakeholders.

In this example the support needed is a way to assemble the servers into the path of the HTTP request that is flexible, convenient, and prioritizes the stakeholder interests in an orderly way. It is not a matter of naming—at least not directly—because the only name given by the user is the name of the Web server at the end of the chain.

This problem is actually a special case of the problem solved by Distributed Feature Composition (DFC) [30, 31]. DFC is an architecture for telecommunication services composed of independent applications. In DFC the analog to Figure 3 is a dynamically assembled, multi-party, multi-channel graph that can change while connected to a particular endpoint, even splitting into two graphs or joining with another graph. The applications appear in the graph because endpoint names and abstract names *subscribe* to them.

Not surprisingly, routing to assemble a DFC graph uses subscriber data and an extra level of indirection, so its overhead is too high for many Internet applications. The challenge here is to find weaker versions that scale better, so that there are adequate solutions for each situation.

6.4 Security

As with the other research topics, the foundation of security is the association between network structures and the stakeholders who control them, as well as the trust relationships among stakeholders. Security mechanisms can relocate trust, but not create it.

Security is often partitioned into three problems: availability, *integrity, i.e.,* controlling changes to the system state, and *confidentiality, i.e.,* keeping information secret.

Two kinds of security are obviously inherent in the overlay pattern. On behalf of its users, an overlay must enforce their policies on which user processes have the authority to change configured state, and which user processes have permission to communicate with which other user processes. On its own behalf, it must enforce its own policies on which applications, overlays, and overlay members can use it, and how they are held accountable for their expenditures. These mechanisms would seem to generalize well to many aspects of availability and integrity.

Confidentiality is most often enforced by encryption. The most natural place for encrypting messages would seem to be within the overlay that is sending and receiving them, rather than in the underlay that is transmitting them.

Theoretical considerations aside, the main point is that buggy software makes bad security. With respect to security, the purpose of network architecture is to make the operation of network software correct, comprehensible, and modular, so that effective security mechanisms can be designed for it.

7 Conclusions

Because of opportunities created by Internet evolution, there are exciting and important research challenges concerning how networks should be designed to meet the growing needs of global networked applications. The observations in Section 3.5 show that software engineers should be participating in this research, because the skills, perspective, and values of our community are much needed.

"Internet architecture," whether in the classic form of Section 2 or the reality of Section 3, has always served as a boundary between the disciplines of networking (below) and distributed software systems (above) [24]. To participate in Internet research, we must break down this boundary. We must gain more familiarity with the design issues of layers below middleware. We must also work harder to bring our perspective to the networking community, whether in industry, academia, or the Internet Engineering Task Force (IETF).

Finally, software engineers should collaborate with researchers in the networking community to test their ideas and find the most efficient ways to implement them. One of the great strengths of the networking community is building efficient, large-scale implementations of distributed algorithms. Another great strength of the networking community is experimental evaluation of ideas. And only collaborative research will foster the common terminology and shared values needed for both communities to affect technology in practice.

Acknowledgment

Most of my understanding of Internet evolution is due to the generous and expert guidance of Jennifer Rexford.

References

1. M. Shaw, "Continuing prospects for an engineering discipline of software," *IEEE Software*, vol. 26, pp. 64–67, November 2009.
2. W. Emmerich, M. Aoyama, and J. Sventek, "The impact of research on the development of middleware technology," *ACM Transactions on Software Engineering and Methodology*, vol. 17, p. Article 19, August 2008.
3. E. W. Dijkstra, "The structure of the 'THE' multiprogramming system," *Communications of the ACM*, vol. 11, pp. 341–346, May 1968.
4. J. Saltzer, D. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, pp. 277–288, November 1984.
5. D. D. Clark, "The design philosophy of the DARPA Internet protocols," in *Proceedings of SIGCOMM*, ACM, August 1988.
6. N. Feamster, L. Gao, and J. Rexford, "How to lease the Internet in your spare time," *Computer Communications Review*, vol. 37, pp. 61–64, January 2007.
7. B. Carpenter and S. Brim, "Middleboxes: Taxonomy and issues." IETF Network Working Group Request for Comments 3234, 2002.
8. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford, "On static reachability analysis of IP networks," in *Proceedings of IEEE Infocom*, IEEE, March 2005.

9. M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker, "Middleboxes no longer considered harmful," in *Proceedings of the Sixth Usenix Symposium on Operating Systems Design and Implementation*, ACM, December 2004.
10. S. Guha and P. Francis, "An end-middle-end approach to connection establishment," in *Proceedings of SIGCOMM*, pp. 193–204, ACM, August 2007.
11. M. S. Blumenthal and D. D. Clark, "Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world," *ACM Transactions on Internet Technology*, vol. 1, pp. 70–109, August 2001.
12. P. Zave, "Requirements for routing in the application layer," in *Proceedings of the Ninth IEEE International Conference on Coordination Models and Languages*, pp. 19–36, Springer-Verlag LNCS 4467, 2007.
13. C. E. Perkins, "Mobile IP," *IEEE Communications*, May 1997.
14. J. Mysore and V. Bharghavan, "A new multicasting-based architecture for Internet host mobility," in *Proceedings of the Third Annual ACM/IEEE International conference on Mobile Computing and Networking*, ACM, 1997.
15. D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden, "Tussle in cyberspace: Defining tomorrow's Internet," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 462–475, June 2005.
16. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol." IETF Network Working Group Request for Comments 3261, 2002.
17. J. Rosenberg, "A hitchhiker's guide to the Session Initiation Protocol (SIP)." IETF Network Working Group Request for Comments 5411, 2009.
18. M. Afergan, J. Wein, and A. LaMeyer, "Experience with some principles for building an Internet-scale reliable system," in *Second Workshop on Real, Large Distributed Systems (WORLDS '05)*, USENIX Association, 2005.
19. S. A. Baset and H. G. Schulzrinne, "An analysis of the Skype peer-to-peer Internet telephony protocol," in *Proceedings of the 25th Conference on Computer Communications (INFOCOM)*, IEEE, April 2006.
20. P. Zave and E. Cheung, "Compositional control of IP media," *IEEE Transactions on Software Engineering*, vol. 35, January/February 2009.
21. A. Feldmann, "Internet clean-slate design: What and why?," *Computer Communications Review*, vol. 37, pp. 59–64, July 2007.
22. S. Ratnasamy, S. Shenker, and S. McCanne, "Towards an evolvable Internet architecture," in *Proceedings of SIGCOMM*, ACM, August 2005.
23. T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *IEEE Computer*, vol. 38, pp. 34–41, April 2005.
24. T. Roscoe, "The end of Internet architecture," in *Proceedings of the Fifth Workshop on Hot Topics in Networks*, 2006.
25. J. Day, *Patterns in Network Architecture: A Return to Fundamentals.* Prentice Hall, 2008.
26. I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proceedings of SIGCOMM*, ACM, August 2002.
27. Y. Mao, B. T. Loo, Z. Ives, and J. M. Smith, "MOSAIC: Unified declarative platform for dynamic overlay composition," in *Proceedings of the Fourth Conference on Future Networking Technologies*, ACM SIGCOMM, 2008.
28. D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle, "OCALA: An architecture for supporting legacy applications over overlays," in *Proceedings of the Third USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '06)*, 2006.

29. P. Zave, “Address translation in telecommunication features,” *ACM Transactions on Software Engineering and Methodology*, vol. 13, pp. 1–36, January 2004.
30. M. Jackson and P. Zave, “Distributed Feature Composition: A virtual architecture for telecommunications services,” *IEEE Transactions on Software Engineering*, vol. 24, pp. 831–847, October 1998.
31. P. Zave, “Modularity in Distributed Feature Composition,” in *Software Requirements and Design: The Work of Michael Jackson* (B. Nuseibeh and P. Zave, eds.), Good Friends Publishing, 2010.