The Geomorphic View of Networking: A Network Model and Its Uses

ABSTRACT

The Internet is evolving away from its original architecture and toward the use of multiple, customized protocol stacks. A pluralistic architecture is best explained by the "geomorphic view" of networks, in which each layer is a microcosm of networking, and layers can be instantiated at many different levels, scopes, and purposes. Exploiting the commonalities identified by the geomorphic view, an abstract layer model can lead to architectural insights that help extend communication services, derive design principles, and generate network software.

Categories and Subject Descriptors
C.2.2 [Network Protocols]: Protocol Architecture

General Terms
Design, Performance, Security, Verification

Keywords
Internet, overlay, mobility

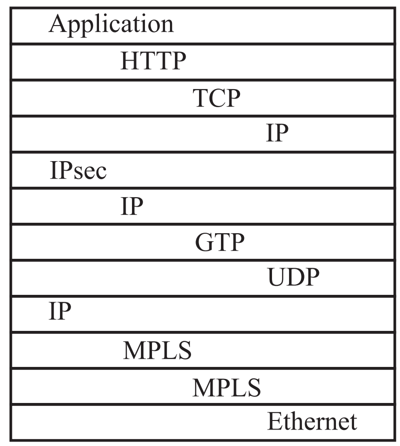

Figure 1: Headers of a typical packet in the AT&T backbone network.

1. INTRODUCTION

The "classic" Internet architecture is usually depicted with five layers: physical, link, network, transport, and application. The network layer is defined by the Internet Protocol (IP), while the transport layer is defined primarily by the transport protocols TCP and UDP. It is now widely agreed that this architecture fails to meet many of society's present and future requirements [2, 4, 5, 12].

The Internet is evolving as numerous stakeholders attempt to meet their goals for dependability, security, mobility, scalability, quality of service, and improved resource management. The result is a trend toward multiple, customized protocol stacks. For example, Figure 1 shows the headers of a typical packet in the AT&T backbone network. Counting one layer per header, this packet is handled by an architecture with twelve layers above the physical, instead of the classic four.

In this customized architecture, every layer above the UDP layer and below the application is middleware. GTP is a protocol that implements quality-of-service agreements, and IPsec provides security. HTTP is serving as a transport protocol, not because it is suitable for the application or even designed as a transport protocol, but because it is almost the only way to traverse NAT boxes and firewalls [11]. The presence of two MPLS layers, three IP layers, and two transport protocols from the IP suite is an indication of *ad hoc* code re-use, as protocols are instantiated at different positions in the stack to serve different purposes.

The good news to be found in Figure 1 is that middleware can be used to meet a wide variety of requirements, even for those stakeholders who cannot influence the lower layers of the architecture.

The bad news to be found in the figure is more extensive. It illustrates major problems and unmet needs in the following three areas.

Communication services: HTTP is a transaction-oriented client-to-server protocol. Although it supports Web services fairly well, there is a wide variety of potential Internet applications for which it is poorly suited. These include real-time, connection-oriented, mobile, peer-to-peer, server-to-client, and multi-party applications. When implemented on top of HTTP, these applications are difficult to develop, difficult to deploy and maintain, and very inefficient [16]. We need to

MW4NG Dec. 2012, Montreal, Canada

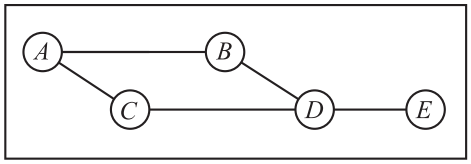

Figure 2: Members and links of a layer.

provide a broader range of communication services so that all applications can be developed easily and efficiently, while security policies appropriate to each application are reliably enforced.

Design principles: With so many layers, this customized architecture is very unlikely to be the most efficient way to satisfy the stakeholders' requirements. In fact, there is no way of predicting what its performance properties are, as an estimated 15 load-balancing algorithms apply to each packet, and each of the algorithms has been designed and analyzed mostly in isolation [14]. We need principles to help us design customized architectures that meet a variety of functional and nonfunctional requirements in an efficient, modular, and predictable way.

Software development: If the Internet evolves so that each application runs on its own customized Internet architecture, then a large amount of new network software will be required. We need to develop this software through code re-use and code generation, because conventional programming will be too slow, too expensive, and too prone to errors. Figure 1 shows code re-use, but does not meet other requirements such as efficiency and predictability.

This paper outlines a research agenda aimed at solving these problems. It is based on a "geomorphic view" of networks, in which each layer is a microcosm of networking—it has all of the basic ingredients of networking in some form. Layers are the architectural modules. In a network architecture there are many layer instances; they appear at different levels of the "uses" hierarchy, with different scopes, with different versions of the basic ingredients, and for different purposes. This view is described in more detail in Section 2.

The beauty of the geomorphic view is that any lesson we learn about layers in general can be used many times over. Section 3 explains how it might be possible to develop an abstract, formal model of a layer that would be helpful in solving the problems introduced above. We discuss both the abstractions and our progress on building such a model.

Finally, Section 4 proposes some ways in which the layer model could be exploited to solve the problems above. First, it could delineate design spaces and elucidate how decisions in one space depend on decisions in other spaces, leading to principles of network design. Second, it could broaden the set of mechanisms available to network designers, leading to a richer set of communication services without sacrificing other goals such as dependability, security, performance, and scalability. Third, it could serve as a framework in which a custom layer can be generated by selecting and integrating library components. We also summarize briefly our progress thus far.

2. THE GEOMORPHIC VIEW OF NETWORKS

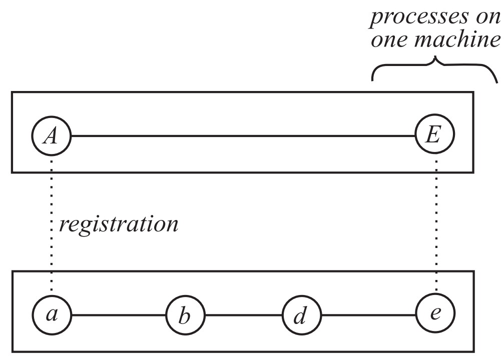

Figure 3: Implementation of a link in an overlay by a session in an underlay.

In the geomorphic view of networks, the architectural module is a *layer.* A layer has *members*, each of which has a unique, persistent *name.* For example, Figure 2 is a snapshot of a layer with five members, each having a capital letter as a name. In general a member is a process, *i.e.,* an independent, asynchronous locus of state and control. The actual behavior of a member may be no more complex, however, than a sequence of procedure calls.

The members of a layer communicate with each other through *links*, shown by lines in Figure 2. A link is a communication channel. In general, a layer does not have a link between each pair of members.

One of the two primary functions of a layer is to enable members to send messages to each other. To do this, a layer needs *routes* indicating how one member can reach another through links and intermediate members. For example, (*A, B, D, E*) is a route from *A* to *E.* It also needs a *forwarding protocol* that runs in all members. The forwarding protocol enables members to send and receive messages. In addition, when a member receives a message on an incoming link that is not destined for itself, its forwarding protocol uses the route information to decide on which outgoing link it will forward the message.

A *channel* is an instance of a communication service.[1] As mentioned above, a link is a channel. If a layer does not implement its links internally, then its links are implemented by other layers that this layer uses, placing the other layers lower in the *uses hierarchy.*

If an underlay (lower layer) is implementing a link for an overlay (higher layer), then the state of the channel must be stored as data in the underlay. In the underlay, the channel is known as a *session.* (There must be two names for channels, because a typical layer has both links and sessions.)

The second primary function of a layer is to implement enriched communication services on top of its bare message transmission. Typical enrichments for point-to-point services include FIFO delivery and quality-of-service guarantees. This function is carried out by a *session protocol.* A layer can implement sessions on behalf of its own members, as well as or instead of as a service to overlays.

For a link in an overlay to be implemented by a session in

[1] Although there is a wide variety of possible communication services, including broadcast, anycast, and publish/subscribe, in this brief summary we assume that channels are point-to-point.

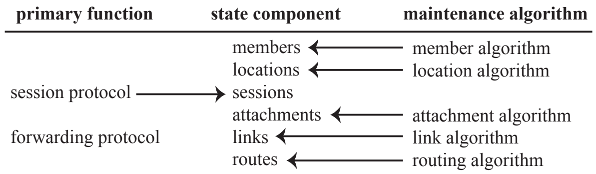

Figure 4: Major components of a layer. Arrows show which protocol or algorithm writes a state component.

an underlay, both endpoint *machines* must have members in both layers, as shown in Figure 3. A *machine* is delimited by an operating system that provides fast, reliable communication between processes on the machine. This fast, reliable operating-system communication is the foundation on which networked communication is built.[2]

A *registration* is a record that relates an overlay member to an underlay member on the same machine. Registrations must be stored as data in both layers. In the overlay they are called *attachments*, because they indicate how a member is attached to the network through a lower layer. In the underlay they are called *locations*, because they indicate that a member is the location of a process in a higher layer.

The session protocol creates and maintains *sessions* data in its layer, and uses *locations* data. For example, in Figure 3, A sent a request to a for a session with E. To create this session, a learned from its layer's *locations* that E is currently located at e. Messages sent from A to E through the link in the overlay travel through a, b, d, and e; the first and last steps uses operating-system communication, while the middle three steps use networked communication.

The six major components of the state of a layer are listed in Figure 4. All can be dynamic. We have seen that the session protocol creates and maintains *sessions*; the other five are created and maintained by their own maintenance algorithms.

This view of networking was inspired by the work of Day [3], although we have made many changes and additions in both content and presentation. It may seem familiar and obvious because both the classic Internet architecture [1] and the OSI reference model [7] also describe network architecture as a hierarchy of layers, but in fact there are several radical differences.

In the Internet and OSI architectures, each layer has a specialized function that is viewed as different from the function of the other layers. In both architectures, there is a fixed number of layers. In the geomorphic view, each layer is viewed as the same in containing all the basic functions of networking, and there can be as many layers as needed. Consequently, the network (IP) and transport (TCP/UDP) layers of the Internet architecture fit into one "Internet core" layer of the geomorphic view, where IP is the forwarding protocol and TCP and UDP are variants of the session protocol offering variants of Internet communication service.

Because layers instantiated at different levels have different purposes, their functions take different forms. For one example, we are most familiar with routing algorithms in the

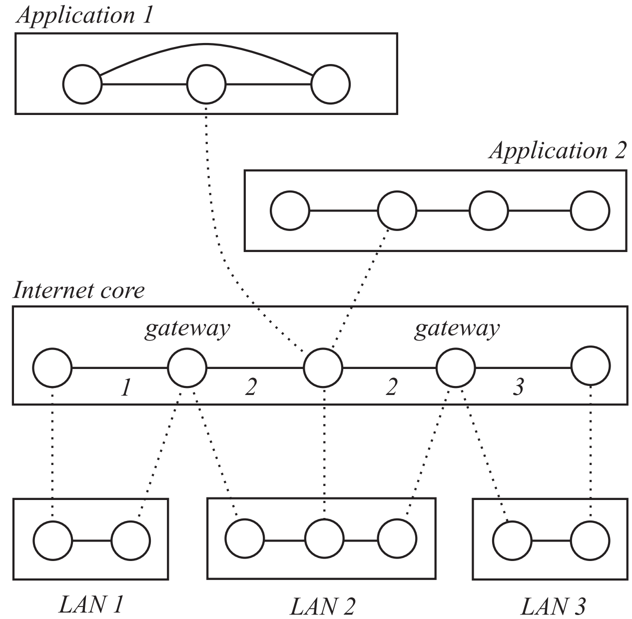

Figure 5: Geomorphic view of the classic Internet architecture. Internet links are labeled with the LAN that implements them.

Internet core, where their purpose is reachability. A higher-level middleware layer might offer security as part of its communication services. Implementing security might entail routing all messages to a particular destination through a particular filtering server, so that, in this layer, part of the purpose of routing is security. An application layer might have a link or potential link between any two members, implemented by communication services below, so that in this layer the routing algorithm is vestigial. For another example, low-level layers such as Ethernet LANs provide broadcast as a communication service. The services provided by the Internet core are point-to-point, while an application layer might implement its own form of broadcast.

The *scope* of a layer is its set of potential members. In the Internet and OSI architectures, each layer has global scope, and there is exactly one layer at each level of the hierarchy. In the geomorphic view, as shown in Figure 5, a layer can have a small scope, and there can be many layers at the same level of the hierarchy. Figure 5 shows the geomorphic view of the classic Internet architecture, with many LAN layers at the bottom level. In each LAN layer, the data structures, algorithms, and protocols are precisely those of the particular LAN technology being used. This is in sharp contrast to the idea of a generic, global "link layer," which cannot be made precise because it is a generalization of a large number of different technologies.

It is self-evident that fixed layer structures cannot explain today's customized architectures, as exemplified by Figure 1. The geomorphic view is intended not only to describe them,[3] but also to generate a design space including many others not yet explored. We call it the "geomorphic" view because

[2] A virtual machine can be regarded as a *machine*, in which case communication through the hypervisor and soft switch of the physical machine is regarded as networked communication.

[3] In the geomorphic view Figure 1 would probably correspond to 7 layers, from bottom to top: Ethernet, MPLS, MPLS, IP+UDP+GTP, IP+IPsec, IP+TCP+HTTP, Application.

its possible arrangements of layers resemble the complex arrangements of layers in the earth's crust.

3. TOWARD AN ABSTRACT LAYER MODEL

Our goal is to develop an abstract formal model of a layer, so that we can exploit it to solve the problems presented in Section 1. Because the diversity of network technologies and the overall complexity of networking make this goal appear quixotic, we first explain some of the factors that make this goal feasible, before summarizing our progress.

3.1 What makes a useful layer abstraction seem feasible?

As noted above, each layer instance corresponds to something real and specific, such as a particular LAN, as opposed to a generalization. This means that we can compare a real layer instance to an abstract formal model, and decide without ambiguity whether the layer instance is a special case of the formal model.

Although the geomorphic view is not intended to limit network functions in any way, it is intended to be somewhat prescriptive in how they are described. For one example, a layer has one name space, in which a member has one name. It follows that if there appears to be a process with an "identifier" and a "locator," it must actually be two processes on the same machine, each in a different layer, with one being registered to the other. For another example, there is no concept of tunneling within a layer. Wherever "tunneling" is used, the "tunnel" is a link in one layer, and it "tunnels through" the links of a lower layer. For this reason, the two MPLS layers in Figure 1 must also be two distinct layers in the geomorphic view.

Prescriptive description brings many benefits. Most relevant to this subsection, each layer is a simpler structure, because it need not include multiple name spaces, tunnels, and other unnecessary complexities. Another benefit is that it forces designers to make explicit decisions that are usually left unexplained or even undefined, such as the purposeful relationship between routing in an overlay (whose links are "tunnels") and routing in the underlay (whose routing implements the tunnels). Finally, prescriptive description helps ensure that each architecture has only one correct description, rather than many synonymous ones. This should prove beneficial in comparing architectures and generating design spaces.

Finally, in our current layer model most data structures (*members, attachments, locations, links,* and *routes*) are regarded as centralized, or, more precisely, their distribution over the states of layer members is not specified. In the same way, the functional components that maintain them (*member, attachment, location, link,* and *routing algorithms*) are specified as (centralized) algorithms rather than (distributed) protocols.

This abstraction is extremely important because most of the difficult decisions made in designing a network layer are about or related to how the state is distributed, and how protocols maintain adequate consistency across the layer. Choices range from initializing state structures that cannot change thoughout the life of a layer, through using a centralized database with lookup and update transactions, to highly redundant, distributed state and complex consistency protocols. These choices, in concert with choices about the shape of the member/link graph, the structure of names, and other constraints, are the major influences on scale, performance, and dependability. We cannot defer these choices forever, but the abstraction enables us to defer them until they are relevant.

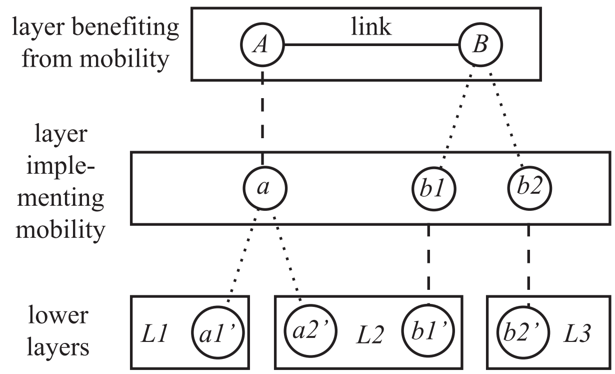

Figure 6: A session with attachment mobility serving endpoint A and location mobility serving endpoint B. Dynamic registrations are shown as dotted lines.

3.2 Progress on the model

We have a formal model of a point-to-point session protocol written in Promela, and have verified many desirable properties with the model-checker Spin [6]. We have a formal model of members, attachments, locations, links, sessions, routes, and the forwarding protocol written in Alloy, and have verified static consistency properties with the Alloy Analyzer [8]. Relatively little is included about the algorithms that maintain these data structures.

Most of our efforts have been expended on investigating and modeling the layer mechanisms needed for *mobility.* In networking, this term refers to both a problem and its solutions. As a problem, it means that a layer member is changing its attachment to lower layers of the network, in particular while using communication services. As a solution, it means maintaining the member's communication channels, despite the movement.

Using the geomorphic view, we have discovered that there are two completely distinct implementations of mobility. Figure 6 shows a layer with two mobile members A and B. Both are benefiting from mobility implementations, in the sense that their link will be preserved as they move. The mobility of each endpoint is provided by a different mechanism.

On the left, as the machine of A moves physically, it goes out of range of LAN *L1* and enters the range of LAN *L2*. Consequently, its attachment in the lowest level changes from *a1'* to *a2'*. In the middle layer, which is implementing mobility, this means that links to a implemented by *L1* are replaced by links to a implemented by *L2*. The hard work in this layer is performed by its routing algorithm, which must re-route to maintain the reachability of a through new links. The parts of the layer state affected by this implementation, called *attachment mobility*, are its attachments, links, and routes.

On the right, as the machine of B moves physically, it goes out of range of LAN *L2* and enters the range of LAN

L3. Consequently, its attachment in the lowest level changes from *b1'* to *b2'*. Although this is exactly the same at this level as *A*'s mobility, It is handled by the implementing middle layer in a completely different manner.

In the middle layer, the location of *B* changes from member *b1* to member *b2*. The most common reason for this to occur is that the implementing layer has a large, hierarchical, topology-dependent name space (as the Internet does). As the machine of *B* moves from *L2* to *L3*, it cannot keep the same name in the middle layer, because topological constraints would be violated. Instead, its member *b1* in the middle layer dies and is reborn as member *b2*. The hard work in this layer is performed by its session protocol and location algorithm. The location algorithm must update *B*'s location, while the session protocol must update distributed session state so that *a* sends messages to *b2* instead of *b1*. The parts of the layer state affected by this implementation, called *location mobility*, are its members, locations, and sessions.

On first hearing this explanation, most people insist that the difference between attachment and location mobility must be illusory, merely a difference of viewpoint. The geomorphic view shows, precisely and unambiguously, that the difference is real: the two mechanisms exercise disjoint protocols and algorithms in a layer, and alter disjoint data structures. This shows the power of an abstract layer model to clarify subtle points.

Of the two mobility implementations, attachment mobility is by far the most familiar. Cellular networks, VLANs [15], and Mobile IP [10] all use variants of attachment mobility. TCP Migrate [13] is one example of location mobility. Section 4 will use mobility to illustrate solving real problems with the layer model.

Our next step toward an abstract layer model will be to investigate and model security mechanisms. Although some aspects of security belong strictly in the operating system or application, others can be provided as part of network services (*e.g.,* encryption and authentication) or supported by network services (*e.g.,* guaranteed routing through security servers). The latter aspects can be investigated and modeled in the same spirit as mobility.

Overall, our abstract layer model is at a very preliminary stage—it is rudimentary or incomplete in most places and overly constraining in some. Nevertheless, our results on mobility are an existence proof that the work is worth doing, and may yield benefits that are difficult to get with other research approaches.

4. USING THE MODEL TO SOLVE PROBLEMS

In this section, we suggest how the previous ideas might contribute to solving the problems introduced in Section 1.

4.1 Communication services

As previously mentioned, restrictive policies embedded in most firewalls and NAT boxes make it very difficult to implement many applications, particularly those requiring real-time, peer-to-peer, or server-to-client communication services. Those restrictive policies exist because of the Internet's security crisis, but they are a blunt instrument.

With a carefully structured and articulated architecture, it should be possible to build in higher-level security policies that are sound and appropriate to their applications. This work could provide the basis of an argument for bypassing the blunt security policies enforced at lower levels of the hierarchy.

The geomorphic view also promotes clean, well-specified interfaces between layers. This improved understanding of network architecture might encourage middleware designers to offer a richer set of communication services to application developers, and might encourage application developers to use well-engineered middleware services rather than program their own versions.

Richer communication services might include channels that are not point-to-point, for example anycast, conference, and publish/subscribe channels. They might include "multihomed" channels that can take advantage of the bandwidth of multiple communication media simultaneously, for example cellular and WiFi transmission. They might include mobility services such as transparent migration of application processes. Or they might include services based on routing through application-specific middleboxes such as privacy servers. All of these services have their place in the geomorphic view of networks.

4.2 Design principles

Our work toward developing design principles breaks into two parts. The first is concerned with solving individual design problems. The second is concerned with composing individual solutions into an architecture that solves many problems simultaneously. We illustrate both parts with a mobility example.

Imagine that we are designing a mobility service for laptops. Laptops are often used on buses, because each bus has its own LAN for the benefit of its riders. Of course, the bus is mobile also. We might see this as breaking into two mobility problems, one for buses and one for laptops. It is important for scalability that solutions to the two problems be independent. In other words, we cannot accept solutions that require an update for every rider when a bus moves, nor can we accept solutions that require an update for the bus when a rider with a laptop gets on or off.

Assuming that there is already a tentative layer architecture, a solution to either of the mobility problems might be assigned to any layer, and it might be a special case of attachment mobility or a special case of location mobility. If a layer *L* is going to implement location mobility for a layer *L+* above it, then the name spaces of *L* and *L+* cannot be in one-to-one correspondence (see Figure 6), and *L* must have an efficient location algorithm that makes the locations of mobile members of *L+* available throughout the layer. If a layer *L* is going to implement attachment mobility, then there are quite a few choices concerning how *L* will optimize the update, storage, and path-stretching costs of dynamic routing (see [9] for an overview of these costs). These choices constitute the design space of a mobility problem. By studying these choices further, we hope to gain more insight into the qualitative and quantitative constraints and trade-offs relevant to solving individual design problems.

Figure 7 shows a new solution to the joint problem, generated from the design space of mobility, that has the independence and scalability advantages explained above. In the figure, the top layer contains a middleware process *M* on a mobile laptop, with an ongoing link to a middleware process *S*. In the layer below, *b* is the Internet interface of

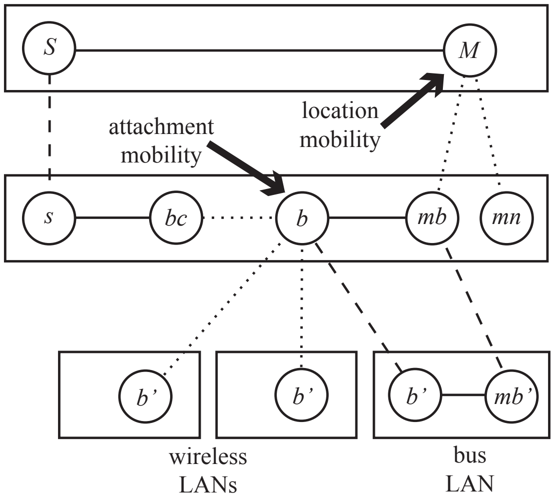

Figure 7: An implementation of mobile laptops on a bus.

a bus LAN. When the laptop is on the bus, M is registered at Internet interface mb, and b is acting as a router for mb. Note that b and mb belong to the same block of Internet addresses, so when the laptop is taken off the bus and attached somewhere else, it will register at a different process mn.

The big arrows indicate the processes benefiting from mobility and the type of mobility, but not where it is implemented. In fact, both mobility problems are solved in the same layer. The middle layer implements location mobility for M. This implementation must do something special to update sessions and locations when the laptop moves on or off the bus, but does nothing when the laptop is on the bus and the bus is moving. The middle layer also implements attachment mobility for b. This implementation must do something special to restore links and routing when the bus moves from the range of one roadside wireless LAN to another, but does nothing when riders get on or off the bus.

In a network architecture, different mobility problems might be solved in different, adjacent layers. We have used the layer model presented in Section 3 to argue that mobility solutions in different layers are independent and non-interfering. In other words, layer composition (in which an overlay uses the services of an underlay) continues to work smoothly.

In Figure 7, different mobility problems are solved by different mobility mechanisms in the same layer—another kind of composition. We are currently completing a proof that location and attachment mobility mechanisms in the same layer are independent and non-interfering, so that they can be composed safely.

One of the goals of composition, both inter- and intra-layer, is more efficient architectures. For example, it has been noted that Figure 1 appears to correspond to 7 layers in the geomorphic view. It is almost certainly possible to implement the same functions in fewer layers. For example, the purpose of IP + UDP + GTP appears to be quality of service, while the purpose of IP + IPsec is security. If the mechanisms implementing these goals can be shown to be independent and compositional, then both can be implemented in the same layer, possibly along with other functions as well.

4.3 Software development

As indicated in Figure 4, the layer model is intended to decompose the software structure of a layer into components with well-understood interfaces and dependencies. The structure in the figure is very coarse, but further work should make it possible to refine it.

Hopefully the refined structure can serve as a framework where specific implementations of mechanisms such as location and attachment mobility plug in. If so, the software framework brings us closer to being able to generate a custom layer by selecting and integrating library components.

5. REFERENCES

[1] D. D. Clark. The design philosophy of the DARPA Internet protocols. In *Proceedings of SIGCOMM*. ACM, August 1988.

[2] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's Internet. *IEEE/ACM Transactions on Networking*, 13(3):462–475, June 2005.

[3] J. Day. *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2008.

[4] A. Feldmann. Internet clean-slate design: What and why? *Computer Communications Review*, 37(3):59–64, July 2007.

[5] M. Handley. Why the Internet only just works. *BT Technology Journal*, 24(3):119–129, July 2006.

[6] G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.

[7] ITU. Information Technology—Open Systems Interconnection—Basic Reference Model: The basic model. ITU-T Recommendation X.200, 1994.

[8] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006, 2012.

[9] J. Mysore and V. Bharghavan. A new multicasting-based architecture for Internet host mobility. In *Proceedings of the 3rd Annual ACM/IEEE International conference on Mobile Computing and Networking*. ACM, 1997.

[10] C. E. Perkins. Mobile IP. *IEEE Communications*, May 1997.

[11] L. Popa, A. Ghodsi, and I. Stoica. HTTP as the narrow waist of the future Internet. In *Proceedings of the 9th Workshop on Hot Topics in Networks*, 2010.

[12] T. Roscoe. The end of Internet architecture. In *Proceedings of the 5th Workshop on Hot Topics in Networks*, 2006.

[13] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proc. MOBICOM*, 2000.

[14] O. Spatscheck. Cloud computing and my worries about the network that enables it. `http://clouds10.mytestbed.net/presentation/oliver_clouds10v2.pdf`, 2010.

[15] M. Yu, J. Rexford, X. Sun, S. Rao, and N. Feamster. A survey of virtual LAN usage in campus networks. *IEEE Communications*, 49(7):98–103, July 2011.

[16] P. Zave. Internet evolution and the role of software engineering. In *The Future of Software Engineering*, pages 152–172. Springer, 2011.