

DOI:10.1145/3274770

**Used in the design of the Internet and Unix, the layered services of the hourglass model have enabled viral adoption and deployment scalability.**

BY MICAH BECK

## On The Hourglass Model

THE HOURGLASS MODEL of layered systems architecture is a visual and conceptual representation of an approach to design that seeks to support a great diversity of applications and allow implementation using a great diversity of supporting services. At the center of the hourglass model is a distinguished layer in a stack of abstractions that is chosen as the sole means of accessing the lower-level resources of the system. This distinguished layer can be implemented using services that are considered as lying below it in the stack as well as other services and applications that are considered as lying above it. However, the components that lie above the distinguished layer cannot directly access the services that lie below it.

David Clark called the distinguished layer the “spanning layer” because in the Internet architecture it bridges the multiple local area network implementations that lie below it in the stack (see Figure 1). Clark defined the function of the spanning

layer by its ability to “... hide the detailed differences among these various technologies, and present a uniform service interface to the applications above” and identified the Internet Protocol as the spanning layer of the Internet (see Figure 2).<sup>5</sup> Arguably the spanning layer also includes other elements of the Internet Protocol Suite that access lower-layer services (such as ARP and DHCP).

The shape suggested by the hourglass model expresses the goal that the spanning layer should support various applications and be implementable using many possible supporting layers. Referring to the hourglass as a design tool also expresses the intuition that restricting the functionality of the spanning layer is instrumental in achieving these goals. The elements of the model are combined visually in the form of an hourglass shape, with the “thin waist” of the hourglass representing the restricted spanning layer, and its large upper and lower bells representing the multiplicity of applications and supporting layers, respectively.

The hourglass model is widely used in describing the design of the Internet, and can be found in many modern networking textbooks.<sup>8</sup> A similar principle has also been implicitly applied to the design of other successful spanning layers, notably the Unix operating system kernel interface by which we mean

### » key insights

- **Adoption of a common service interface is a key to interoperability, portability, and “future-proofing” in the face of rapid technological change.**
- **The design of both the Internet and Unix followed the hourglass principle, leading to dominance in two software markets while also enabling disruptive innovation.**
- **Many successful interface designers adhere to a discipline of simplicity, generality, and limitation of the common interface.**
- **This article introduces the Deployment Scalability Tradeoff, a principle that seeks to explain the reason for the success of this discipline.**

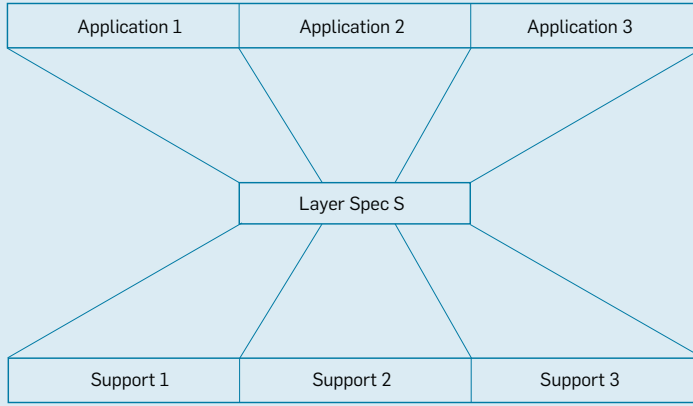
ILLUSTRATION BY PETER GROWTHER ASSOCIATES



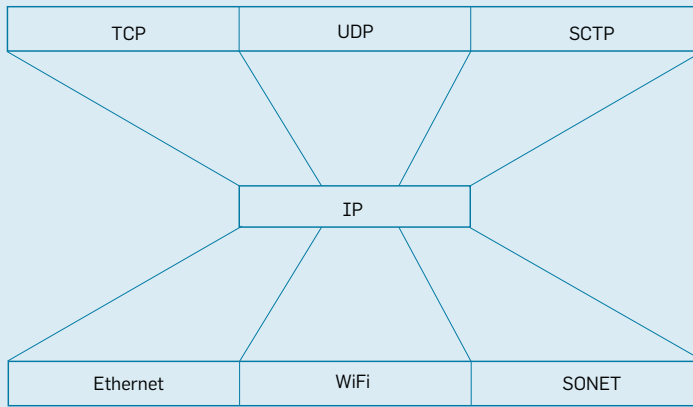




**Figure 1. The hourglass model.**



**Figure 2. The Internet hourglass.**



the primitive system calls and the interactions between user processes and the kernel prescribed by standard manual pages.<sup>9</sup> The impressive success of the Internet has led to a wider interest in applying the hourglass model in other layered systems, with the goal of achieving similar results.<sup>1,7,11</sup> However, application of the hourglass model has often led to controversy, perhaps in part because the language in which it has been expressed is informal.

The purpose of this article is to present a formal model of layering and to use this model to prove some relevant properties.<sup>a</sup> I will then use this formal model to explain the application of the hourglass model in the design of the Internet and Unix, and to show how it relates to some less formal concepts in the design of layered systems.

## Overview

Let's begin by presenting an abstract framework for reasoning about layered architectures and spanning layers in particular. We assume the existence of a certain relationship between layers, namely that one layer specification can support another. This article does not give definitions for layer specifications or the *supports* relation, as the complete formalization is somewhat complex and not, I believe, necessary to understand the argument. Given the existence of the supports relation, I share definitions of possible supports and possible applications of a layer in terms of it. The Hourglass Theorem consists of two simple properties that can be derived from these definitions (see the section on the Hourglass Theorem). These definitions and the properties that we derive from them create a framework for characterizing a spanning layer in terms of the multiplicity

of its possible applications and supports. The Hourglass Theorem, which expresses a trade-off between the multiplicity of possible applications and supports and the logical strength of the spanning layer, is proved. Moreover, the question of whether the Hourglass Theorem provides a formal justification for end-to-end arguments is explored. I then state and argue for the validity of a more general principle—the Deployment Scalability Trade-off—as there is an inherent correlation between deployment scalability of a system with a given spanning layer and the weakness, simplicity, generality and resource limitation of that layer's specification.

The DST is intended as a design principle relating the hourglass design in layered models to the scalability of systems that they describe. The DST combines logical weakness with design criteria that will not be formalized in order to put that principle into a context that also includes more familiar related concepts. The intention is that future work may lead to formalization of some of these other characteristics, the development of metrics, and even a characterization of the trade-offs precise enough to accurately model the implications of specific service design choices.

## The Hourglass

*Definition 1.* A service specification is a formal description of the syntax and necessary properties of an operating system or application-programming interface (API).

A service specification  $S$  describes an interface: it specifies the behavior of certain program elements (functions or subprograms) through statements expressed in program logic. For instance, these might be such statements:

1.  $\forall A, B \in \mathbb{Z} [(A + 1) + B = (A + B) + 1]$
2.  $x, y : \mathbb{N} [\{x > 0\} y := x * x \{y > x\}]$

In formal terms, a service specification is a theory of the program logic. The set of all such specifications expressed in the language of the specific logic is denoted by  $\Sigma$ . In practical terms, a service specification describes the operations of a protocol suite or a programming interface, such as operating system calls.

*Definition 2.* A specification  $S_1$  proves another specification  $S_2$  (written

<sup>a</sup> An undertaking that was suggested to me many years ago by Alan Demers.

$S_1 \vdash S_2$ ) iff  $S_2$  can be derived from  $S_1$  through application of the rules of the logic in which they are both expressed.

**Definition 3.** A specification  $S_1$  is *weaker* than another specification  $S_2$  iff  $S_2 \vdash S_1$ .  $S_1$  is *strictly weaker* than  $S_2$  iff  $S_2 \vdash S_1$  but  $S_1 \not\vdash S_2$ .

**Definition 4.** A *supports* relation  $S \prec_p T$  exists between two service specifications  $S$  and  $T$  and a program  $p$  iff in any model where  $S$  is correctly instantiated, the program  $p$  correctly implements  $T$  using the instantiation of  $S$ .

The supports relation is intended to be analogous to the “reduces to” relation of structural complexity theory.

**The Hourglass Lemma.** The intuition behind this lemma is that any API that can be supported by a given underlying layer can also be supported by any underlying layer that is stronger. Similarly, a layer that can support a given API can also support one that is weaker.

While detailed definitions of service specifications and the supports relation have been omitted here, I call upon the intuition of the reader to justify the following lemma presented here without proof. This lemma is the only place where the omitted basic definitions are used, and the remainder of this discussion is based upon the lemma.

**LEMMA 1.** If  $S_1$  is *weaker* than  $S_2$ , then 1)  $S_1 \prec_p T \Rightarrow S_2 \prec_p T$ , and 2)  $T \prec_p S_2 \Rightarrow T \prec_p S_1$ .

*Proof omitted.*

The two properties that comprise the Hourglass Lemma follow directly from fundamental definitions in program logic, and they also correspond very closely to covariance of return types and contravariance of argument types in object-oriented inheritance.<sup>3,b</sup>

**Pre- and post-images.** Formal analogs to scalability are expressed in terms of how large the sets of service interfaces are that can possibly support or can be supported by a particular specification. To this end, the pre- and post-images of a specification are defined under *supports* (see Figure 3).

These definitions are given relative to the set  $\Pi$  of programs considered as possible implementations of one layer in terms of another. We do not specify  $\Pi$  because we know of no accepted characterization of all “acceptable im-

plementations” of one layer in terms of another. This is certainly a limited class, and is in fact finite since programs that are too large are considered unwieldy from a software engineering point of view. This class also changes over time, as hardware and software technology changes the set of available implementation tools.

**Definition 5.**  $pre_{\Pi}(S) = \{T \mid \exists p \in \Pi [T \prec_p S]\}$

**Definition 6.**  $post_{\Pi}(S) = \{T \mid \exists p \in \Pi [S \prec_p T]\}$

In representing the set  $\Pi$  in our model as an external parameter we are not accounting for software engineering aspects of these definitions.

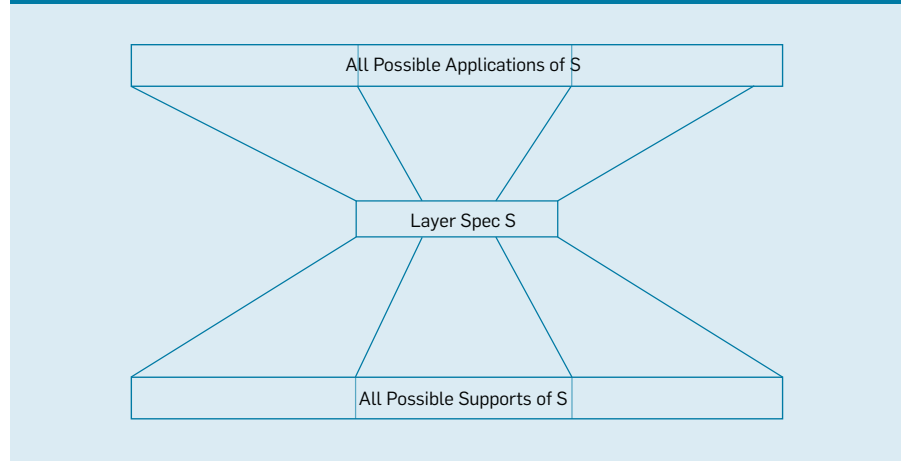
**Using the hourglass as an analytical tool.** Reference to the hourglass model is sometimes conflated with the idea of the spanning layer as a standard enforced by some external means such as legal regulation or as a condition of membership in some community.

However, we can use the analysis of pre- and post-images of the supports relation as tools to analyze a layered system without necessarily relating it to any standards process.

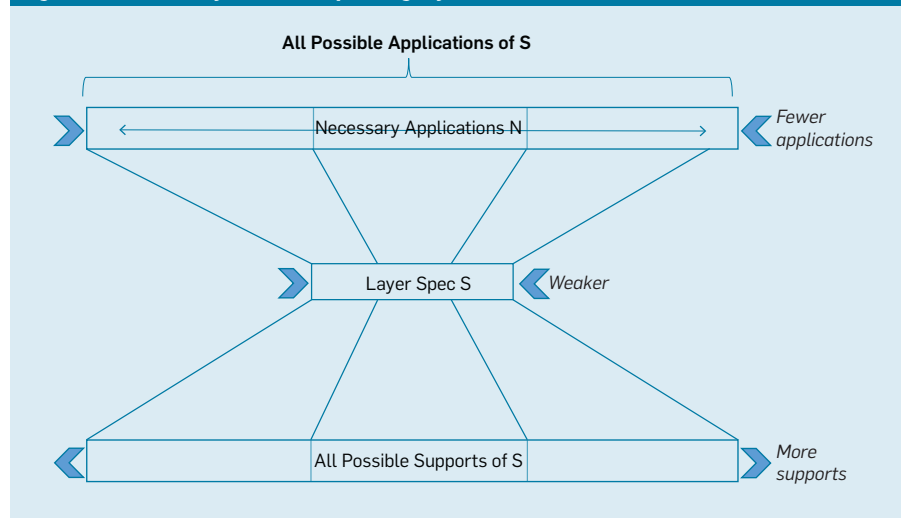
By selecting any set of services at one level of a layered system, we can ask what the design consequences would be if it were adopted as the spanning layer of a hypothetical system. Adoption as a spanning layer means that no other services would be available at that layer. Any participant in the system would have to use it as the sole means of accessing the services and resources of lower layers. Viewed in this way, the pre-image of the supports relation denotes all possible implementations of the prospective spanning layer and the post image denotes all of its possible applications.

I use the term “denotes” because the pre- and post-image are not necessarily useful in actually enumerating these

**Figure 3. Pre- and postimages.**



**Figure 4. A minimally sufficient spanning layer.**



<sup>b</sup> This observation courtesy of Tim Griffin.

sets of specifications, since there is no formal specification for the value of  $\Pi$ , nor a way of determining whether a particular program  $p$  is in  $\Pi$ . Even when there is community agreement that certain programs are either in  $\Pi$  or in its complement, there may still be contention regarding some boundary cases.

Taking a descriptive view of the hourglass allows us to use it as an analytical or predictive tool to understand the impact of a community's adopting a particular interface as a standard, be it de jure or de facto. Making the distinction between the use of the hourglass as a descriptive tool or as a means of justifying a standard also explains how different hourglasses can be examined and compared within the discussion of the same layered system. Every prospective spanning layer has an associated pre- and post-image, regardless of whether it is considered for any kind of standardization.

### The Hourglass Theorem

This theorem is central to our understanding of the hourglass model.

**THEOREM 1.** *If a specification  $S_1$  is weaker than another specification  $S_2$ , then 1)  $\text{post}_\Pi(S_1) \subseteq \text{post}_\Pi(S_2)$ , and 2)  $\text{pre}_\Pi(S_1) \supseteq \text{pre}_\Pi(S_2)$ . Proof:*

1. *By definition,  $T \in \text{post}_\Pi(S_1)$  iff  $\exists_p \in \Pi [S_1 \prec_p T]$  so by Lemma 1  $S_2 \prec_p T$ , thus  $T \in \text{post}_\Pi(S_2)$*


2. *The proof is symmetric to Part 1.* □

The Hourglass Theorem conveys (approximately) that a weaker layer specification has fewer possible applications but more possible supporting layers than a stronger layer specification.


### Minimal Sufficiency

In terms of the hourglass shape, the thin waist (weak spanning layer, as noted earlier) naturally tends to give rise to the large lower bell of the hourglass (many supports). However, a weaker spanning layer also tends to give rise to a smaller upper bell (fewer applications). Thus, some countervailing element must be introduced into the model to ensure it is in fact possible to implement all necessary applications (see Figure 4).

As a design goal, we model the necessity of implementing certain appli-



**The thin waist of the hourglass is a narrow straw through which applications can draw upon the resources that are available in the less restricted lower layers of the stack.**



cations by introducing the set of necessary applications as another external parameter  $N \subseteq \Sigma$ .

**Definition 7.** A specification  $S$  is sufficient to support a set of specifications  $N$  iff  $N \subseteq \text{post}_\Pi(S)$ .

A spanning layer must be strong enough to support all necessary applications, but the stronger it is the fewer possible supports it has. The notion of minimal sufficiency serves as a means to balance these two design requirements:

**Definition 8.** A specification is *minimally sufficient* for  $N$  iff it is sufficient for  $N$  but there is no strictly weaker specification sufficient for  $N$ .

The balance between more applications and more supports is achieved by first choosing the set of necessary applications  $N$  and then seeking a spanning layer sufficient for  $N$  that is as weak as possible. This scenario makes the choice of necessary applications  $N$  the most directly consequential element in the process of defining a spanning layer that meets the goals of the hourglass model.

Note the implication that the trade-off between the weakness of the spanning layer and its sufficiency for a particular set of applications  $N$  is unavoidable. This suggests the design of a spanning layer may have a tendency to fail if it attempts to both achieve a high degree of weakness and also be sufficient to support a large set of necessary applications.

### End-to-End Arguments

End-to-end arguments have influenced the design of many layered systems, most famously the Internet. Historically, end-to-end arguments have often been invoked in discussions of whether it is appropriate to add functionality to a layer of the network, and, in particular, when discussing the Internet's spanning layer.

Claims have often been made that an end-to-end argument implies that adding functionality to the spanning layer will result in a diminution of the scalability of the Internet, although this term does not have a generally agreed-upon definition. Here, the hourglass model is used as a reference, to hypothesize that scalability is enabled by a spanning layer that has implementations using as many different supports as possible, given the necessary applica-

tions that it in turn supports. The analysis is that end-to-end arguments do not necessarily lead to a spanning layer that maximizes possible supports.<sup>c</sup>

The introduction of the classic paper “End-to-End Arguments in System Design”<sup>10</sup> gives a general statement of the argument that applies to all kinds of layered systems:

“The argument appeals to application requirements, and provides a rationale for moving function upward in a layered system, closer to the application that uses the function.”

However, the discussion then focuses on the more specific context of layered communication systems, and the argument is described again:

“The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)”

Much of the paper is devoted to the context of layered communication systems. Examples of issues implementing functions in lower layers fall into two major categories: When the lower layer lacks knowledge of application requirements or status, and when local communication functions are combined to create a global service, so the characteristics of the global service can only be detected by its clients.

Moving function upward in a layered system can have the effect of removing responsibility for particular functionality required by applications from lower layers. This leaves higher layers free to implement their true requirements without imposing costs or other artifacts due to inappropriate functionality being implemented by lower layer services. However, when applied to the spanning layer, end-to-end arguments do not necessarily lead to a design that is logically weaker, and thus has more possible supports.

Examples can be found in which moving function upward in a layered

system indeed leads to a weaker spanning layer. However, other examples can be given in which it leads to a stronger one. These examples have been omitted due to lack of space, but their existence suggests this question: Why have end-to-end arguments been so commonly invoked in discussions of scalability?

This analysis of the relationship between the hourglass model and end-to-end arguments is included because those arguments are often cited as a founding principle of the Internet, and credited as a major reason for its remarkable success. In fact, it was interest in understanding the power that is attributed to end-to-end arguments that led us to formalize scalability in layered systems and hypothesize logical weakness as an underlying cause. Finding no necessary causal connection between end-to-end arguments and logical weakness was unexpected, and the result is indeed noteworthy.

In light of this result, we offer a hypothesis for the apparent impact of end-to-end arguments on the scalability of the Internet: In the cases where application of an end-to-end argument results in a weakening of the spanning layer while still supporting all necessary applications, the result may be an increase in possible supports due to that weakening. If the result is an increase in scalability that increase may be attributed to the end-to-end argument, even if the effect is more specifically due to increased weakness. If this hypothesis is true, it would explain why end-to-end arguments are a relevant but inexact tool in the design of layered systems for maximum scalability.

Saltzer et al.<sup>10</sup> present a very general design approach for the placement of specific functionality in layered systems: keep the lower layers general in order to allow the specific requirements of higher layers to be most effectively addressed. This approach is rooted in the methodology of formal reasoning in logic, mathematics and the sciences, but its application was informed by experience in the design and implementation of complex systems, with Multics being cited most often. As practitioners of a field grounded in principle as well as practice, computer scientists are drawn to ask why this approach has sometimes

seemed so powerful in conferring scalability and how it could be used to predict effects on system scalability of different design alternatives.

We understand Salter et al.’s classic paper as finding a justification for the more general argument in the particulars of layered communication systems. To the extent the Hourglass Theorem is a causal element in system scalability, it is not necessarily applicable to explain the effectiveness of end-to-end arguments. We now turn to other aspects of the hourglass model that have traditionally been associated with scalability and attempt to relate them to the formal model of layered systems and to end-to-end arguments, although we do not have formal results analogous to the Hourglass Theorem to justify claims of causal linkage. To fit these other aspects of the hourglass model into our framework is a step toward developing a more complete formalization.

### Spanning Layer Characteristics

Consider a design space of spanning layers that can support a particular set of necessary applications. Each point in this space is characterized in a number of ways, according to its logical or engineering attributes. One important job of a system architect is to find a point in this design space in which certain goal attributes fall into target ranges by adjusting the values of those attributes that are under their control. From this perspective, the subspace of feasible designs has some shape that the system architect must understand and navigate in order to reach their design goals.

The hourglass model can be understood as describing the general shape of the subspace that we navigate in designing layered systems. If one goal is maximizing possible supports, then the Hourglass Theorem tells us that the slope of the subspace of feasible solutions when considering this goal as a function of the logical weakness of the spanning layer is non-negative. We have no metrics for logical strength or for the size of the space of possible solutions, only for the notions of one service description being weaker than another and one set of service descriptions being included in another.

These definitions allow system architects to reason about the sign of

<sup>c</sup> Jerry Saltzer illuminated the point that the hourglass model is distinct from end-to-end.



the slope, but not its steepness nor what value is necessary in order to achieve a particular design goal. Because only the relationship between one independent attribute and one dependent goal attribute has been formalized, there are no results about interactions between the various dimensions. The ability to obtain such abstract results is one strength of mathematical logic; the fact that such results are not more specific and perhaps more satisfying to readers unfamiliar with logic may seem to some as a weakness. The purpose here is to create a structure for the definition of metrics and the proof of further properties by researchers in the field.

With this goal in mind we now consider a number of other spanning layer attributes (simplicity, generality, and resource limitation) that have been viewed as important within the design community, and present a hypothesis for how they act together to impact the overall goal of system scalability. These choices and the explanations offered reflect the author's study of and experience as a researcher in the fields of operating systems and wide area network services.

**Simplicity.** The attribute of simplicity is one aspect of the thin waist of the hourglass. (Note, simplicity is not correlated with logical weakness, as the strongest possible predicate is the primitive assertion "False," which is also the simplest.) Simplicity is an important aspect of the acceptability of the spanning layer as a tool used in the implementation of higher layer services.

A key aspect of simplicity is orthogonality. In a service interface, orthogonality means there is only one way of gaining access to any fundamental underlying service or resource. Redundant features increase the complexity of an interface without making it logically stronger. System architects understand the value of orthogonality in the design of interfaces and are more likely to accept as a community standard a design that has this form of simplicity.

An example of orthogonality in the Unix system call interface is the decomposition of file movement between directories into the creation of a physical link (using `link()`), creating a copy of a pointer in a destination location, followed by deletion of the original (using

`unlink()`). The composite file movement operation is implemented in a user level command (`mv`). This allows the user level file movement operation to be easily generalized to include movement between physical volumes (which requires copying of contents) and for efficient file sharing within a volume to be implemented using `link()`.

**Generality.** It is often observed the diversity of applications supported by the Internet far outstrips those foreseen by its original designers. Rather than crafting a spanning layer to support the functionality of only the initial target applications, the designers created a set of general primitives such that those target applications lay within the space of applications supported by them. That space also contains many other applications, including many they may not have originally foreseen.

In terms of our analysis of the hourglass model, the design of the spanning layer  $S$  yielded a very rich set of possible applications  $post_H(S)$ . The design challenge was to do so without increasing the logical strength of the spanning layer. Our analysis of end-to-end arguments leads us to associate their application with the design of a general spanning layer. This may help to explain why, even in cases where applying an end-to-end argument does not result in a weakened spanning layer, there may still be an increase in the class of supported application due to greater generality.

**Resource limitation.** The spanning layer provides an abstraction of the resources used in its implementation, preventing them from being accessed directly by applications.<sup>5</sup> As such, it also defines the mechanisms that allow those resources to be shared by applications and among users. In some communities, the modes of sharing are open, with few restrictions in place to ensure fairness among users (such as resource quotas). Such openness is one way of enabling the spanning layer to be logically weak (such as by not implementing authorization, metering and billing of resource utilization.) One way of managing more open modes of resource sharing is to limit the resources used by any individual service request, requiring large allocations of resources to be fragmented, as in statistical multiplexing.<sup>6</sup>

Such fragmentation allows for more fluidity in the allocation of resources, with competition between users occurring on a finer scale. This point is perhaps clearest when comparing extremes, such as the provisioning of a virtual circuit of unbounded duration compared to the forwarding of a single datagram with a bounded Maximum Transmission Unit. A similar extreme comparison can be made between the allocation of a disk partition for an unbounded period of time and obtaining a time-limited lease on a single storage object with a maximum size. Resource limitation, along with the definition of acceptable algorithms for aggregating individual allocations (for example, "TCP friendly" flow control in Internet applications) means that use of the specification will not result in overtaxing the resources of the platform on which it is implemented.

In other words, the thin waist of the hourglass is also a narrow straw through which applications can draw upon the resources that are available in the less restricted lower layers of the stack. Resource limitation can affect the ability of the system to function in environments where the demand for resources locally or transiently exceeds the capacity of the system. A countervailing consideration is performance, as extremely fine-grain contention for resources can impose unacceptable overheads.

## Deployment Scalability

End-to-end arguments have sometimes been cited as a reason that failure to keep complex functionality out of the spanning layer (maintaining a thin waist) will limit the scalability of a layered system. We have constructed a formal framework for analyzing layered systems and sought to use it understand the effect of end-to-end arguments. We have tried to characterize aspects of thinness in both formal and informal ways, and now seek to use this model to account for the design principle that motivates maintaining the thin waist of the hourglass.

The Hourglass Theorem has shown a structural link between the logical weakness of the spanning layer and an expansion in the set of possible supports. I have argued informally that sim-

licity, generality, and resource limitation can also affect possible supports and applications.

The design principle sought to examine is that thinness of the spanning layer is correlated with greater success in its adoption and longevity. This is sometimes expressed as the system exhibiting scalability. However, scalability means little if we do not specify the attribute in which we desire the system to exhibit an ability to grow.

*Deployment scalability* is used to characterize a spanning layer being adapted to finding success in the form of widespread adoption. Deployment scalability is intended to imply the kind of “viral” adoption that the Internet and Unix spanning layers have exhibited. This definition is proposed as a (admittedly imprecise) characterization of success in global infrastructure service interface design.


*Definition 9.* Deployment scalability is defined as widespread acceptance, implementation, and use of a service specification.

The notion of deployment scalability is introduced in order to have a vocabulary for expressing the goal that is implicit in the design of a spanning layer for global infrastructure.


For example, in describing the role of the Internet’s thin waist, Peterson and Davie<sup>8</sup> state “The hourglass’s narrow waist represents a minimal and carefully chosen set of global capabilities that allows both higher-level applications and lower-level communication technologies to coexist, share capabilities and evolve rapidly.” It is the meaning of “minimal” and “carefully chosen” that we are trying to characterize.

In terms of the formal model of layered systems, we suggest that having many possible supports and many possible implementations is correlated with the goal of deployment scalability. The Hourglass Theorem would then extend this to a correlation between minimal sufficiency and deployment scalability. We have given some informal arguments to support similar relationships between other aspects of a thin spanning layer and deployment scalability.

Each of these aspects can be evaluated in isolation, but in the service of our original motivation to link the thin waist to a general notion of scalability,



## The Hourglass Theorem has shown a structural link between the logical weakness of the spanning layer and an expansion in the set of possible supports.



we offer the Deployment Scalability Trade-off (DST): There is an inherent correlation between deployment scalability of a system with a given spanning layer and the weakness, simplicity, generality, and resource limitation of that layer’s specification.

The original motivation for creating a formal model of layered systems was to better understand end-to-end arguments in the context of the hourglass model. We sought to explain and guide efforts to generalize shared network resources while addressing the intuitive design principle that the requirements of scalability had to be the primary and overriding constraint. The DST is a candidate as a more general design principle that situates end-to-end arguments in a complex space of design criteria. The Hourglass Theorem is a first step in an explanation of the role of logical weakness in the DST.

### Examples and Applications

Giving an account of an application of the Hourglass Theorem can be tedious. The antecedents of the theorem require the definition of the specification language, a program logic, all acceptable programs  $\Pi$  and the set of necessary applications  $N$ . This presentation will be restricted to giving a less formal account of the implications of the DST.

**Tree building in IP multicast.** Global Internet routing is made possible through the use of interoperable approaches to internal and external routing within and between local networks. The metrics assigned to individual links by network administrators are somewhat arbitrary, but when used as inputs to a combination of shortest path algorithms (interior gateway protocols) and a policy driven peering protocol based on commercial agreements (the Border Gateway Protocol), the result is often acceptably similar to some intuitive notion of efficiency.

Multicast routing is much more complex. An IP multicast group is based on tree-structured forwarding, with the tree being built dynamically as clients join and leave the group. The notion of efficiency in multicast must not only account for the path taken from the source to each receiver, but also the amount of control communication required during discovery of the paths that actually reach receivers



(such as flood and prune), maintaining the tree and responding to changes in topology. Algorithms that maintain accurate trees require persistent state at intermediate nodes, which results in the spanning layer being strengthened.

Historically, a number of protocols have been proposed that perform well in different environments, with particular bifurcation between groups that are sparse in the subnets they reach (with a low degree of branching toward the leaves of the tree) and those that are dense (with a higher degree of branching toward the leaves). Because different candidate protocols perform better in different scenarios, multiple implementation approaches have been maintained by network providers and selected by applications.

The resulting “fat” multicast spanning layer has limited simplicity and generality in not offering a single universal solution. The best choice for a particular situation may be unclear, or may change over time. This has arguably contributed to the lack of continuous, universally available deployment of IP multicast throughout the Internet. Application builders have used overlay multicast and repeated unicast as workarounds at the cost of redundant traffic.

**Internet address translation.** Network Address Translation (NAT) is a technique for allowing sharing of an IP address by multiple endpoints within a subnetwork. NAT uses DHCP to assign local addresses to endpoints within a “NATed” subnetwork that cannot in general be reached by datagrams sent from outside. The NAT-aware router then translates local addresses to use a single externally reachable source IP address on TCP connections initiated by clients within the NATed subnet. UDP protocols can also be supported.

The ability of a router to interpose itself between end points in a NATed subnetwork and external servers allows the semantics of TCP connections initiated from within the subnetwork to match the specification of the non-NATed network. The most common cases are connections between a Web browser or other client within the network and external servers. However, connections from outside the NATed subnet to endpoints within it are not possible without additional administrative intervention.

Viewing the Internet in terms of the hourglass model, adding NATed subnetworks to the implementation is a weakening of the IP spanning layer. The global reachability condition that datagrams can be sent from any sending endpoint to any receiver’s IP address does not hold in the NATed Internet. This breaking of symmetry in reachability is often viewed as a weakness of NAT.

In spite of arguments against it, NAT has become ubiquitous in the consumer Internet. While NAT does solve a problem with scarcity of IPv4 addresses, there are other ways to allow sharing of a single IP address by many nodes, some of which maintain symmetric reachability. Our analysis suggests the logical weakness of the NATed Internet’s design may in fact help to explain its greater deployment scalability.

By abandoning symmetric reachability, the NATed Internet trades-off a logically weaker spanning layer against an expanded class of possible supports. This comes at the expense of excluding some possible applications that require global reachability (such as pure peer-to-peer systems). The exclusion of some applications has been generally acceptable to the community of commercial Internet users, sometimes using workarounds created by the providers of commercial peer-to-peer services that require general reachability.

Users of applications that require symmetric reachability have responded by working within a separate community of interoperability, sometimes connecting to non-NATed networks such as those at many universities and research laboratories using Virtual Private Networks. This bifurcation is made more acceptable by the fact that most home and business users do not require global reachability. In this analysis, the broader support possible for NAT has overcome resistance due to violations of layering and lack of symmetric reachability.

**Process creation in Unix.** In early operating systems it was common for the creation of a new process to be a privileged operation that could be invoked only from code running with supervisory privileges. There were multiple reasons for such caution, but one was that the power to allocate operating system resources that comprise a new process was seen as too great to be delegated to the application level.

Another reason was the power of process creation (for example, determining the identity under which the newly created process would run) was seen as too dangerous. This led to a design approach in which command line interpretation was a near-immutable function of the operating system that could only be changed by the installation of new supervisory code modules, often a privilege available only to the vendor or system administrator.

In Unix, process creation was implemented by the `fork()` system call, a logically weaker operation that does not allow any of the attributes of the child process to be determined by the parent, but instead requires that the child inherit such attributes from the parent.<sup>9</sup> Operations that changed sensitive properties of a process were factored out into orthogonal calls such as `chown()` and `nice()`. These were fully or partially restricted to operating in supervisory mode or integrated with `exec()` (which is not so restricted) using `chmod()` and the set-user-ID bit. The decision was made to allow the allocation of kernel resources by applications, which allows the possibility of “fork-bomb” denial of service attacks.

The result of this design was not only the ability to implement a variety of different command line interpreters as nonprivileged user processes (leading to innovations and the introduction of powerful new language features) but also the flexible use of `fork()` as a tool in the design of multiprocess applications. This design approach has allowed the adaptation of kernels that implement the Unix-based POSIX standard to run on mobile and embedded devices that could not have been anticipated by the original designers.

**Caching metadata in HTTP.** The World Wide Web established HTTP as a near-universal protocol for accessing persistent data objects using a global namespace (commonly referred to as the REST interface). This general use of HTTP has created a community of interoperation that has adopted it as a spanning layer.

The original specification of the HTTP protocol did not include any requirement of consistency in the objects returned in response to independent but identical HTTP requests. However, in the common case where

HTTP responses are based on a collection of stored objects they exhibit stability over time and consistency between clients. Temporal stability is the basis of caching implemented in Web clients and additional consistency between different clients enabled shared Web caching.<sup>4</sup> However, this stability is not perfect and in particular does not hold for dynamic HTTP responses that are the result of arbitrary server side computation. This can result in the return of stale cache responses.

By using the HTTP Cache-Control header directives in an HTTP response, the server can declare the extent of temporal stability, stability across clients, or the complete lack of stability in that response. If servers respect the stability guarantees declared in Cache-Control directives, Web caches can use them to ensure correctness of their responses.

Viewed as a service specification, HTTP with a requirement for accuracy in Cache-Control directives is logically stronger because it enables accurate assertions to be made regarding the correspondence between such metadata and server responses. In terms of the Hourglass Theorem, the weakness of the less constrained interpretation of HTTP without accurate caching metadata allows for looser implementations. This is traded off against the ability to support applications that require consistency in HTTP responses.

In practice, the ease and cost savings of ignoring consistency of lifetime metadata in server content management has generally won out over the ability to support applications requiring consistency. While Web browsers do take advantage of temporal consistency, they also sometimes return stale responses and require end users to intervene manually. The popularity of shared HTTP caches has been hampered by their inability to ensure consistency. The inefficiency of uncached HTTP in delivering stable responses has largely been countervailed by the trend toward increasing bandwidth in the Internet, although it is a significant factor inhibiting the deployment scalability of the Internet in parts of the world where network bandwidth is highly constrained.

**Designing a spanning layer for node services.** Network architects have long sought to define an interface to enable interoperation in the creation of new

services using the generalized local transfer, storage and processing services of network intermediate nodes. Examples of such efforts include active networking, middleboxes, the computational grid, PlanetLab, and GENI, as well as current efforts at defining containers for computational workloads. A full survey is beyond the scope of this article.

Nodes that comprise such general networks are variously characterized as virtual machines or programmable routers. A standard interface to local node services would act as a spanning layer defining a community of interoperability in service creation. Many current proposals for such a standard define spanning layers that are logically strong, for instance, allowing for the guaranteed reservation of resources.

The Hourglass Theorem can be the basis for an argument that such a spanning layer should be chosen so as to be minimally sufficient for a set of necessary applications in order to maximize the number of possible supports.<sup>2</sup> If we accept the DST as a more general design rule, then simplicity, generality, and resource limitation should also be maximized.

A review of current proposals may reveal an acceptance of strong assumptions, complexity, specialization, and unbounded resource allocation as “necessary.” If so, the DST suggests such designs may suffer diminished deployment scalability, which can be detrimental in any standard so vital to the future of global information infrastructure.

## Conclusion


This article is intended as a first step in a research program to devise a common language for analyzing the design of spanning layers in layered systems of all kinds and predicting the outcomes of such designs. The primary technical contribution is the formulation of a layered system of service interfaces in terms of program logic. This yields a definition of “deployment scalability” that seeks to capture the intent of the hourglass model.

The further discussion of other aspects of the thin waist is intended to capture some of the informal arguments that have been made about the design of the spanning layer. The Deployment Scalability Tradeoff is a general design principle intended to fulfill a role in ar-

guing for thinness. All aspects of this characterization seem ripe for further formalization and refinement.

## Acknowledgments

Thanks to my long-time collaborator Terry Moore for his encouragement and philosophical dialog, to my comrade Tim Griffin for his critical support, and to Jerry Saltzer, Joe Touch, Rick McGeer, Bob Harper, Glenn Riccart, Elaine Wenderholm and the reviewers for their insightful comments.

This work was performed under financial assistance award 70NAN-B17H174 from the U.S. Department of Commerce, National Institute of Standards and Technology. 

## References

1. Akhshabi, S. and Dovrolis, C. The evolution of layered protocol stacks leads to an hourglass-shaped architecture. In *Proceedings of the ACM SIGCOMM 2011 Conference*. ACM, New York, NY, 206–217.
2. Beck, M., Moore, T., Luszczek, P. and Danalis, A. Interoperable convergence of storage, networking, and computation. *Advances in Information and Communication. Lecture Notes in Networks and Systems 70*. Springer, 2020, 667–690.
3. Cardelli, L. A semantics of multiple inheritance. *Information and Computation*. Springer-Verlag, 1988, 51–67.
4. Chankhunthod, A., Danzig, P.B., Needaels, C., Schwartz, M.F. and Worrell, K.J. A hierarchical Internet object cache. In *Proceedings of the 1996 USENIX Technical Conference*, 153–163.
5. Clark, D.D. Interoperation, open interfaces and protocol architecture. *The Unpredictable Certainty: White Paper*. The National Academies Press, Washington, DC, 1997, 133–144.
6. Fagg, G., Moore, T., Beck, M., Wolski, R., Bassi, A., Plank, J.S., and Swamy, M. The Internet backplane protocol: A study in resource sharing. In *Proceedings of the 2<sup>nd</sup> IEEE/ACM Intern. Symp. Cluster Computing and the Grid*.
7. Foster, I., Kesselman, C. and Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations. *The Intern. J. High Performance Computing Applications 15*, 3 (2001), 200–222.
8. Peterson, L.L. and Davie, B.S. *Computer Networks, A Systems Approach, 5th Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.
9. Ritchie, D.M. and Thompson, K. The Unix time-sharing system. *Commun. ACM 17*, 7 (July 1974), 365–375.
10. Saltzer, J.H., Reed, D.P. and Clark, D.D. End-to-end arguments in system design. *ACM Trans. Comput. Syst.* 2, 4 (Nov. 1984), 277–288.
11. Shilton, K., Burke, J., Zhang, L. and Claffy, K. Anticipating policy and social implications of named data networking. *Commun. ACM 59*, 12 (Dec. 2016), 92–101.

**Micah Beck** (mbeck@utk.edu) is an associate professor in the Department of Electrical Engineering and Computer Science at University of Tennessee, Knoxville, and is currently with the Office of Advanced Cyberinfrastructure in the National Science Foundation. The work discussed here was completed prior to his government service and does not reflect the views, conclusions, or opinions of the National Science Foundation or of the U.S. Government.

Copyright held by author/owner.  
Publication rights licensed to ACM. \$15.00.



Watch the author discuss this work in the exclusive *Communications* video.  
<https://cacm.acm.org/videos/on-the-hourglass-model>