The compositional architecture of the Internet

Pamela Zave
AT&T Labs—Research
Bedminster, New Jersey
pamela@research.att.com

Jennifer Rexford
Princeton University
Princeton, New Jersey
jrex@cs.princeton.edu

ABSTRACT

Contrary to the "classic" Internet architecture familiar to most people, today's Internet is a composition of a wide variety of networks. The IP protocol suite offers a general-purpose network design with a widely available implementation; as such, it is re-used to design and implement networks with many different purposes. Compositional architecture explains how, despite the fact that IP has not changed significantly since 1993, the Internet has evolved to meet many new requirements and challenges since then. In this paper we argue that understanding and modeling network composition is the key to continued evolution of the Internet, and to meeting society's demands for Internet services that can be verified to meet their requirements, particularly for security and reliability. We first define networks, requirements on network services, and bottom-up reasoning that a network meets its service requirements. Next we define the composition operators of layering and bridging. Rich examples from everyday networking not only illustrate the operators, but also show reasoning across a composition hierarchy. In conclusion, we show how a shift toward more explicit use of composition would greatly enhance our ability to make the Internet better.

ACM Reference format:
Pamela Zave and Jennifer Rexford. 2017. The compositional architecture of the Internet. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 9 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In 1993, the explosive growth of the World Wide Web began. The architecture of the Internet was commonly described as having four layers above the physical links, each providing a distinct function: a *link* layer providing best-effort local packet delivery, a *network* layer providing best-effort global packet delivery (and characterized by the Internet Protocol or IP), a *transport* layer providing communication services such as reliable byte streams (TCP) and message service (UDP), and an *application* layer. 1993 was also the year of the last major change to this "classic" Internet architecture, for reasons eloquently explained by Handley [10].

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

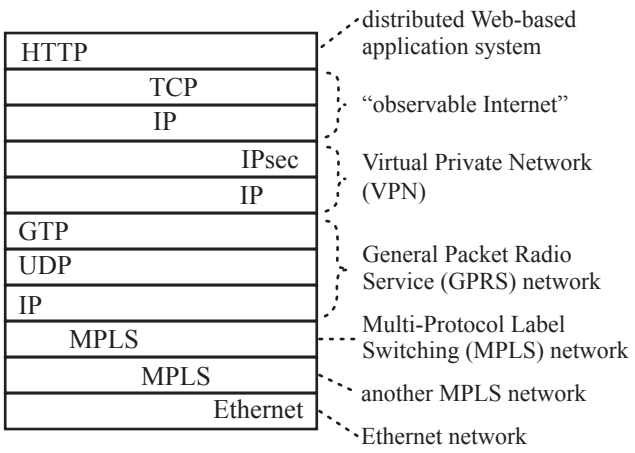

Figure 1: Headers of a typical packet in the AT&T backbone network. Headers lower in the diagram are outermost in the actual packet.

A lot has happened in the world since 1993. The overwhelming success of the Internet has created many new uses and challenges that were not anticipated by its original architecture:

- Today, most networked devices are mobile.
- There has been an explosion of security threats.
- Most of the world's telecommunication infrastructure and entertainment distribution has moved to the Internet.
- Cloud computing was invented to help enterprises manage the massive computing resources they now need.
- The IPv4 32-bit address space has been exhausted, and IPv6 has not yet taken over the bulk of Internet traffic.
- In a deregulated, competitive world, network providers control costs by allocating resources dynamically, rather than provisioning networks with static resources for peak loads.

Here is a conundrum. The Internet is meeting these new challenges to some extent, yet neither the IP protocol suite nor the way that experts describe the Internet has changed significantly since 1993. How can this be?

In this paper we will show that today's Internet is better described as a *composition* of many networks. These networks vary wildly in their purposes, their geographical spans, their levels of abstraction, and their internal designs. At the same time, each network is a microcosm with the potential to contain all of a network's basic working parts and functions. All networks have fundamental similarities, and in particular similar interfaces that allow them to be composed. The IP protocol suite retains its primacy simply as

software that can be found on most networked devices, and can thus be re-used in the design and implementation of many networks.

Evidence of composition is easy to find. Figure 1 shows the headers of a typical packet in the AT&T backbone [13]. The headers tell us that this packet is being sent by a distributed Web-based application system. It is being transmitted by a layered composition of six networks, the topmost of which is called the "observable Internet" because it is the IP network that the application interacts with. Below it, a VPN provides secure service over public links. A GPRS network implements cellular data service. Two layered MPLS networks allocate backbone resources by sending the packet over selected paths at different levels of abstraction. An Ethernet network transmits the packet within a local area. The IP protocols and implementation software are used to build three of these networks. In the classic Internet architecture, nothing between the "observable Internet" and the Ethernet would exist.

The onslaught of new challenges is not over. The consequences of cyberattacks are worsening as the world's infrastructure becomes more heavily networked. Real-time Internet applications are increasing, including safety-critical ones such as medical applications and self-driving cars. Based on current projections, the Internet of Things will increase the number of networked devices by a factor of 25, while mobile access must be provided at 1/25th the current cost of cellular service.

In this paper we will argue for the critical importance of understanding composition as a foundation of network architecture. On an intellectual level, composition is indisputably present, and ignoring it will frustrate attempts to describe or generalize network architectures with any precision or utility. Moreover, there are two extremely important practical issues for which understanding of composition is essential.

Interoperation and evolution: In response to current problems and forecasts of worse ones, many researchers have investigated "future Internet architectures" that aim to eliminate the weaknesses of the current Internet with a "clean slate" approach based on new technology, particularly for security [7, 8]. These thought experiments encourage innovation, but there will be no opportunity to shut down the current Internet and start up a new one. New solutions to problems must interoperate with the current Internet; eventually successful designs will be used more widely, and the Internet will evolve. In this paper we will show that the key technology for interoperation and evolution is composition of networks. By understanding and formalizing the interfaces between composed networks, we can encourage the design of networks that are easily composed. This will expand the design space for effective and efficient problem solutions that can actually be deployed. It will also help to bridge the artificial and unproductive divide between networking and distributed systems [12].

Requirements on communication services: Practically every issue of *CACM* contains a warning about the risks of rapidly increasing automation, because software systems are too complex for people to understand or control, and too complex to make reliable. Networks are a central part of the growth of automation, and there will be increasing pressure to define requirements on communication services and to verify that they are satisfied. As we will show in this paper, the formal framework for composition is exactly the formal framework needed to specify and verify network requirements. It supports both top-down and bottom-up compositional reasoning. Without it, the nascent research area of network verification can help network providers manage their networks, but cannot give the public trustworthy Internet services.

A note on terminology: The terminology in common use for networking is based on the classic Internet architecture. It is a barrier to understanding because it is inherently ambiguous in today's Internet. Our ambition is to define terms unambiguously and use them precisely, which first requires choosing the terms we will use. We have chosen common and intuitive terms, which although heavily overloaded in practice, should be interpreted here as meaning only and exactly what they are defined as.

2 WHAT IS A NETWORK?

2.1 A top-down view

The users of *networks* are distributed application systems—computer systems with operational modules spread across different physical machines. The modules of a distributed system need a network to communicate.

Each module using a network must have running on its machine a *member* of that network. The network member is a software (and possibly hardware) module that participates in the network. Application modules and network members on the same machine communicate with each other through the fast, reliable operating system of the machine—in contrast to the slower, faultier communication between machines. Both app modules and network members have *names*, in the namespaces of the app system and network, respectively.

The network offers to its users one or more communication *services*. An instance or usage of a service is a *session*. A *packet* is a transmissible unit of data. In each session, a group of related packets from an app *sender* is delivered to an app *receiver*. So the basic user interface to the network is that the sender has action *send (packet, session)* to send a packet in a session, and the network has action *deliver (packet, session)* to deliver a packet to the receiver.

A distributed system imposes requirements on the network services it is using. These properties or constraints are defined in terms of the using distributed system, not the network (later we will see why). There are four common categories of requirement:

- *Reachability* requirements specify which receivers a member can send packets to.
- *Performance* requirements specify quantities such as maximum latency, minimum bandwidth, and maximum packet loss rate.
- *Behavioral* requirements are more service-specific. For example, TCP service requires a pair of sessions, where the sender of one is the receiver of the other. It guarantees reliable, FIFO delivery of a byte stream in either direction.
- *Security* requirements are diverse. For example, access control is the negation of reachability. Denial-of-service protection is a kind of access control in which the bad senders are identified by volume, packet contents, and other traffic characteristics rather than by name. If a service includes authentication, then packets must be sent by the sender named when the session is set up. If the service has privacy

and data integrity, then no packets are read or modified by unauthorized parties while in the network.

2.2 A bottom-up view

The parts of a network are *members* and *links*. As defined above, members are named modules on participating machines. The state of the network is distributed across its members. A link is a communication channel that accepts packets from a sender (member) and delivers them to a receiver (member).[1] Many links are "best-effort" channels, which means we can only say formally that they deliver a packet within some latency t, with some probability r.

A network has two principal activities or functions, which are its *forwarding protocol* and its *session protocols*. The following descriptions of them are intended to apply to all networks. They do not cover every aspect of networking, but they do cover enough to explain how networks compose.

A fully-connected network has a link from each member to every other member. Most networks are not fully connected, but packets can reach their destinations over *paths* of multiple links. In these networks, when a member receives a packet that is not destined for it, the member forwards the packet toward its ultimate destination. We use the word *forwarder* for a member whose primary or sole purpose is to forward, rather than words like "router" or "switch" with many specialized connotations.

The network's *forwarding protocol* is a set of conventions that govern headers on packets, forwarding state in members, and how members use the forwarding state to handle received packets. More specifically, each member has a local mapping from *headerPattern* and *inLink* to *outLink*, where *headerPattern* matches some subset of packet headers, and *inLink* and *outLink* are local identifiers for the links of that member. The mapping tells the member that on receiving a packet on incoming link *inLink* whose header matches *headerPattern*, it should forward the packet onto outgoing link *outLink*. To select an *outLink* for packets originating at the member, a distinguished value *self* can be used as the "incoming link." The mapping can also tell the member, explicitly or implicitly, to drop the packet. Most commonly the mapping is realized by entries in a table, but it can also be realized with some computation, which allows the use of more information from the header.

A *session protocol* is a set of conventions governing the packet format, packet sequence, member state, and member actions used to implement a service according to its requirements. A network has a session protocol for every service that it offers.

Each packet in a session has a header, which must contain at least the *source, destination,* and *session identifier*.[2] The *source* and *destination* are the names of the members serving as *endpoints* of the session, as shown in Figure 2. The *session identifier* is chosen so that the header uniquely identifies the session. The header format of a session protocol is a specialization of its network's forwarding format, so a header format must conform to both.

Session state is stored in both endpoints, and possibly other members (called *middleboxes*) in the path of the session packets. When a sender executes *send (packet, sessIdent)*, the source member

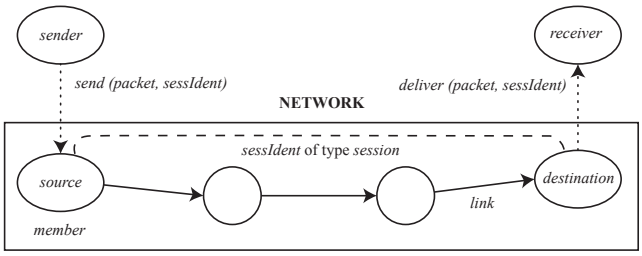

Figure 2: Packets of a session are forwarded from source to destination endpoints. The unique identifier *sessIdent* indicates which packets belong to this group.

gets the packet, disassembles it into smaller packets if necessary for the network, encapsulates each packet in the session header, and sends them out on its links. When a destination receives session packets from its links, it decapsulates them by removing the session headers, assembles them into larger packets if required for the service, and executes *deliver (packet, sessIdent)* to transfer the packet to the receiver. In addition, endpoint members and middleboxes can perform other actions as defined by the session protocol.

A network has a single administrative authority, which is responsible for meeting the network's requirements. The administrative authority provides resources, particularly links and *controlled* members. The network can also have *free* members that are not controlled by the authority; typically these are the endpoints of sessions, located on the machines of network users.

The distinction between controlled and free members makes a crude start on a trust model that can be used to reason rigorously about security. For example, for protecting the resources of the network, only controlled members can be trusted. For protecting a session, the controlled members and session endpoints can be trusted, but no others. On a global scale, it will be necessary for the administrative authorities of networks to distinguish which other networks they can trust [2].

Packet processing in a network includes its forwarding protocol, its session protocols, and the creation and destruction of on-demand sessions and links (see §5). Most networks also have control functions, provided by their administrative authorities to configure and manage the states of controlled members. Typically controlled members provide status information to the control function, and the control function adjusts state such as forwarding state according to current conditions. It is important to note that separation and composition of networks are, so far, defined on their packet processing only. We do not yet know to what extent control functions are modular like packet processing is, and to what extent they must be intertwined for effective optimization.

2.3 Reasoning between top and bottom

Reasoning about a network should be self-contained. The goal is to combine knowledge of the network design and resources with knowledge or assumptions about its link properties, and to reason upward that the network satisfies its service requirements. To achieve this goal, however, it is necessary to have two kinds of information from the top.

[1] Although the theory allows one-to-many sessions and links, they are omitted for simplicity.

[2] It may also need an identifier of the session protocol, if there is more than one.

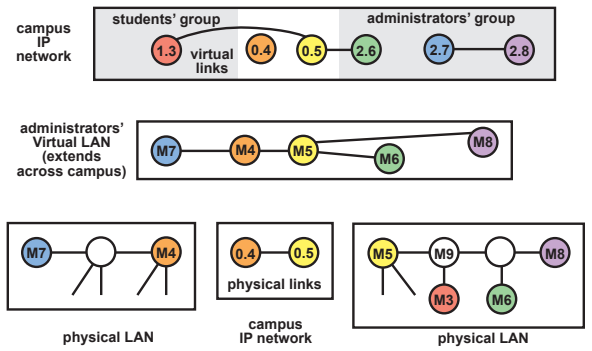

Figure 3: The architecture of a campus network. In this picture, all lines between members are bidirectional pairs of links. All network members on the same machine are the same color, and end with the same digit.

The first kind of top-down information is critical for packet processing as well as for reasoning. For each distributed system that uses the network, there should be a *directory* mapping names of app modules to the names of the network members that they use to communicate. Consider, for example, how Web services use the Internet. A Web request is sent from a sender (client) C to a receiver (server) S, which is an app module having a domain name. For an IP network to implement this communication, it must discover the network name (IP address) of S, which will be the destination of the TCP session carrying the request. DNS is the directory providing this information.[3]

Secondly, there should be knowledge or assumptions about the *load* of sent packets, not just from one distributed system, but from all of those that use the network. A typical load characterization is a traffic matrix, with an expected packet rate from each member as a source to each member as a destination. Most of the theory of networking so far concerns how to reason about the relationships among network topology, link properties, load characterization, and observed path performance, so reasoning about performance properties has a good foundation.

The remainder of this paper defines compositional architecture and uses examples to show the value of thinking compositionally. Many everyday uses of networking are complex, subtle, and confusing, even to most practitioners, yet the model enables us to explain them precisely. We also show how bottom-up reasoning can be used to satisfy service requirements of all four types introduced in §2.1.

3 NETWORK COMPOSITION BY LAYERING

Layering is a composition operator in which some links of a network are implemented by sessions of another network. In other words, the *overlay* network is one of the using distributed systems—in this case *not* an app system—of the *underlay* network. In Figure 1, every network except the "observable Internet" is an underlay in this sense, and every network except the Ethernet is an overlay. Layering is conceptually very simple, because all the groundwork has already been laid. The overlay network is a using distributed system, which is where requirements are defined, so the properties of its links are the requirements on the underlay service. Reasoning that the underlay network satisfies its requirements supports reasoning that the overlay network satisfies its requirements.

For a link of one network to be implemented by an underlay network, both endpoints of the link must be *attached to* or *located at* members of the underlay. The attached members are on the same machine, and communicate through the operating system of the machine. The directory of the underlay network maps names of attached members to names of their attachments. In Figure 3, members 2.7 and 2.8 of the topmost network are attached to members M7 and M8, respectively, of the middle network. If a network is used by several others with overlapping name spaces, the directory entries will also need network names for disambiguation.

As a second example of what layering can do, Figure 3 shows a common architecture for a campus network [15]. At the top level, there is an IP network with private IP addresses in the prefix range 192.168/16 (henceforth we will show only suffixes of this prefix). For security and management purposes, the user machines are partitioned into groups for administrators, departments, students, etc. (only two are shown). These groups are allocated different ranges of suffix space. The IP network also includes routers such as 0.4 and 0.5. In the figure we show the virtual IP links needed for communication paths between 1.3 and 2.6, and 2.7 and 2.8.

At the middle level of the figure, there is a Virtual Local Area Network (VLAN) for each group (only one VLAN is shown). Each user machine has a member in one VLAN, while the IP routers can have members in multiple VLANs. In VLANs the names of members are the MAC addresses of their machines, indicated mnemonically to match IP addresses.

The principle of this architecture is that all VLANs are completely isolated, so machines can communicate across group boundaries only through the IP network. There are several reasons for this principle, but the relevant one is that groups are defined for security, and its IP address is the only way to tell which group a user machine belongs to. Therefore security rules must be enforced by the IP network, and all IP paths across group boundaries must go through an IP router.

In a VLAN, the IP routers are fully connected by links, and each user machine is linked to its nearest (see below) IP router. So the links between 2.7 and 2.8 are implemented by VLAN sessions between M7 and M8. The links between 0.5 and 2.6 are also implemented by sessions in the administrators' VLAN, while the links between 1.3 and 0.5 are implemented by sessions in the students' VLAN.

At the bottom level of the figure, each user machine is a member of a physical LAN for the area of campus in which it is located. Each LAN contains switches that are not members of higher-level networks. A member of any VLAN can be located in any LAN. A LAN does not need a directory, because its members and their corresponding overlay (VLAN) members have the same names.

Each VLAN link between a user machine and its nearest IP router is implemented by a LAN. Packets in the LAN require two Ethernet headers (as well as an IP header). To see why, consider the packets being transmitted on the virtual IP link from 2.7 to 2.8.

[3]Because Web services never inititate sessions to clients, client Web modules do not need explicit names.

Their inner Ethernet header, which belongs to the VLAN, labels them as traveling in a session with source M7 and destination M8. Their first outer Ethernet header, which belongs to the LAN on the left, labels them as traveling in a session with source M7 and destination M4. Later the same packets, when traveling through the LAN on the right, will have outer Ethernet headers labeling them as traveling in a session with source M5 and destination M8. In both LANs the outer Ethernet headers will have VLAN tags, which identify the overlay the packets belong to (see §5).

VLAN links between IP routers are implemented by cross-campus high-speed links. These links belong only to the campus IP network; in the figure the IP network appears twice, at the top with its relevant virtual links shown, and at the bottom with its relevant physical links shown. VLAN packets with source M7 and destination M8 will be encapsulated in an IP header with source 0.4 and destination 0.5 to traverse the physical link in the IP network (these IP-in-Ethernet-in-IP packets are in the "VXLAN" format). Note that these packets will not trigger security functions in 0.5 because, being destined for 0.5, they are not *entering* a group. We can see this is correct, because in fact the entire communication path lies within the admin group and VLAN.

This example shows that a directed graph of the layering relation on networks will sometimes have cycles, particularly where an IP network is layered on itself (here the IP network uses itself through VLAN intermediaries). For a more precise and useful relation, we can define layering on links in networks. In this relation, some virtual links in the IP network are layered on physical links in the IP network, but no IP link is layered on itself, and the relation is hierarchical (i.e., acyclic).

Now we consider requirements on this architecture. We might wish to understand the consequences for reachability of the failure of one pair of physical links, between M5 and M9 in the LAN on the right. Based on analysis of the lowest level of Figure 3, we would conclude that the user machines with MAC addresses M3 and M6 could still reach each other. This conclusion would be incorrect. The outage would break the links between M5 and M6 in the admin VLAN, and between M5 and M3 in the student VLAN. This would, in turn, break the links between 1.3, 0.5 and 0.5, 2.6 in the IP network.

In general, the campus network will have access-control rules for packets from/to any pair of groups. These rules must be installed in IP routers. We might wish to install the minimal number of rules in each router, yet verify for security that every packet from one group to another goes through a router with the access-control rules for that from/to pair. Reasoning from the bottom up, virtual links in the VLANs are abstractions of packet transmission in the LANs, so it is sufficient to reason about VLANs. Each user machine is a member of one VLAN, where it has links to a single IP router. So security for packets from group F to group T can be verified if its access-control rules are installed in all routers R such that the MAC address of R is in VLAN F.

In response to the reachability problem above, network managers might install a backup link between M9 and another IP router 0.11 with MAC address M11. In this case M11 would also appear in the admin VLAN, and the rule above would remind the managers to install access-control rules for the admin group in M11.

4 NETWORK COMPOSITION BY BRIDGING

Bridging is a composition operator in which sessions or a service are implemented by a set of networks at the same level of abstraction. With bridging, the two endpoints of a session can be members of different networks.

There are several variations on bridging, depending on how much structure the bridged networks share. In the simplest case there are two networks with identical namespaces and protocols, such that there are some machines having a member that belongs to both networks. These *shared* or *bridging* members can forward packets from one network to the other. Provided that the names of all network members are unique across both networks,[4] and that members of both networks have access to the directory of the other, little changes except that the reach of both networks is extended. Because sessions span bridged networks, they can implement a service jointly.

The public Internet is a large set of IP networks composed by bridging. A network can be bridged with more than one network, and the extension of reachability provided by bridging can be transitive across chains of networks. In this case the main significance of composition is that each of the contributing networks can have a different administrative authority. When two networks with different authorities are bridged, most commonly a shared member is *controlled* in one network and *free* in the other.

In other cases, bridged networks are less similar. They may have different or overlapping name spaces. They may have different session protocols. In these cases bridging can still be effective, with the addition of *compound sessions*. A *compound session* is simply a session in which there is at least one middlebox acting as a *joinbox*. The joinbox serves as a destination for one simple session and a source for another simple session, and maintains state that associates the two simple sessions so that packets received by the joinbox as a destination are then sent by the joinbox as a source.

By far the most familiar compound sessions are those whose joinboxes are Network Address Translators (NATs) as in the lower level of Figure 4. NATs are joinboxes for sessions set up in in bidirectional pairs, usually for TCP service. The session pair in the figure is initiated by X in the private IP network on the left. Since the NAT is controlled by the private IP network and free in the public IP network on the right, it can also serve as a firewall protecting the private network from unsolicited packets. In this figure, the gray box represents the public Internet as one network, ignoring the fact that it is really a bridging of many networks.

It has been a long time since there has been enough room in the IPv4 32-bit name space to give every networked machine a unique name. Outright exhaustion of the name space has been delayed by the fact that most private networks re-use the same set of private IP addresses. The cost of this strategy is that private IP addresses are ambiguous except in their local context, and a machine with a private address cannot be reached from outside its local network except with a compound session.

This is the purpose of the compound sessions in Figure 4. The first session is initiated from the private address X, to public address **G**. When the NAT receives the session-initiating packet, it alters it before forwarding, thus making an outgoing session with

[4] A bridging member can have one name, or a different name in each network.

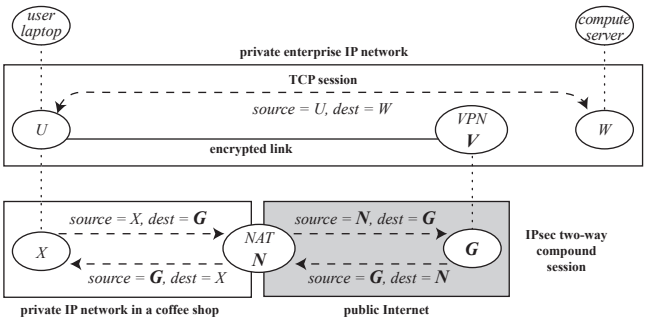

Figure 4: Two private networks communicate through the public Internet. Public addresses are in boldface, while private addresses are not. Vertical dotted lines show attachment of members to members of other networks.

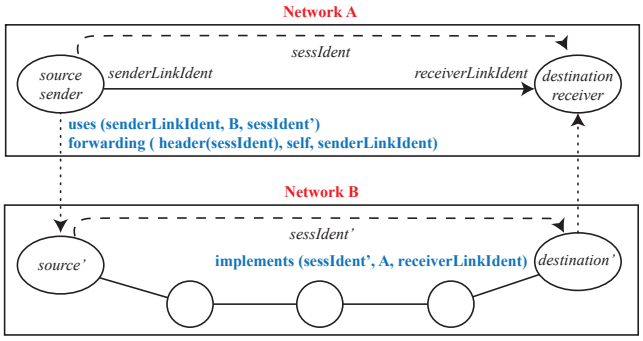

Figure 5: Network state (in blue, near the member where it is stored) for a link implemented in an underlay network.

its own public name **N** as the source. When **G** accepts this session and initiates a session in the reverse direction, it uses reachable **N** as the destination rather than unreachable *X*. Because the session-initiation packet in the reverse direction includes the session identifier from the forward direction, the NAT can match the reverse session-initiation packet with the compound session in the forward direction. With this information, it can continue the reverse compound session by initiating a simple session to *X*.

Figure 4 as a whole shows how an employee using a WiFi network in a coffee shop can access a compute server at his workplace—two private networks communicating through the public Internet. First, the user must connect to a Virtual Private Network (VPN) server in the private IP network of his enterprise, which is done by contacting gateway **G** from his temporary address *X* in the coffee shop. The VPN server authenticates the user's laptop and gives it temporary address *U* in the enterprise network. *U* and **V** then set up an encrypted link implemented by an IPsec session between *X* and **G** (other links are not shown). *U* and *W* are now both members of the private enterprise network, and can communicate freely.

Concerning reasoning about requirements, the fact that the user's laptop can reach the enterprise compute server is predicated on the facts that *X* and **G** can reach each other through compound sessions, and *U* can reach **V** through a dynamically created virtual link in an IP network.

There is an interesting problem here with a behavioral requirement. For the link between *U* and **V** to satisfy its security requirements, IPsec must be able to behave as designed. This is tricky because most NATs are only designed to admit TCP and UDP as IP session protocols. IPsec cannot masquerade as either of them because its session identifiers are quite different (the workaround is to encapsulate IPsec packets in UDP packets). If the Internet had been designed for composition, all IP packets would have a common format for the session identifier, because session identifiers play a key role in composition—and security—that is independent of particular session protocols.

5 EVOLUTION BY LAYERING

Layering is an obvious way to introduce networks with clean-slate designs (e.g., [1, 14, 18]) to the Internet, so that the Internet can evolve toward whichever proposals are most successful. Layering requires extra state in members to associate overlay links with underlay sessions. Figure 5 illustrates a general-purpose model for layering state.

Often virtual (overlay) links are transient or dynamic, just as sessions are. For example, in Figure 4, the encrypted link is created on user demand. Setting up a dynamic link to carry packets of the session *sessIdent* in Figure 5 proceeds as follows. The *source* of *sessIdent* is the same as the *sender* of the dynamic link. In the course of link setup, *sender* calls user-interface function *createSession (B, receiver)* to create a session in Network B. Its member in B, *source'*, executes this function by looking up *(B, receiver)* in its directory, finding its attachment or location *destination'*, and sending a session-initiation packet to it. *source'* also returns the new identifier *sessIdent'* to *sender*, which can now store the *uses* tuple so it knows where to send packets outgoing on the link. Note that the link has a unique local identifier at each endpoint. Note also that *sender* adds a tuple *forwarding (header(sessionIdent), self, senderLinkIdent)* to its forwarding rules.

The header of the session-initiation packet contains a fourth required field *overlay* containing *A* (compare to §2.2), because it is the network being served by the session. When *destination'* receives the session-initiation packet, it calls user-interface function *createLink (A)*, which *receiver* executes by setting up incoming link *receiverLinkIdent*. *receiver* returns the new identifier *receiverLinkIdent* to *destination'*, which stores it in the *implements* tuple so that it knows where to deliver packets received in session *sessionIdent'*. This is all the detail needed for composition, even though the session protocol may also entail acceptance, acknowledgment, etc.

Layering is heavily used in cloud computing. Often each cloud tenant has its own private IP network with its own copy of the private IP address space. Members of a tenant's network are virtual machines. At the physical level, the resources of a virtualizable machine are divided into slots, where each slot is a member of a network implemented in hardware and software. Slots are allocated dynamically to tenant networks, which means that each virtual machine is attached to or located at a slot.

Explicit layering state can be valuable for sharing the resources of the physical network among the tenant networks, as in [16]. In this proposal, a tenant network can dynamically request virtual

machines and virtual links, where the links can have bandwidth and latency requirements. A virtual link is implemented by a path through the physical network, on which each physical link must have capacity reserved for the virtual link.

Referring back to Figure 5, session *sessIdent'* is an explicit, named abstraction for the network path shown passing through three intermediate members. To reason (within the cloud's physical network) about the bandwidth of *sessIdent'*, we take the minimum over the path links of the bandwidths allocated to *sessIdent'*; this will tell us whether the session satisfies the minimum-bandwidth requirement of the virtual link. Reasoning about maximum-latency requirements is similar, except that we sum the latencies of the links on the path. Reasoning that the resources of the cloud have been allocated safely is a matter of determining which sessions share each link, and checking that their reserved capacities do not exceed the capacity of the link.

Layering has other uses in clouds besides resource allocation. In VL2 [9], layering is used to make virtual machines mobile (a virtual machine can be located anywhere in a data center) and to implement a customized routing scheme. In CloudNaaS [3], it is used to provide customers with services by means of middleboxes inserted in paths through the cloud. In SIMPL, where middleboxes are inserted in paths using forwarding rules, it is used to reduce the number of forwarding rules in central network elements [11].

6 INTEROPERATION BY BRIDGING

Mobility is a network service that preserves reachability to a member, and preserves the member's ongoing sessions, even though the member is moving and its connection to the network is changing. According to this strong definition, the only mobility service most people experience at present is cellular voice service.

Mobility is intrinsically difficult to implement in the Internet. Internet names (IP addresses) are allocated hierarchically, where the hierarchy is based on geographical, topological, and administrative proximity. The address hierarchy allows aggregated routing, in which an entry in a forwarding table suffices for a (large) range of IP addresses, and this is how the Internet works at global scale. Mobility creates individual, dynamic exceptions to aggregated routing, as a mobile device can connect to the Internet through a network where its own IP address does not belong in the network's address range. In cellular service, these dynamic exceptions are confined to networks owned by the cellular provider, but the implementation is still neither simple nor cheap.

Fortunately, there are other ways to implement mobility. With the help of the model of network composition, we were able to discover that all mobility schemes fit into exactly two patterns [17]. Cellular mobility is an instance of "dynamic routing" mobility, because individual rules in forwarding tables are updated as a mobile device moves. In the other pattern, mobile devices are members of a network in which they have *identifiers* (persistent names) by which they can always be reached. This network is layered on top of another network—in practice always an IP network—in which mobile devices have members with normal location-dependent names called *locators*. As a mobile device moves, its overlay member stays the same, but its underlay member changes its name in accordance with its new location. Because the far end of an underlay session often learns of new locations through the session protocol, we have called this pattern "session-location" mobility.

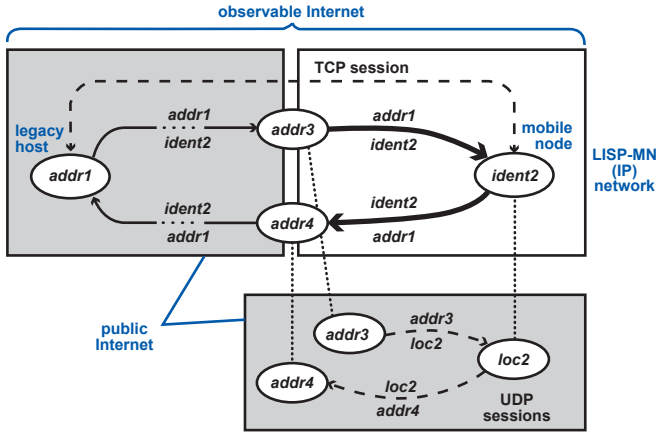

Figure 6: The interoperation of session-location mobility with the public Internet.

LISP Mobile Node [5, 6] is an implementation of session-location mobility that interoperates well with the public Internet, as shown in Figure 6. At the top level of this figure, the public Internet is bridged with a LISP-MN network, which is a specialized IP network. Because of the bridging, a legacy host with IP address *addr1* has been able to initiate a TCP session with a mobile node whose identifier is IP address *ident2*. As in §4, the public Internet is colored gray and depicted as if it were one network.

In the figure, each link (solid line), UDP session (dashed line), or path of links and forwarders (solid line broken with dots) is labeled above with the source address of the packets traveling on it, and labeled below with the destination address of these packets. These labels show that the two middleboxes (*addr3* and *addr4*) doing the bridging are not joinboxes and the TCP session is not a compound session, as the addresses are the same along each end-to-end path. These middleboxes belong to the LISP-MN network.

The LISP network owns a range of IP addresses, from which *ident2* is drawn. The LISP network also has a directory mapping identifiers of mobile nodes to their current locations. The middlebox at *addr3* is one of many belonging to the LISP-MN network that advertise the mobile range of IP addresses into the public Internet, which means that each packet destined for an address in this range will be forwarded to one of them. When such a middlebox receives its first packet for *ident2* (at least, first in a long time), it gets *ident2*'s location *loc2* from the directory, creates a dynamic link to *ident2*, and forwards the packet on it. Subsequent packets to *ident2* use the same link. The LISP network is layered on top of the public Internet, so that dynamic LISP links are implemented by public UDP sessions as shown in the figure. When a mobile node changes its location, it notifies all the middleboxes with which it is communicating through UDP, and also updates the directory.

Packets from *ident2* to *addr1* also use a virtual LISP link, because the mobile node might be located in a subnetwork where packets with source address *ident2* would be considered fraudulent. A locator, on the other hand, is always drawn from the local address

range. So the virtual link from *ident2* is implemented with UDP packets from *loc2* to *addr4*, which is a middlebox located where packets with source *ident2* can be transmitted.

The main requirement on LISP Mobile Node is that communication with legacy hosts should perform almost as well as normal Internet service. This requirement will be difficult to satisfy if the middleboxes are far away from both endpoints, so that packets between them must travel on elongated paths. The key is to scatter LISP-MN middleboxes geographically. Then packets going to a mobile node will be forwarded to whichever middlebox is closest to the sender. A mobile node can send its packets to a middlebox near to it. Thus in either direction packets go through a middlebox close to their sender, and end-to-end paths are not elongated.

§1 introduced the concept of the *observable Internet*, which can now be defined. Due to composition, the observable Internet can differ from user session to user session. For each user session, the *observable Internet* is the highest-level set of IP-based networks that, bridged together, transmit packets of the session from end to end. In Figure 6 the observable Internet is a bridging of the LISP-MN network and the public Internet.

7 A NEW INTERNET STORY

For many people concerned with the future of the Internet, there is a simple and somewhat depressing story. There is a single Internet (not counting adminstrative boundaries), which cannot be replaced because it is too much a part of civilization's infrastructure. Because it does not meet all current and projected requirements, we must seek to add a never-ending list of new features to it. Because its complexity is growing continually, we must work ever harder to find ways to secure and verify it.

In this paper we have presented evidence that the true situation is much different, and much more promising. In our story, the IP protocol suite offers a general-purpose network design with a widely available implementation; as such, it is re-used to design and implement many networks. The Internet as we know it is a vast collection of networks composed in a rich variety of ways by layering and bridging, including being composed with themselves.

The Internet of our new story evolves by means of new networks and new compositions. These are easy to add locally (campus networks, cloud computing) or at high levels of the composition hierarchy (mobility, network virtualization, distributed systems). They are slower to disseminate when both global and low in the composition hierarchy (IPv6).

The difference between these stories is profound. Imagine the consequences of acknowledging and studying the compositional architecture of the Internet. Further, imagine the consequences of a relatively minor shift to an Internet with the same architecture as today's, but with some explicit recognition, labeling, and use of common structures of composition such as those presented in this paper (these structures are all present today, but they are often implicit, hidden, or idiosyncratic). This shifted Internet would bring many benefits, including the following.

(1) On each networked machine there might be members of many networks, including multiple IP networks. Each could be designed and analyzed as a separate module (whether implemented separately or not).

Members of the public Internet may be the most complex because its links are sometimes layered on itself (often called "tunneling"). Even in this case, composition could replace ad hoc packet processing with uniform, analyzable operations such as those sketched out in §5. Such operations could be organized by factors such as which network is using the public Internet, and implemented by libraries in software or programmable hardware.

(2) It would be easier to evolve the Internet through new networks and new compositions, because the emphasis on composition would force networks to provide more robust and more general interfaces.

(3) Required properties could be stated in a meaningful and valid way. For example, reachability between the user laptop and compute server in Figure 4 has little to do with reachability in the public Internet.

(4) Verification would be performed hierarchically. Much of the reasoning would be performed on networks that are highly specialized to enforce the properties that are expected of them, and thus easy to verify. Explicit composition structures would tie the layers together so that the verified properties of an underlay could be assumed in an overlay.

(5) There are many consistency conditions that should hold between networks composed with each other. For example, a path in a virtual network (see §5) is available only if its links are available, and these are implemented by paths in an underlay. With more study, we could figure out how control functions could coordinate changes to composed networks so that consistency conditions are satisfied whenever it is possible to satisfy them.

(6) Networking is full of optimizations, and network composition would be no exception. Composition can be optimized by removing redundant data, replacing encapsulation (which increases header size) by rewriting (which does not), and many other transformations. The difference is that, in the Internet of our imaginings, pre-optimization information would be available for uniform and efficient reasoning. Also, optimizations could be proved safe in the context in which they are applied, and could thus be applied even more widely than they are now.

ACKNOWLEDGMENTS

John Day's book *Patterns in Network Architecture* [4] was our inspiration for viewing networks compositionally.

REFERENCES

[1] David G. Anderson, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. 2008. Accountable Internet Protocol (AIP). In *Proceedings of ACM SIGCOMM*.

[2] David Barrera, Laurent Chuat, Adrian Perrig, Raphael M. Reischuk, and Pawel Szalachowski. 2017. The SCION Internet architecture. *Commun. ACM* 60, 6 (June 2017), 56–65.

[3] Theophilus Benson, Aditya Akella, Anees Shaikh, and Sambit Sahu. 2011. CloudNaaS: A cloud networking platform for enterprise applications. In *2nd ACM Symposium on Cloud Computing*.

[4] John Day. 2008. *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall.

[5] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. 2013. The Locator/ID Separation Protocol (LISP). IETF Request for Comments 6830. (January 2013).

[6] D. Farinacci, D. Lewis, D. Meyer, and C. White. 2017. LISP Mobile Node. IETF Internet Draft draft-meyer-lisp-mn-17. (April 2017).

[7] Anja Feldmann. 2007. Internet clean-slate design: What and why? *ACM SIGCOMM Computer Communication Review* 37, 3 (July 2007), 59–64.

[8] Darleen Fisher. 2014. A look behind the Future Internet Architectures efforts. *ACM SIGCOMM Computer Communication Review* 44, 3 (July 2014), 46–49.
[9] Albert Greenberg, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, Dave Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of ACM SIGCOMM*.
[10] Mark Handley. 2006. Why the Internet only just works. *BT Technology Journal* 24, 3 (July 2006), 119–129.
[11] Zafar Ayyub Qazi, Cheng-Chun Tu, Louis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. 2013. SIMPLE-fying middlebox policy enforcement using SDN. In *ACM SIGCOMM*.
[12] Timothy Roscoe. 2006. The end of Internet architecture. In *Proceedings of the 5th Workshop on Hot Topics in Networks*.
[13] Oliver Spatscheck. 2013. Layers of success. *IEEE Internet Computing* 17, 1 (2013), 3–6.
[14] Arun Venkataramani, James F. Kurose, Dipankar Raychaudhuri, Kiran Nagaraja, Suman Banerjee, and Z. Morley Mao. 2014. MobilityFirst: A mobility-centric and trustworthy Internet architecture. *ACM SIGCOMM Computer Communication Review* 44, 3 (July 2014), 74–80.
[15] Minlan Yu, Jennifer Rexford, Xin Sun, Sanjay Rao, and Nick Feamster. 2011. A survey of virtual LAN usage in campus networks. *IEEE Communications* 49, 7 (July 2011), 98–103.
[16] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. 2008. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review* 38, 2 (April 2008), 17–29.
[17] Pamela Zave and Jennifer Rexford. 2013. The design space of network mobility. In *Recent Advances in Networking*, Olivier Bonaventure and Hamed Haddadi (Eds.). ACM SIGCOMM.
[18] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, and Van Jacobson. 2014. Named Data Networking. *ACM SIGCOMM Computer Communication Review* 44, 3 (July 2014), 66–73.