A Material History of Bits

Jean-François Blanchette
Department of Information Studies, Graduate School of Education and Information Studies, University of California, Los Angeles, 218 GSEIS Bldg., Box 951520, Los Angeles, CA 90095-1520.
E-mail: blanchette@ucla.edu; Web: http://polaris.gseis.ucla.edu/blanchette

In both the popular press and scholarly research, digital information is persistently discussed in terms that imply its immateriality. In this characterization, the digital derives its power from its nature as a mere collection of 0s and 1s wholly independent from the particular media on which it is stored—hard drive, network wires, optical disk, etc.—and the particular signal carrier which encodes bits—variations of magnetic field, voltages, or pulses of light. This purported immateriality endows bits with considerable advantages: they are immune from the economics and logistics of analog media, and from the corruption, degradation, and decay that necessarily result from the handling of material carriers of information, resulting in a worldwide shift "from atom to bits" as captured by Negroponte. This is problematic: however immaterial it might appear, information cannot exist outside of given instantiations in material forms. But what might it mean to talk of bits as material objects? In this paper I argue that bits cannot escape the material constraints of the physical devices that manipulate, store, and exchange them. Such an analysis reveals a surprising picture of computing as a material process through and through.

Introduction

By some accounts, the digital age fundamentally differs from all previous information epochs insofar as information has finally achieved what it has aspired to throughout history, namely, unburdened itself from the shackles of matter. As a mere collection of 0s and 1s, digital information is independent of the particular media on which it is stored—hard drive, optical disk, etc.—and the particular signal carrier which encode bits, whether magnetic polarities, voltage intensities, or pulses of light.

This purported independence from matter would have two distinct and important consequences: (a) digital information can be reproduced and distributed at negligible cost and high speed, and thus, is immune to the economics and logistics of analog media; (b) digital information can be accessed, used, or reproduced without the noise, corruption, and degradation that necessarily results from the handling of material carriers of information. Immateriality, then, is fundamental to the ability of the digital to upend the analog world, the reason why any media that can be digitized or produced digitally will eventually succumb to the logics of digital information and its circulation through electronic networks—an argument powerfully encapsulated by Negroponte's (1995) slogan, "from atom to bits."

Such a characterization is quite problematic. If bits are not made of atoms, what could they possibly be made of? In this paper, I argue, as common sense intuitively suggests, that bits are necessarily both logical and material entities. Furthermore, as the theoretical and empirical material presented in this paper will demonstrate, computing systems are suffused through and through with the constraints of their materiality. I thus use materiality as an entry point in the analysis of the computing infrastructure, the infrastructure that already mediates a breathtaking proportion of social relations—from education and healthcare to the search for romantic partners.

The computing infrastructure—e.g., operating systems, networking protocols—is precisely tasked with relieving users and programmers from the specifics constraints of the material resources of computation: within a given platform, applications run regardless of processor type, storage media, or network connection. Yet, this abstraction from the material can never fully succeed. Rather, it stands in dialectical tension with the evolution of these material resources and with the efficiency trade-offs their abstraction requires. Materiality then is a key analytical category from which to track the complex positioning of market players as they respond to fundamental shifts in infrastructure—wireline to wireless, single to multicore, desktop to cloud and mobile. Indeed, the characteristics of this infrastructure matter a great deal, since it determines the base material conditions under which applications, services, and devices will perform (Engler & Kaashoek, 1995).

Furthermore, a focus on materiality highlights that computation is a mechanical process based on the limited resources

Received July 27, 2010; revised March 4, 2011; accepted March 4, 2011

 DOI: 10.1002/asi.21542

of processing power, storage, and connectivity. Indeed, the computing professions devote much of their activity to the management of these limitations. In mediating access to the physical resources of computation, infrastructure software must also manage the competing demands users place on them. A material analysis foregrounds how systems design must necessarily engage in the oldest political problem in the world: the allocation of scarce resources among competing stakeholders. While the shift to cloud computing, the defining infrastructural work of our time, is typically framed either in the language of technical rationality or that of the information age's infinite frontier, materiality provides for an analysis of infrastructure building in terms of the politics of resource allocation. Indeed, a focus on materiality suggests a profound disconnect between such political work and the self-portrayal of computing science as primarily concerned with the design of efficient abstractions (e.g., Wing's [2006] "computational thinking").

There is thus much to be gained—theoretically, methodologically, empirically—from approaching bits as materials objects. Yet various factors, including the trope of immateriality, have resulted in inadequate theorization of this fundamental dimension of information systems. Given this, the argument that follows will require delving into the nitty-gritty of the technical worlds where the constraints of that materiality are confronted. This exploration will take the form of technical histories of major system abstractions—the von Neumann machine, the file, the packet—that will retrace efficiency trade-offs resulting from shifts in the material basis of computation. Such design histories reveal the digital world's independence from material as permanently unsettled, under constant pressure to renegotiate the exact terms of that liberation.

This, then, is a paper about stuff, the stuff of computation (Miller, 2009). I begin by tracing the development of the immaterial trope and its impact on professional practice, and review recent attempts at analyzing the materiality of the digital. Building on this prior work, I focus on the design strategy of modularity functions as the core mechanism for abstracting, structuring, and distributing the material resources of computation, at the cost of efficiency trade-offs. Three empirical sections then illustrate the historical evolution of such trade-offs in the context of major computational resources—processing, storage, and connectivity. I conclude by discussing the implications of this argument for historical studies of computing, systems design, and governance.

Information, Immaterial

The trope of immateriality is not a new phenomenon by any measure. While William Gibson's *Neuromancer* (1984) precipitated the term "cyberspace" into public consciousness, it also reiterated for a new set of technologies long-standing themes in the history of electronic communications, dating back at least to the telegraph: "the promise of telegraphy is metaphysical: by annihilating space and time, it allows humankind to escape physical limitations. The power and ubiquity of electricity are metaphorically attached to a newly disembodied consciousness" (Rosenheim, 1997, p. 93). Networked computers have provided renewed valence to this promise, as articulated with great lyrical force in the defining mid-90s manifesto of the Internet, Barlow's "A Declaration of the Independence of Cyberspace" (1996). Barlow placed immateriality at the center of his analysis of cyberspace as a place altogether distinct from the material world:

> "Governments of the Industrial World, you weary giants of flesh and steel, I come from Cyberspace, the new home of Mind. . . . Your legal concepts of property, expression, identity, movement, and context do not apply to us. They are all based on matter, and there is no matter here."

Less lyrical but equally influential, Negroponte's *Being Digital* (1995) is also structured around the liberation of information from matter. Contrasting the costly and laborious movement of physical goods with "the global movement of weightless bits at the speed of light" (p. 12) leads him to conclude that, in the digital era, "the medium is no longer the message" (p. 61).

One might be tempted to dismiss Barlow's and Negroponte's manifestos as partially guided by an irrational exuberance that has since been tempered by, among other things, the crash of the dot-com economy at the turn of the millennium. Yet, for a number of influential scholars, the immateriality of digital information continues to serve as the conceptual linchpin for their analysis of the widespread impact of information technologies.

It is central, for example, to Viktor Mayer-Schönberger's recent and widely discussed essay on the value of forgetting, *Delete* (2009). He argues the negligible cost of preserving and accessing information threatens to usher in an era of "perfect remembering," with dire consequences for the fundamental human cognitive process of forgetting. The argument is predicated on an analysis of digital information as superior to all previous media "because it lacks the noise problem" (p. 57). This purported ability to escape the decay wrought by use, reproduction, or time strongly shapes Mayer-Schönberger's subsequent analysis of potential remedies to the end of forgetting (Blanchette, 2011).

It is also central to *Blown to Bits: Your Life, Liberty and Happiness after the Digital Explosion*, a widely praised guide to the information age co-written by MIT's Hal Abelson, Harvard's Harry Lewis, and Ken Ledeen (2008). Again, the analysis is structured around binary encoding as the ground zero of information representation: "Bits are bits, whether they represent movies, payrolls, expletives, or poems. Bits are bits, whether they are moved as electrons in copper wire, light pulses in glass fiber, or modulations in radio waves. Bits are bits, whether they are stored in gigantic data warehouses, on DVDs sent through the mail, or on flash drives on keychains" (p. 294). The imperviousness of bits to their material embodiment is highly significant for designing appropriate information policies. For the first time in history, we are in a position to enact regulations that do not depend on

DOI: 10.1002/asi

the historical accretions that have heretofore bound together media and content:

> "Law and policies regulating information developed around the technologies in which that information was embodied. The digital explosion has reduced all information to its lowest common denominator, sequences of 0 and 1s. . . . The universality of bits gives mankind a rare opportunity. We are in a position to decide on an overarching view of information. We can be bound in the future by first principles, not historical contingencies" (p. 294).

In certain areas of professional practice, the question of digital materiality will play a fundamental role in delineating the shape of things to come. While records managers must deal with the shift to electronic documents, the rules that govern the admissibility and weighing of documentary evidence in courts were largely designed around the technologies of the printed world—paper, ink, handwritten signatures, stamps, etc., with the consequence that long-standing evidential concepts of authorship, originals, integrity seem altogether inapplicable to the world of digital records. In a recent treatise on the question, George Paul (2009) argues that reform must necessarily proceed from the recognition that electronic documents are made from entirely new stuff:

> "Writings in the digital realm are different. They do not depend on the alteration of matter. Such records are very close to 'pure information,' and exist by virtue of a mere succession of the differentiation of 1s and 0s, distinguished by electricity flowing in machine systems. In writing today we deal in pure information objects, unfettered by matter. They can be whisked or shaken or rearranged in an instant" (p. 19).

This immateriality implies that entirely new methods for establishing authenticity must be deployed: "Because digital records do not depend on the alteration of matter, a process of inspecting them is not a reliable paradigm for testing authenticity" (p. 21). The appropriate paradigm, Paul argues, is one based on mathematical algorithms, cryptographic digital signatures, which will offer new (and superior) guarantees for integrity and authorship.

The above examples thus suggest that the trope of immateriality is more than a convenient metaphor information age pundits reach for to cut through technical complexity. Rather, it clearly plays a central role in several important arguments over the implications of our current society-wide shift to digital information. In fact, Hayles (1999) argues it is fundamental to the project of posthumanity, a worldview that informs and is articulated within the various scientific disciplines and literary genres that claim cybernetics as their intellectual ground—including artificial intelligence, robotics, artificial life, science-fiction, etc. At the heart of this project lies a fundamental assumption, that informational patterns (including human consciousness) are ontologically superior to their (accidental) material instantiations (including the human body); a promise, that information "can be free from the material constraints that govern the material world" (p. 13); and a vision, the implication that "if we can become the information we have constructed, we can achieve effective immortality" (p. 13). Digital information systems provide a particular valence to this project, given that

> ". . . reality at a fundamental level is seen as form rather than matter, specifically as informational code whose essence lies in a binary choice rather than material substrate. . . . The assumption that form occupies a foundational position relative to matter is especially easy to make with information technologies, since information is defined in theoretic terms . . . as a probability function and thus as a pattern or form rather than as a materially substantiated entity" (pp. 232–232).

By all measures, then, philosophical commitments to immateriality should not be underestimated. But even if a critical exercise were to corral the rhetorical efficacy of such a position, what alternative models exist? If digital information is not immaterial, in what ways is it material? What relevant physical constraints should a theoretical model of the materiality of digital information capture? Understandably, it is only recently that scholars have begun seriously investigating analytical frameworks that might provide appropriate answers to these questions.

Information, Material

A direct consequence of the prevalence of the trope of immateriality is the dearth of research on the topic, and it is only recently that researchers have self-identified as exploring the materiality of digital information. Several of these researchers have been inspired by previous work in the field of descriptive bibliography, work that sought to link the material conditions of the production, expression, and reception of printed materials to their production of semantic meaning (McKenzie, 1985/1999). For example, Drucker (2009a) notes "the stripping away of material information when a document is stored in binary form is not a move from material to immaterial form, but from one material condition to another" (p. 147). The task then is to map how the particular material condition of electronic media makes possible or impossible new potentialities for reading. In a similar vein, Hayles (2002) has explored the theme of materiality as manifested in electronic literature, arguing that literature has traditionally conceived of the body of the book, of the writer, and of the reader in terms of "assumptions specific to print," and that electronic media brings them together in new configurations, providing us with "an opportunity to see print with new eyes."

Knoespel and Zhu (2008) suggest the popular characterization of cyberspace as "an ethereal escape from the filthy, hopeless 'meat' world" is inherited from a Cartesian dualism that posits a strict dichotomy between language (spirit) and the material world. Moving beyond such "romantic notions of immateriality," they suggest computing systems are characterized by a "continuous materiality,"

> ". . . a wide spectrum of materiality activated by a hierarchy of codes that moves from 'lower' machine code to 'higher' readable computer languages and to codes in general (structural,

DOI: 10.1002/asi

legislative, social, cultural, etc.). Each level of code engages natural language and the physical world in different ways, varying from the shifting voltage of computer circuits to our everyday activity. Altogether, the hierarchy of codes constructs a field of diverse materiality that is continuous and interconnected" (p. 236).

Continuous materiality accounts for the materiality of computing on several levels: through the immanence of embodied experience in language, manifested by the dual registers through which code operates. Instructions to machines (open window, cut and paste) are also apprehended by humans via the metaphorical function of language. Even while programmers mostly operate within strictly positivists conceptions of language, computer code creates relationships among multiple symbolic systems, those necessary to move the cogs of the machine, and those necessary for those operations of the machine to be situated within language, and thus, social order. At the same time, multiple kinds of computer code coexist within the computer, each potentially mediating among different codes pertinent to different social systems.

In a similar vein, Warner (2009) has argued that the linguistic concepts of syntagm and paradigm, and the information theoretic concepts of message and messages for selection are derived from a common material basis, that of the line and the surface. He suggests that "understanding the material basis for concepts from linguistics and information theory, and locating them precisely in relation to current material realities, might then yield a basis for a fuller understanding of the effects of computational procedures, themselves constrained by a common and inherited material reality" (p. 198).

Kirschenbaum (2008) has offered the most original and sustained investigation of the physical constraints that obtain on digital media, through his extensive analysis of the mundane, the ubiquitous, and yet opaque and mysterious hard drive, the inscription workhorse of the computing age, and yet, until Kirschenbaum, a device bereft of sustained analysis.

His first line of attack concerns the curious discrepancy existing between the literary critics' view of electronic writing as ephemeral, fundamentally unstable, forever malleable and that of computer forensics experts, whose livelihood is predicated on the recovery of the numerous traces digital objects leave behind, even after their presumed deletion. The confrontation rapidly exposes the influence of a certain "media ideology of electronic text . . . the notion that in place of inscription, mechanism, sweat of the brow (or its mechanical equivalent, steam), and cramp of the hand, there is light, reason, and energy unleashed in the electrical empyrean" (p. 39). Kirschenbaum's project then is to define "an approach capable of accounting for the ways in which electronic data was simultaneously perceived as evanescent and ephemeral in some quarters, and remarkably, stubbornly, perniciously stable and persistent in others" (p. 27). His answer rests on the distinction between two types of digital materiality, "forensic" and "formal."

Institutions with highly detailed protocols for controlling the creation, access, and eventual disposal of sensitive, classified information have been long aware that deleting digital information from hard drives requires more than simply moving it to the trash icon. Various methods and procedures have been developed to combat the phenomenon of "data remanence," the residues left behind by the physical processes used to write and erase digital data on electronic storage media—from overwriting to media destruction. Because the performance of these processes varies from one inscription/deletion to the next—due to variations in the magnetic substrate and the exact positioning of the read/write head—earlier data may still be accessed in the form of an "erase band" along the edge of magnetic track. Thus, in ways that points to its shared condition with other media, the storage of digital information exhibits specific constraints on "reversing or obscuring what are tangible interventions in a physical medium" (p. 60). "Forensic materiality" thus captures the application of the principle of individualization, "the idea that no two things in the physical world are exactly alike" (p. 10) to digital storage. As Kirschenbaum points out, "that the scale here is measured in mere microns does not change the fact that data recording in magnetic media is finally and fundamentally a forensically individualized process" (p. 63). And it should come as no surprise that the social adoption of a new writing technology gives birth to "an eruption of tools and techniques to fix, expunge, and recover their meaning-bearing marks and traces" (p. 71).

Kirschenbaum's concept of "formal materiality" encompasses two different dimensions. The first suggests one possible answer to the question of how digital writing so compelled academics to uncritically characterize it as free from the material. Borrowing an insight from Daniel Hillis (1999), Kirschenbaum notes how computers' ability to continually perform error-correction enables them to present digital information as "noiseless." As he notes, "computers are unique in the history of writing technologies in that they present a premeditated material environment built and engineered to propagate an illusion of immateriality" (p. 135). The second dimension of formal materiality points to file formats and the structuration they impose on digital data as powerful constraints on mutability of bits—for example, in the case of JPEG images, different levels of compression result in images perceptually indistinguishable, but from which some information has been irretrievably lost. Similarly, the encoding of data in a file format enables or disables specific kinds of computational manipulation—e.g., a TIFF image of a document does not support search in same way a text file will. Thus, despite the flexibility and mutability of digital information, "the play of code is *not* always infinitely fungible and arbitrary—transformations are not always reversible, nor are all transformations always possible and achievable" (p. 149).

In spite of these insights, scholars still find it difficult to characterize the digital in material terms. For example, Leonardi (2010) notes the material properties of artifacts are those that enable and constrains them in ways that "simply cannot be overcome," e.g., the opacity of wood. Proceeding from the premise that "a digital technology like a word processing program is an artifact that is not comprised of matter,"

DOI: 10.1002/asi

he then concludes that "moving away from linking materiality to notions of physical substance or matter may helps scholars of technology integrate their work more centrally with studies of discourse, routine, institutions and other phenomena that lie at the core of . . . social theory, more broadly."

Building on the works outlined above, I propose in the following sections an analytical framework that may in fact integrate digital materiality with a broad range of social scientific disciplines. The primary mechanism that mediates and structures this materiality is the design strategy of modularity.

Modularity and Layering

Information systems can be divided into three major types of components: applications that provide services to users, usually according to some task model or metaphor (e.g., "the desktop," "word processing," "show slides"); infrastructure software that mediates applications' access to shared computing resources, and the physical devices that provide processing power, storage, networking. Infrastructure software may be located in operating systems on commodity computing devices, embedded in hardware (e.g., firmware), or execute on specialized computers (Web servers, routers, etc.). The interoperability of applications, infrastructural software, and devices is an extraordinary engineering achievement. The sending of a simple email over the Internet requires the correct functioning of thousands upon thousands of heterogeneous material and logical components, connected together in a network of staggering complexity. Such a system must be able to accommodate, among other things, growth in size and traffic, technical evolution and decay, diversity of implementations, integration of new services to answer unanticipated needs, emergent behaviors, etc. The solution adopted by the software and hardware industry to manage this complexity is the design strategy of modularity, a strategy with widespread application in manufacturing (from automobile to disposable razors), architecture, and education (curriculum design).

Modularity is a strategy for designing the architecture of an artifact, in particular, the relationship of its function to its structure (Ulrich, 2007). The design of a disposable blade safety razor, for example, realizes two distinct functional requirements, cutting hair, and hand manipulation.[1] Safety razors are typically structured in two separate components (or modules), the blade and the handle, each implementing a distinct functional requirement. The blade and handle components are decoupled, insofar as a change in one component (gradual wear of the blade) will not result in a complete breakdown of the artifact, since it can be replaced. A modular architecture is one that realizes a one-to-one mapping between functional requirements and components, as well as decoupled interfaces between those components (Ulrich, 2007).

Such separation of functional specification from implementation has multiple advantages for computing systems design. As early as 1959, McGee noted that pressures to extract maximum value from expensive data processing equipment led programmers to "hand-tailor their programs," rather than developing more general techniques. This resulted in "first of all, a prodigious outlay of programming time; and secondly, a running program which is 'chiseled in granite' and which effectively defies any attempts to modify it at a later date" (McGee, 1959). 25 years later, Parnas argued that modular design provided just the solution to this vexing issue:

> ". . . it should be possible to make a major software change as a set of independent changes to individual modules, i.e., except for interface changes, programmers changing the individual modules should not need to communicate. If the interfaces of the modules are not revised, it should be possible to run and test any combination of old and new module versions" (Parnas, Clements, & Weiss, 1984, p. 409).

In addition to providing a strategy for managing change, modularity also reduces system complexity by division of labor: modules can be assigned to different teams, each module small enough to be fully comprehend by a single individual (Blaauw & Brooks, 1997). The working of modularity is plainly visible when it comes to the widely different hardware components that can connected to computer systems through a single peripheral interface specification—e.g., USB or SCSI. Such an interface specifies both the services which the particular device must provide (e.g., storage and retrieval of bits, status information, etc.) and the software and hardware language necessary to interact with the module (e.g., connector pins assignment, with corresponding control signals).

Layering is a specific flavor of modularity where modules are organized in a series of client-server relationships: each layer is a server to the layer above, and a client to the layer below. While the best-known example of layering in software infrastructure is the famous 7-layers deep "network stack" defined by the ISO OSI Model (Zimmermann, 1980), each computing resource (i.e., network, storage, and processing) is accessed through a similar stack of layers. In each case, bits move up from their grounding as signals in some physical media (fiber optic, magnetic drive, electrical wires) to binary information organized according to units defined by each layer (file, datagram, etc.) Applications access the stacks through "application programming interfaces" (APIs) to the various modules of the operating system.

Trade-offs

The plug-and-play possibilities that modularity bring to systems design are often remarked on—it is for example one of five essential principles of new media identified by Manovitch (2007), and it is at the core of Zittrain's (2009) analysis of the "generative architecture" of the (early) Internet. That these possibilities must be understood together with the particular constraints modular designs bring to the table is, however, rarely remarked on. As McGee remarked, the most efficient programs are hand-tailored, providing

[1] One should note that the assignment of functional elements to artifacts is far from a well-defined issue; see Preston (1998, 2000).

DOI: 10.1002/asi

no generalization whatsoever; conversely, highly general abstractions will result in significant loss in efficiency. This is because the specification of an abstraction (the interface) general enough to accommodate a wide range of implementations necessarily involves trade-offs, "between the freedom that the abstraction provides and the efficiency of possible implementation" (Agre, 1997).

A simple example can help illustrate this concept. Consider the problem of organizing a closet full of disparate objects—e.g., sporting equipment, children toys, craft supplies, clothing. If the primary goal is to pack as many objects in the closet as possible, the best approach is to pack based solely on objects' size and shape, using the closet itself as a box. A much more practical solution, however, will use a widely available modular structure, storage bins, and group objects by categories, filling and stacking as many bins as will fit. In contrast with the first solution, packing, locating, and retrieving objects is greatly simplified, but this convenience comes at the expense of overall density: bins will be more or less full, and they will fit more or less snuggly in the closet itself. Different bin sizes, as well as different types of objects, will result in different space inefficiencies.[2]

This is the classical dilemma of high-level programming languages: the more a language's constructs abstract away from the underlying physical machine, the less efficient the resulting code tends to be. For example, functional languages (e.g., Lisp) relieve programmers from the burden of requesting and releasing memory locations for variables, allowing them to proceed as if memory was an inexhaustible resource. But memory is, in fact, always a finite resource, and instead of manual management by users, "garbage collection" routines must reclaim obsolete memory locations, a process that itself consumes processing power, as it seeks to reconstruct after-the-fact the memory space allocated and de-allocated by the programmer. The programming convenience of a boundless memory is thus incurred at the cost of processing resources. This makes garbage collection particularly inappropriate for real-time applications (e.g., software that implements antilock brakes), given the routine may request processing power at a crucial moment.[3] The point here is that the trade-offs implied by modularity will not affect all applications equally, or even the same application under all circumstances. Yet, the design trade-offs inherent in abstracting from physical resources are rarely acknowledged in the computing literature.

The Digital Abstraction

These trade-offs manifest themselves all the way down to the lowest level of the stacks, the physical layer. Agre (1997) notes the most fundamental abstraction computers rely on is the "digital abstraction," the transformation of physical signals into discrete binary quantities. From Tinker Toys to hydraulic valves, as long as a material can support the basic operations of the digital abstraction, it can be used as the basis for a computing system (Hillis, 1999). However, each of these materials brings its own characteristics to the performance of these operations, including susceptibility to interference, frequency of mechanical failure, relative lack of speed, resistance and attenuation, and, of course, cost.

The digital abstraction can be maintained in spite of this "noise" because, as Kirschenbaum notes, through error-correction codes, buffering, and other techniques, computers can self-efface the static—scratches on a record, smudges on paper—that typically signals the materiality of media:

> "All forms of modern digital technology incorporate hyper-redundant error-checking routines that serve to sustain an illusion of immateriality by detecting error and correcting it, reviving the quality of the signal, like old-fashioned telegraph relays, such that any degradation suffered during a subsequent interval of transmission will not fall beyond whatever tolerances of symbolic integrity exist past which the original value of the signal (or identity of the symbol) cannot be reconstituted" (p. 12).

These mechanisms, formally described in information theory, are used throughout networked computing systems: the impact of media irregularities on hard drive platters can be mitigated through the use of error-correction codes; the unpredictability of network bandwidth can be mitigated through the use of buffering, ensuring smooth delivery of latency-sensitive content—Hillis (1999) calls this "the essence of digital technology, which restores signal to near perfection at every stage" (p. 18). It is this ability to ceasessly cleanup after its own noise that so powerfully enables computers to seemingly sever their dependency on physical processes that underlie processing, storage, and connectivity.

Yet the physical characteristics of a resource (be it computation, storage, or networking) cannot simply be transcended, and noise can only be conquered at the expense of other resources. For example, manufacturers must design electronic circuits using a voltage differential between 0 and 1 broad enough to fight off interference by galactic cosmic rays ("single event effects"), at the cost of increased power consumption (May & Woods, 1979); error-correcting codes, widely used to protect against transmission interference, result in both data expansion (and thus, reduced capacity) and increased processing load. In the latter case, designers will choose among different codes according to both the expected profile of the noise (frequency, intensity), and the resource trade-offs. Once again, then, independence from the material can only be obtained at the costs of certain trade-offs.

Sharing

Computing resources (processor, storage, network) are not only finite, but to maximize their efficiency and return

[2]This very trade-off was at the heart of the design of the most widespread modular structure in the world—the shipping container. See Levinson (2006), in particular Chapter 7, "Setting the Standard."

[3]In fact, the most efficient garbage collection algorithms are referred to as "stop the world" algorithms, as they require the executing program to stop altogether during the memory reclamation process, leading to an "embarrassing pause" (Henriksson, 1998).

DOI: 10.1002/asi

on investment must be shared among multiple applications and users. Thus, abstractions not only relieve programmers from the need to manage the finiteness of resources, but also from the need to manage how they are shared with other applications, competing for their share of limited processing power, memory, bandwidth, storage. This is not only for the purpose of programming convenience, but because policies for sharing must implemented at the system (rather than application or user) level. Once more, this will inevitably involve various trade-offs, favoring some types of applications over others. For example, packet switching protocols maximize the utilization and sharing of finite communication links by breaking down users' messages in small packets, and routing them to their destination using a "best-effort" policy that impacts unevenly latency-sensitive (voice, streaming video) and latency-insensitive (browsing, email) applications.

Stack Equilibrium

Two opposing forces are thus at play with respect to the make-up of the stack that obtain at any particular moment of transition in technical history: on the one hand, the freedom provided by modular design and the resulting efficiency trade-offs; on the other hand, the primary drive of computing systems design, greater efficiency, as measured by "the amount of useful computational work that gets done in the service of specified goals by a given amount of machinery in a given period of time" (Agre, 1997, p. 67). It is the conflicting pressures of these two forces that determine the evolution of the layered abstractions that link digital information to its material basis.

In the next three sections I illustrate the operation of these forces by tracing the historical definition of major abstractions within the processing, storage, and networking stacks, and their evolution as they respond to changes in the material basis of computing resources. In particular, I will highlight how the drive to efficiency manifests itself as the pressure to co-design layers, thus violating the fundamental principle of modular independence itself.

The Processing Stack

A processor, or central processing unit (CPU), contains circuit logic designed to execute programs, i.e., sequences of instructions. These instructions enable a programmer to access three basic set of resources: (a) numerical routines, typically provided by the arithmetic-logic unit (ALU); (b) memory management services, i.e., reserving, storing to, and reading from memory locations; and (c) flow control, i.e., selecting the next instruction to be executed, based on conditional branching, jumps, etc.

Each processor (or processor family) provides its own set of instructions, each directly operating on the processor's hardware by performing the necessary sequences of logical operations (opening and closing gates, moving data to and from memory, etc.) to produce the appropriate result. The set of instructions of a processor is its machine language, and provides the interface to the processor conceived as a module. The computational model expressed by an instruction set is referred to as its "instruction set architecture" (ISA).

The designers of a processor and its accompanying machine language must contend with the fundamental trade-offs between convenience of instructions to programmers and efficiency of implementation. That is, in machine language, "the expressions are costly, ... each operator and variable in the vocabulary must be implemented and realized by the interpreting mechanisms. Each bit in a machine-language program occupies a costly memory cell and must be obtained from that cell at the expense of costly time" (Blaauw & Brooks, 1997, p. 17). The computer architects thus evaluates each expression of the machine language against their "bit budget," the amount of memory locations they have to work with, as well as the "bit traffic" each expression will generate.

While early computers were always programmed directly in machine language, the difficulty of writing and debugging machine-level code of increasing difficulty generated interest in the development of "high-level" programming languages that would provide more readable notations for specifying instructions, leading to greater efficiencies in program development and tuning. Before they can be executed by a processor, programs written using high-level programming languages must first be processed by a compiler, a program which takes a program in a source language and translates it into an equivalent program in the machine language of a given processor. A program in machine language is "that representation of programs that resides in memory and is interpreted (executed) directly by the hardware" (Blaauw & Brooks, 1997, p. 16).

High-level programming languages provide several services for programmers that simplify access to and use of the basic resources of the processor, such as automatic memory management, data type checking, and enforcement. They also extend the underlying computational model of the machine, by providing for the creation of new data types and operators, broader ranges of control (e.g., recursion), and the ability for programmers to create their own abstractions (functions, objects). In other words, "a language rebuilds the machine to provide more convenient facilities, and a program further rebuilds the language to provide facilities closer to the problem to be solved" (Sethi, 1996, p. 11). The computational model implemented by programming language thus defines "virtual machines" that run on the basic physical hardware implemented by the processor. The basic trade-off for this convenience is one of efficiency, that is, the code generated automatically by a compiler typically takes longer to run and occupies more space than hand-crafted machine language code. Despite this, most programmers today resort to high-level languages, and compilers designers are tasked with reconciling the abstractions offered by a language's computational model, and its implementation in the machine language of the underlying machine. However, the independence of these two layers of abstraction—machine versus programming language—is under constant pressure.

DOI: 10.1002/asi

Co-evolution of Layers

In certain cases, the abstractions defined by high-level languages may lead to design decisions at the level of implementation, in top-down fashion. In the 1980s, the RISC chip design revolution proceeded in part from the observation that the convenience of the abstractions provided by high-level languages had become expected by programmers:

> "... instruction sets for conventional CPUs have been defined with an implicit assumption that many programmers will use assembly language.... But, increasingly, programmers do not use assembly language, except where optimal performance is essential or machine functions are required that are not reflected in the source language" (Radin, 1983, p. 40).

Given this, it made sense to design chips that directly implemented high-level abstractions with improved efficiency.[4] In 1980, IBM experimented with the design of a minicomputer whose machine language was co-designed with an optimizing compiler for the PL.8 language (a subset of PL/1). In similar fashion, from the 1970s through the 1990s, several generations of machines providing hardware support to run Lisp programs more efficiently were developed for the AI community by both startups and established computer manufacturers, including Symbolics, Xerox, and a Texas Instruments/Apple partnership (Pleszkun & Thazhuthaveetil, 1987). Other machines were developed to support the object-oriented strategies of Smalltalk (e.g., Ungar, Blau, Foley, Samples, & Patterson, 1984). As viable commercial products, all succumbed to the rise of commodity personal computers, whose cost/performance ratio negated much of the commercial rationale of these efforts.

Another type of specialized processor design, parallel architectures, has also suffered a long history of commercial failures, despite the dazzling promise of increasing processing power by several orders of magnitude. Parallel architectures altogether eschew the von Neumann model of serial computation (first proposed in 1945 in the context of the EDVAC), which forces algorithmic design through the very narrow funnel of sequential programming. Reducing all problems to sequences of atomic instructions has proved enormously convenient for programmers, but has resulted in serious design constraints: because of this "von Neumann bottleneck," much of a conventional processor's circuitry remains inactive at any one moment, often waiting on the much slower memory subsystem. Thus, speed increases over the last 40 years have been predicated on a strategy of increasing clocking (the speed at which instructions are processed) and transistor density:

> "...the implicit hardware/software contract was that increased transistor count and power dissipation were OK, as long as architects maintained the existing sequential programming model. This contract led to innovations that were inefficient in terms of transistors and power . . . but that increased performance while preserving the sequential programming model" (Asanovic et al., 2009, p. 56).

This contract has become unsustainable, as chip designers have now reached the "power wall," i.e., physical limitations on the ability of transistors to dissipate heat efficiently (resulting in burning hot laptops!). In 2004, Intel announced all future product designs would be based on multicore architectures, the packing of multiple processors on a single chip assembly. The decision signaled a turning point in the evolution of computing: "the La-Z Boy era of program performance is officially over, and programmers who care about performance must get up off their recliners and start making their programs parallel" (Patterson, 2010, p. 32).

This will require more than a simple motivational exercise: by all accounts, breaking the dominance of the von Neuman model is as formidable a challenge as the computing professions have ever faced, a profound break with existing programming practice. It will require a greater commitment to parallel methods in the computer science curriculum, but just as important, it will require the development of new abstractions that will shield programmers from the inherent complexity of parallel programming. Furthermore, this new stack of abstractions will face the difficult task of simultaneously supporting applications that leverage the power of multicore architectures, while still ensuring that "legacy code still works with acceptable performance" (Asanovic et al., 2009, p. 59).

These pressures on the evolution of the processing stack illustrate the tension between the freedom afforded by modularity, and the inefficiencies that it necessarily brings into play: in this case, a single modular design, the von Neumann model, ruled the stack for over 60 years. This dominance resulted in economies of scale that defeated repeated attempts at creating a viable market for alternative, parallel architectures, despite their promise for increased processing power. The persistence of the model was further enabled by a computer science curriculum committed to the convenience afforded by sequential programming. Yet changes in the material basis of computing resource necessarily ripple up the stack, as exemplified by efforts to design new programming abstractions for parallel architectures.

The Storage Stack

In contrast to the electronic components that make up the processor, the media leveraged over the years to store and access data—punch cards, magnetic tapes, hard drives, flash memory—profusely signify their materiality, through mechanical noise, slow speed, poor reliability, and sensitivity to wear. The abstractions that make up the storage stack must thus provide consistent services to applications in the context of wide discrepancies in the performance characteristics of storage technologies. The defining characteristic of these devices is their reliance on mechanical motion. From punched card to magnetic tape to disk drives, large-scale external storage has been realized by the physical movement

[4] See also Hopper and Mauchly (1953) for an early argument for co-design of hardware and programming languages.

DOI: 10.1002/asi

of data, spread over one-dimensional (tape), two-dimensional (floppies), or three-dimensional surfaces (hard drives).

Overall, the use of mechanical motion, however finely controlled, does not sit comfortably with the world of solid-state electronics. The most important friction is the huge differential in access time between internal and external memory. In contrast to the movement of bits in strictly electronic hardware, the reading and writing of bits on external media is extraordinarily slow, from four to six orders of magnitude slower! Thus, in applications that process large amounts of data, fetching and writing data to and from external memory is often the main performance bottleneck in computation. As Blaauw and Brooks (1997) note,

> "Four orders of magnitude is an immense ratio. Imagine a CPU doing an operation each second; a disk half-rotation 10^4 slower takes three hours! . . . While a disk turns half around or a tape accelerates to reading speed, a workstation CPU can execute perhaps 100,000 instructions" (p. 453).

This differential has been an important constraint from the very first days of digital computer design. In the 1950s, engineers experimented with a wide range of media and technologies as potential candidates for both internal and external storage: in the first case, these included Williams tubes, mercury delay lines, and magnetic drums, etc. (Eckert, 1953); in the second case, magnetic tape, photographic film, paper tape, magnetic wire, and magnetic drums (Snyder, 1952). While speed was a primary consideration, design choices were, as Eckert pointed out, "strongly influenced by the cost of achieving that speed, and by the requirements of the other circuits" (p. 1393).

Blocks are one structure that aims to reduce the impact of the speed differential. The block size of a device is typically the amount of data transferred to/from the device in a single operation:

> "Access varies greatly because of medium motion. Finding an arbitrary bit may take quite a long while. Finding the next bit on the track is very quick—a fraction of a microsecond. Therefore, if there is the slightest chance . . . of needing the next datum after finding the one sought, one is well-advised to read it also into memory. This logic leads inescapably to the reading and writing of data in blocks whose size is limited chiefly by the cost and availability of memory space. If it takes a long time to go to the well, one should bring back as much water as the bucket will hold" (Blaauw & Brooks, 1997, pp. 453–454).

The trade-off here is that because the block, rather than the bit, becomes the fundamental unit for reading and writing to storage, results in significant amounts of wasted space for applications that generate large quantities of small files.

Furthermore, storage devices strive to efficiently provide two fundamental and somewhat contradictory objectives: providing the highest throughput possible in reading and writing sequential streams of bits, and minimizing the time it takes to locate a particular datum on the media—so-called "random access" (Buchholz, 1963, p. 91). To optimize both of these features requires carefully balancing the structures that govern the placement of the data over the media (e.g., tracks, cylinders) with the mechanisms that govern the motion of data.

The File System Interface

The primary abstraction that governs the relationship between applications and storage device is the file. The story of this abstraction must begin with the prominent role played by punch-card equipment in data processing until the late 1950s, a role now recognized in several studies (e.g., Campbell-Kelly, 1990; Yates, 2005). Punch cards served as input and output media, as well as long-term storage to the tabulating system. One punch card and its various data "fields" constituted a "record," while a collection of cards constituted a "file" (Haigh, 2009).

The development of the electronic computers that would eventually replace tabulators was a gradual process, in which the issue of storage media played a defining role. Not only did the efficiency and cost of computers depend to a great extent on suitable techniques for fast internal direct access memories, but their ability to integrate with existing input, output, and storage technologies (i.e., punch cards) was a crucial factor in ensuring their adoption. A computer such as the IBM 650, the world's first mass-produced computer, remaining in production from 1953 to 1969, encompassed the full range of memory devices and input-output technologies available, including a disk drive, a magnetic tape drive, units for reading and punching cards, as well as tape-to-card and card-to-tape conversion units. Because of this extensive variety of storage devices, by 1957 IBM designers "were already distinguishing between logical and physical aspects of data storage in tape files, a key concept for the decoupling of application programs from specific hardware configuration and file formats" (Haigh, 2009, p. 10).

The designers of the UNIVAC, one of the 650's main competitors at the time, chose to make reliance on magnetic tape a distinguishing feature of their system. Indeed, "for most customers, what was revolutionary about the UNIVAC was not so much its stored-program design or even its electronic processor, it was the use of tape instead of punched cards" (Ceruzzi, 2003, p. 30). Yet, the logical organization of the information was directly copied from that used in tabulating systems:

> "The concepts of records, files, fields, special codes to mark the beginning and end of files, and the merging information from one file to another (all ubiquitous in computer systems today) have their origins in electromechanical punched card machine methods dating back to the 1930s. Records using the same basic format were laid out sequentially along the strip of magnetic tape. Additional codes were introduced to provide checks against corrupted data" (Haigh, 2009, p. 7).

The influential time multics operating system that emerged in the early 1960s to provide time-sharing for mainframes included a "general-purpose" file system for external storage.

DOI: 10.1002/asi

Its design included features still widespread today, including hierarchical directories, symbolic links (i.e., aliases), and access control. MULTICS' designers clearly saw the role of the file system as insulating users (i.e., programs) from the complexity of storage:

> "In most cases a user does not need to know how or where a file is stored by the file system. A user's primary concern is that the file be readily available to him when he needs it. In general, only the file system knows on which device a file resides. The file system is designed to accommodate any configuration of secondary storage devices. These devices may cover a wide range of speeds and capacities. All considerations of speed and efficiency of storage devices are left to the file system. Thus all user programs and all other systems programs are independent of the particular configuration of secondary storage" (Daley & Neumann, 1965, p. 222).

Ritchie and Thomson's design for Unix envisioned an even more prominent role for files, stating, "the most important job of Unix is to provide a file system" (Ritchie & Thompson, 1974, p. 366). In fact, the file is the dominant abstraction Unix provides to programmers for any input-output device, from paper tape to hard drives. Ritchie and Thompson proposed a new data structure, the "inode," that would release the file system from the need to determine in advance how large a disk file might eventually grow and improve the dynamic sharing of storage space among users. An inode is an extensible tree structure that provides an index to the disk location of the blocks containing the file data. As the file grows, such "direct blocks" are replaced by "indirect blocks" that, instead of data, contain the disk location of the direct blocks.

Such a dynamic data structure provides enormous flexibility to the file system—it may, for example, grow the file as large as needed, and rapidly reclaim disk space for other users as the file shrinks. The (significant) downside is that as different users and applications create, expand, shrink, and delete files on shared storage, the contents of a file become randomly distributed over the storage device, with consequential negative impact over sequential access. Subsequent implementations of the original system have corrected for this problem by increasing the block size (at the cost of increased wasted space) and optimizing the sequential placement of blocks, by trying "to allocate new blocks on the same cylinder as the previous block in the same file" (McKusick, Joy, Leffler, & Fabry, 1984, p. 188). There is thus content tension between the freedom provided by a system abstraction (grow the file as needed), and the inefficiencies it introduces with respect to the spatial placement of data.

The syntax and behavior of the Unix file system has been standardized as part of the IEEE POSIX process, providing a uniform interface to the various services its provides—file creation, deletion, reading, writing, seeking, etc. It is this standard interface that provides the glue for the design of the Google File System (GFS) suitable for the processing needs of its software engineers (i.e., working with files typically in the multi-GB range) and for a computing environment consisting of "hundreds or even thousands of storage machines built from inexpensive commodity parts" (Ghemawat, Gobioff, & Leung, 2003, p. 29). The main data structure of the GFS is a "super inode" that sits above the Unix file system. Like a regular inode, it contains pointers to either direct or indirect blocks, with the distinction that these blocks are ordinary Unix files, which may be located on any number of drives. The Unix file system abstraction is thus "encapsulated" by the larger abstraction defined by the GFS, with quite different parameters of course—the block size is a whopping 64 MB, over the 8K common in Unix implementations. In designing the GFS, Google engineers enjoyed a considerable advantage: the freedom to breach the independence of the application and file system layers, a freedom that otherwise rarely obtains in software infrastructure design:

> "One thing that helped tremendously was that Google built not only the file system but also all of the applications running on top of it. While adjustments were continually made in GFS to make it more accommodating to all the new use cases, the applications themselves were also developed with the various strengths and weaknesses of GFS in mind. . . . We could push problems back and forth between the application space and the file-system space, and then work out accommodations between the two" (McKusick & Quinlan, 2009, p. 45).

Such accommodations are detailed in a discussion of MapReduce, Google's in-house parallel programming environment: "We conserve network bandwidth by taking advantage of the fact that the input data (managed by GFS) is stored on the local disks of the machines that make up our cluster. . . . When running large MapReduce operations on a significant fraction of the workers in a cluster, most input data is read locally and consumes no network bandwidth" (Dean & Ghemawat, 2008, p. 110). That is, by breaching the independence of the GFS layer, MapReduce can thwart the potentially massive inefficiencies of a highly parallel environment. Thus, even in the context of Google's massively distributed data processing centers, the issues remain the same: how to reconcile a powerful abstraction that provides considerable convenience to programmers with the need to optimize the spatial organization of the data for particular types of processing.

The Network Stack

The network stack provides applications with services ensuring the error-free transmission of structured bits from one computer to another with the highest throughput (capacity) and lowest latency (time in system) possible. This must be accomplished in the context of significant material constraints: (a) signals must travel over physical media—whether air, copper wire, or fiber optic—each bringing different characteristics to the job, with regard to susceptibility to interference, dissipation, capacity, and cost; (b) the physical infrastructure necessary to provide point-to-point communication is enormously costly, and consequently driven by particular economic dynamics, including network effects, and economies of scale and density; (c) these costs require

DOI: 10.1002/asi

that communication links to be shared among multiple users, with the corresponding need for fair policies to manage traffic and its attendant inefficiencies.

Materials

Common media for digital communication include twisted-pair (telephone wire), coaxial cable (cable television), fiber optic, and radio waves. Like storage, a chief characteristic of communication media is its unreliability, including attenuation, the gradual weakening of the signal due to the physical resistance of the media to electrical current; and noise, the gradual distortion of the signal by a wide variety of sources, including interference from other wires, radio signals, the physical environment, etc. Because attenuation and noise determine the capacity of the media (Czajkowski, 1999), extensive measures must be deployed to counter their effects, including signal amplifiers and cable shielding.

Additionally, signal processing techniques help ensure the correct transmission of data over an unreliable channel: modulation translates digital data into a form suitable for transmission for a given physical medium, while error-correcting codes use redundancy to protect sequences of bits against noise. Different coding and modulation techniques are appropriate given the specific noise characteristics of the transmission channel and the amount of processing they require. In both cases, the fundamental trade-off is accuracy versus capacity, that is, the more protection against noise, the less data the channel can carry, a trade-off first articulated by Shannon (1949).

Physical Infrastructure

Worldwide, the "twisted pair" copper wires deployed for the provision of telephony represents the most important infrastructural investment for telecommunications providers, millions of wires that connect individual dwellings to the network, built over the course of the last century. A defining engineering project of the early 21st century is the conversion of this "voiceband" infrastructure, limited to the transmission of analog voice, to a "broadband" infrastructure, i.e., one capable of carrying significant amounts of digital information. While the physical properties of optical fiber have made it the preferred material for wireline broadband, the enormous costs of infrastructural deployment to the level of the "last mile" ensure that the Internet will continue to operate over a mixture of physical materials:

> "Although many broadband architectures require optical fiber to be brought at least some of the way towards the home from the exchange, bringing the fiber increasingly closer to the home become prohibitively expensive. The cost of fiber deployment is dominated not by the material costs but by the cost of civil works (i.e., the digging of the road/pavement and the laying of the fiber)—a costs which can range between £25 and in excess of £75 per metre, depending on the circumstances" (Czajkowski, 1999, p. 127).

These costs have significant consequences: first, like capacity on highways, the resource is largely inelastic. Unlike processing power or storage, service providers cannot lay additional cables on demand, and thus, must meet future demand through overprovisioning. The widespread adoption of this strategy by telcos at the height of the dot-com era resulted in vast amounts of "dark fiber," i.e., unused capacity; second, the economies of scale and density that characterize such network infrastructures will continue to offer low incentives for telecommunication providers to update their infrastructure in rural areas; third, telcos have been highly motivated to fund research for the development of encoding and multiplexing schemes that maximizes the amount of bits that could be transmitted over the existing infrastructure (e.g., DSL over twisted pairs). These issues are not limited to wireline communication: much of the wireless (cellular) infrastructure was similarly developed for voice traffic and thus required extensive investments from telcos to handle broadband.

Sharing

As with other point-to-point networks (e.g., telephone, mail, or road system), it is not feasible or cost-effective to establish direct links between every computing device. Instead, multilevel hierarchies of communication links are used, whereby individual links connect to hubs (switchboards, mail sorting centers, highway interchanges, airports), themselves connected to each other through shared, high-capacity links. A central design imperative of network stack is thus the efficient use of shared and scarce communication links, given various characteristics of traffic: peak/off-peak demand, data versus voice, quality of service requirements, shape, etc.

A defining characteristic of the Internet is its reliance on packet switching to maximize limited network resources: first, in the context of computer-generated "bursty traffic," exhibiting substantial variations in intensity, breaking down communications in small packets helps evenly spread usage of a communication link among multiple users competing for the resource; second, depending on network topology, packet switching can help alleviate congestion by moving traffic towards less congested links; finally, the network can adapt in real time to significant changes in topology. Yet, as implemented in the TCP/IP protocols, packet switching entails significant drawbacks: no minimum latency may be guaranteed, a significant issue for applications such as streaming media or telephony.

Cross-Layer Design

The network stack differs significantly from the processor and storage stack, insofar as it is the only case involving the a-priori definition of the modular decomposition of the stack and its imposition through standardization (Abbate, 2000; Zimmermann, 1980). For our purposes, the main elements are as follows: in the 1970s, with several experiments

DOI: 10.1002/asi

under way (including Arpanet), the merging of computing and telecommunications was already on the horizon. In 1978, ISO proposed an overall layering framework for networking technologies, that is, a certain modular decomposition of the network stack. The OSI (for Open Systems Interconnection) model specified how a set of networking protocols should fit together to form a complete system, in effect, a meta-standard that incorporated other standards, and specified their interaction. The model listed seven layers to manage the movement of bits from physical media to applications and back: physical, link, network, transport, session, presentation, application. Each layer would provide for different types of services (e.g., network provides routing, transport provides reliable delivery), and for each layer, multiple standards could be specified.

The mixed successes of OSI and of its top-down design through large standardization bodies have often been contrasted with the nimble political structures that characterize Internet governance (Russell, 2006). Yet, as Dave Clark, former chair of the Internet Architecture Board, explains, the benefits of modularity themselves remain unquestioned:

> "All good computer scientists worship the god of modularity, since modularity brings many benefits, including the all-powerful benefit of not having to understand all parts of a problem at the same time in order to solve it. . . . The field of network protocols is perhaps unique in that the 'proper' modularity has been handed down to us in the form of an international standard: the seven-layer reference model of network protocols from the ISO" (Clark, 1996, p. IX).

However, this 'proper' modularity is currently under stress, as it must adapt to the emergence of a new material basis for networking, that of wireless communication. Along with the dazzling possibilities of mobile computing comes significant engineering challenges: a much larger range of channel conditions, new source of interference, and devices that move more or less randomly through the network infrastructure. And just like the case for wireline communications, the infrastructure for wireless communications must be updated to account for the changing nature of the data it carries—from analog voice to digital data. The wireless infrastructure must account for the wide range of applications pushing and pulling the data through the radio spectrum, all with different Quality of Service (QoS) needs, that is, with different degree of sensitivity to latency: "Emerging networks must support various and changing traffic types with their associated Quality-of-Service requirements as well as networks that may have changing topologies. The problem of various traffic types is typified in newly defined 3G networks. These networks must support multimedia traffic with manifold delay, error-rate, and bandwidth needs" (Rappaport, Annamalai, Buehrer, & Tranter, 2002, p. 158). Yet another emerging wireless infrastructure, that of sensor networks, must compose with the very low power consumption requirements of minuscule wireless devices.

Thus, the very different nature of the physical layer, along with the very different needs which wireless media must satisfy, have led to persistent calls for what amounts to the ultimate taboo in network architecture, breaching the independence of the OSI layers through "cross-layer design":

> "In order to meet the challenges of ubiquitous wireless access, network functions (i.e., the various OSI layers) must be considered together when designing the network. QoS requirements that can and will vary according to application will force the network layer to account for the physical-layer design when optimizing network throughput. Further, different applications are better served by different optimizations. This leads to a design methodology that blurs the lines between layers and attempts to optimize across layer functionality" (Rappaport et al., 2002, p. 159).

Unsurprisingly, from the onset, proposals for cross-layer design have raised no small amount of controversy. The first issue is, quite simply, that modularity has proven such an successful design strategy in computing precisely because it circumscribes the range of interactions between modules (layers). Cross-layer optimization is not different to using "goto" statements, as Kawadia and Kumar (2005) argue, "Does one then get unstructured spaghetti-like code that is hard to maintain?" The wonderful benefits of breaking down complex systems into smaller, more manageable, parts are no longer available:

> "Once the layering is broken, the luxury of designing a protocol in isolation is lost, and the effect of any single design choice on the whole system needs to be considered. . . . Compounding this is the fact that some interactions are not easily foreseen. Cross-layer design can thus potentially work at cross-purposes; the 'law of unintended consequences' can take over if one is not careful, and a negative effect on system performance is possible" (Kawadia & Kumar, 2005, pp. 3–4).

This is not simply a hypothetical situation: Kawadia and Kumar are able to exhibit several examples where proposed cross-layer designs of the MAC and physical layer would result in undesirable interactions with the network layer, resulting in net performance losses under certain conditions. Furthermore, the argument that cross-layer design will improve performance thus very much depends on what dimension of performance is emphasized. As they underline, this is always the fundamental trade-off of modular design:

> "On the other hand, taking an architectural shortcut can often lead to a performance gain. Thus, there is always a fundamental tension between performance and architecture, a temptation to violate the architecture. However, architecture can and should be regarded as performance optimization, although over a longer time horizon. An architecture that allows massive proliferation can lead to very low per-unit cost for a given performance. More properly, therefore, the tension can be ascribed to realizing short-term vs. longer-term gain. In the particular case of wireless networks, which may be on the cusp of massive proliferation, our contention is that the longer-term view of architecture is paramount" (Kawadia & Kumar, 2005, p. 4).

That is, layering enables stable forms of market segmentation (and the resulting economies of scale) to take hold, and

DOI: 10.1002/asi

thus, the "very low per-unit cost" that has been a major element of the success of the telecommunication industry. Thus, while the rationale for cross-layer design is understandable, given the new material basis of wireless communications, it is a recipe for economic (and political) instability, at a particularly sensitive time: "it well behooves us to adopt a cautionary approach to cross-layer design at a critical time in the history of wireless networks when they may well be on the cusp of massive proliferation that is the objective of us all" (Kawadia & Kumar, 2005, p. 11).

Discussion

The preceding historical analysis of the evolution of the three resource stacks demonstrates that the materiality of digital information can be understood as the composition of two different sets of constraints: those due to the physical characteristics of the limited resources of computation; and those due to the adoption of modularity as a means of mediating between these resources and the applications that manipulate this information. At their most fundamental, each of these resources deals with bits as physical quantities, whether magnetic polarities, electric voltages, or radio waves. These physical quantities are first abstracted as bits, and circulated up and down the resource stacks, the layered chains of modules that obtain between applications and resources.

The magic of modularity, its ability to decouple functional specification from implementation, provides enormous freedom and flexibility to the management, coordination, and evolution of complex technical systems. It provides programmers with stable interfaces to system resources in the face of continuously evolving hardware. However, in abstracting from the noise that different materials bring to the digital abstraction, from specific implementations of physical resources, from their distribution in space, and from their sharing among multiple users, such decoupling necessarily involve efficiency trade-offs.

These trade-offs must be continuously negotiated with respect to the fundamental drive of systems design, maximizing the use of systems resources in service of greater efficiency. In particular, this drive continuously works to undermine the freedom afforded by modular decomposition, for example, through attempts at co-design of independent layers. Furthermore, changes to the material resources of computing (e.g., wireline to wireless) necessarily ripple through the layers of the stacks, requiring renegotiation of previously established trade-offs, either through the market or the standardization process.[5] The computing infrastructure thus evolves through the conflicting pressures of forces that encourage a certain kind of persistence, and those that make possible certain kinds of moves.

[5]An excellent account of the techno-politics of standardization is von Burg (2001).

Persistence

Several characteristics of modularity act as conservative forces against the evolving material basis of computing. The chief force is inherent to modular decomposition: while it provides for flexible decoupling of abstractions from implementations, the modular decomposition itself is extremely rigid, since any change involves redefining the relationship of a minimum of two layers. Parkas (1984) foresaw as much when he suggested that "only very unlikely changes should require changes in the interfaces of widely used modules."

Furthermore, particular modular decompositions become embodied in the hardware, software, and institutional infrastructure (e.g., standards, technical training) of computing. Such materialization of particular abstractions provides the long-term stability necessary for economies of scale to take hold. It is such economic advantages that have enabled von Neumann machines to provide the best processing power/cost ratio for the last 60 years—the closest we have to eternity in the world of computing. Emergent forms of materiality threaten existing architectures with economic instability, as stakeholders seek adjustments to the stack to improve its efficiency in favorable ways (e.g., cross-layer design). At the same time, the high costs of infrastructural investments ensure that computing resources are repurposed rather than merely replaced. The resource stacks must thus compose with different types of materials (parallel and nonparallel architectures, twisted pair and fiber, local and remote storage) with resulting trade-offs and inefficiencies. Thus, the solutions developed in a particular moment of technical history tend to persist and accrete.

Thus, in contrast to the perception of computing as moving at a furious pace of technical evolution, its infrastructure evolves very slowly. Because of the need to maintain backward compatibility, the incorporation of major changes in the material basis of computing—e.g., multi-core processing, cloud-based, and wireless computing—proceeds conservatively through mutation and hybridization, rather than outright break with the past. For example, rather than reinvent the abstractions that govern processing to take advantage of the new possibilities offered by server farms, Amazon's EC2 cloud service reproduces virtual von Neumann architectures, so that consumers can run existing applications and commercial software.

Moves

In spite of these conservative forces, the resource stacks are in a constant state of (more or less pronounced) flux. The fundamental pressure for change is exerted by the drive for greater efficiency. While we are familiar with efficiency as achieved by material science (greater storage density, greater communication speed, greater clocking speed), less familiar are the kind of moves within the stack that also result in efficiency gains.

The most common strategy is to breach the independence of two adjacent layers through co-design, as illustrated above (co-design of processors and programming language

DOI: 10.1002/asi

abstractions, Google's co-design of GFS and MapReduce, cross-layer design for wireless networks). Another move, encapsulation, consists of wrapping the interface of a module into another layer—e.g., Google's GFS and the Unix file system. Because it reuses the existing file system interface, the GFS will also inherit all of its built-in inefficiencies. These are compensated, however, by the benefits of increased elasticity of the resource, low costs of the components, and co-design of the file and application layer. Inserting new layers in the stack can prove extremely difficult. For the last quarter of a century, the computer science community has attempted, with limited success, to define a standard for a layer (CORBA) that would enable applications to transparently communicate with other applications and enable more distributed forms of processing (Henning, 2006).

The most powerful move remains the imposition of an a-priori modular decomposition before protocols solidify in both hardware and institutions, as attempted by ISO with the OSI model. But the OSI model already conflicted with another architecture, that of TCP/IP, illustrating that when it comes to infrastructure, there is no such thing as a clean slate. The clean slate is precisely what computer scientists often fantasize about when faced with infrastructural change: "Given an excuse to reinvent the whole software/hardware stack, this opportunity is also a once-in-a career chance to fix other weaknesses in computing that have accumulated over the decades like barnacles on the hull of an old ship" (Asanovic et al., 2009, p. 3). Yet, as Ciborra (2000, 2002), Star and Ruhleder (1996), and the preceding sections have suggested, infrastructural change proceeds just as much through improvisation, bricolage, and drift, than it does through planification and control.

These moves, this persistence, provide essential analytical tools for strategic planning, for understanding the complex positioning of market players through vertical and horizontal integration, standards, and interoperability. This is particularly true at this moment in technical history, as networks migrate towards architectures where fundamental computing resources—processing, storage, communication—are extensively distributed and shared, and their power leveraged for an ever-increasing range of societal functions.

Conclusion

What kind of work does a theory of digital materiality perform? One can envision the difficulties that would arise in attempting to account for architecture without a working concept of the tensile strength of steel, of the durability of concrete, of the density of wood. Indeed, that these materials differ in their physical characteristics registers on the entire ecology of the field, from construction methods to economics, design traditions, professional training, cultural symbolism, etc., and no meaningful analysis can ignore these differences. Furthermore, there exists both a precise technical understanding of these differences, appropriate to the expert communities concerned with these issues, and a lay intuition that informs everyday discourse. Yet we today have neither technical language nor intuition for something akin to the tensility, durability, or density of computing resources. Without a basic understanding of the material constraints under which computing systems operate, essential dynamics that animate the built environment of the virtual will remain invisible and unaccounted for.

Furthermore, a fundamental shift in the contemporary social sciences has been to increasingly ground cognition, identity, subjectivity, and collective action in the body and its material environment, rather than solely in the brain (Malafouris & Renfrew, 2010). The Cartesian heritage of philosophies of mind has made it difficult to account for the fact that "as embodied agents, able to move and act in a persisting material world, we are demonstrably able to profit from a variety of strategies that make the most of bio-external sources of order and information" (Clark, 2010, p. 23). Such a perspective yields powerful insights: in *Science and Technology Studies* for example, Donald MacKenzie (2006) has famously argued how financial models (e.g., Black-Scholes option pricing) "perform" markets, through computerized trading systems, but also through the mundane technology of paper—"sheets which floors traders could carry around, often tightly wound cylindrically ... so that a quick squint would reveal the relevant prices." Even more the point, in a perceptive analysis of video codecs, the compression algorithms that power the media culture of the Internet, Adrian Mackenzie argues the material constraints of computing intimately register within our very perceptual systems:

> "Eyes and ears do not have universal, timeless physiological properties. They have media-historical habits. Electronically mediated visual culture shapes eyes and ears, and creates perceptual habits at many levels. For instance, the conventions of the rectangular 4:3 ratio TV screen, the 16:9 ratio cinema screen, the number of scan lines, or the colour models of PAL/NTSC television broadcasts go deep into visual habits. Sensations of colour, texture, brightness and level of detail all feed into habits of viewing. The video codecs behind DVDs, High Definition Television, mobileTV for 3G cellular telephones, RealPlayer, or satellite digital video broadcasts attempts to take those expectations into account and meld them with the limited channel capacities of networks, broadcast spectrum or cables" (Mackenzie, 2010, p. 145).

Indeed, without modes of analysis grounded in the stuff of computing, we shall find ourselves in the awkward situation of resorting to theories that account for embodied subjects situated and interacting in environments curiously lacking specific material constraints.

Conversely, the analysis proposed here provides a picture of computing dramatically at odds with that conveyed by the trope of immateriality. Indeed, it is only recently that computing has been approached as "something having a history, rather than just being permanently in a state of improvement" (Fuller, 2008, p. 7). As Haigh (2009) notes, "software tools encapsulate craft knowledge, working practices, and cultural assumptions.... these encapsulated qualities are

DOI: 10.1002/asi

reproduced with each new software revision, often enduring for decades" (p. 7). Indeed, this paper has shown how abstractions, embedded in software, hardware, and institutions, endure across decades, acting as conservative forces on infrastructure evolution. Yet much of the historical dialectic between abstraction and implementation is absent from computer scientists' own accounts of their discipline. As Blaauw and Brooks (1997) remark in their monumental study of computer architecture, "when reading the professional paper describing the architecture of a new machine, it is often difficult to discern the real design dilemmas, compromises, and struggles behind the smooth, after-the-fact description" (p. 7).

Yet these dilemmas, these compromises, these struggles will increasingly matter, as the software infrastructure comes to mediate a breathtaking proportion of social relations. As Miller (2005) notes, objects are important "precisely because we do not 'see' them. The less we are aware of them, the more powerfully they can determine our expectations by setting the scene and ensuring normative behavior, without being open to challenge. They determine what takes place to the extent that we are unconscious of their capacity to do so" (p. 5). The inability for a technical field to retrace the historical path of its most important and durable contributions has important consequences for its ability to critically reflect on its own evolution and the political work inherent in infrastructural design.[6]

The need for such critical reflection is particularly timely at this juncture in the evolution of computing. The current move towards mobile and cloud computing will introduce fundamentally new economic mechanisms for the valuation of the material resources of computing.[7] In turn, the allocation, distribution, and metering of these resources, and the design of the infrastructure that mediates them will become an increasingly visible and contested phenomena. The analytical framework in this paper provides means to make the infrastructural work of computing more visible, so that it may be engaged with by a broader range of stakeholders.

Acknowledgments

I dedicate this article to the late Susan Leigh Star, whose pioneering work on infrastructure inspired and informed this project. I thank Johanna Drucker, Jonathan Furner, Annette Weisser, and Emma Dillon for insightful feedback, support, and sharing of ideas, as well as two anonymous reviewers for their valuable comments. I especially thank the students of "IS 270: Introduction to Information Technologies," whose questions, concerns, and efforts provided the opportunity to develop much of this material.

[6]The movement of "software patterns" is one attempt to capture what currently gets lost in the current dominant style of technical accounts. See Buschmann, Henney, and Schmidt, 2007.

[7]See, for example, Amazon's spot market for its EC2 cloud computing service, http://aws.amazon.com/ec2/spot-instances.

References

Abbate, J. (2000). Inventing the internet. Cambridge, MA: MIT Press.

Abelson, H., Ledeen, K., & Lewis, H.R. (2008). Blown to bits: Your life, liberty, and happiness after the digital explosion. Upper Saddle River, NJ: Addison-Wesley.

Agre, P. (1997). Computation and human experience. Cambridge, New York: Cambridge University Press.

Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., & Kubiatowicz, J. (2009). A view of the parallel computing landscape. Communications of the ACM, 52(10), 56–67.

Barlow, J.P. (1996). A declaration of the independence of cyberspace. Retrieved from https://projects.eff.org/~barlow/Declaration-Final.html.

Blaauw, G.A., & Brooks, B.F. (1997). Computer architecture: Concepts and evolution. Reading, MA: Addison-Wesley.

Blanchette, J.F. (2011). The noise in the archive: Oblivion in the age of total recall. In Gutwirth, Poullet, De Hert, & Leenes, (Eds.), Privacy and data protection: An element of choice. Heidelberg, Germany: Springer.

Buchholz, W. (1963). File organization and addressing. IBM Systems Journal, 2(2), 86–111.

Buschmann, F., Henney, K., & Schmidt, D.C. (2007). Pattern-Oriented software architecture: On patterns and pattern languages. Hoboken, NJ: John Wiley & Sons.

Campbell-Kelly, M. (1990). Punched-card machinery. In W. Aspray (Ed.), Computing before computers. Iowa City, IA: Iowa State University Press.

Ceruzzi, P.E. (2003). A history of modern computing. Cambridge, MA: MIT Press.

Ciborra, C. (2000). From control to drift: The dynamics of corporate information infastructures. New York: Oxford University Press.

Ciborra, C. (2002). The labyrinths of information: Challenging the wisdom of systems. New York: Oxford University Press.

Clark, A. (2010). Material surrogacy and the supernatural: Reflections on the role of artefacts in 'off-line' cognition. In L. Malafouris & C. Renfrew (Eds.), The cognitive life of things: Recasting the boundaries of the mind (pp. 23–37). Cambridge, UK: McDonald Institute for Archeological Research.

Clark, D. (1996). Foreword. In Computer networks: A systems approach. San Francisco: Morgan Kaufmann.

Czajkowski, I.K. (1999). High-speed copper access: A tutorial overview. Electronics & Communication Engineering Journal, 11(3), 125–148.

Daley, R.C., & Neumann, P.G. (1965). A general-purpose file system for secondary storage. In AFIPS '65 (fall, part I): Proceedings of the November 30-December 1, 1965, Joint Computer Conference. New York: ACM.

Dean, J., & Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107–113.

Eckert, J.P. (1953). A survey of digital computer memory systems. Proceedings of the IRE, 41(10), 1393–1406.

Engler, D.R., & Kaashoek, M.F. (1995). Exterminate all operating systems abstractions. In Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (Hotos-V), 78.

Fuller, M. (2008). Software studies: A lexicon. Cambridge, MA: MIT Press.

Ghemawat, S., Gobioff, H., & Leung, S.T. (2003). The Google file system. ACM SIGOPS Operating Systems Review, 37(5), 43.

Gibson, W. (1984). Neuromancer. New York: Ace Books.

Haigh, T. (2009). How data got its base: Information storage software in the 1950s and 1960s. IEEE Annals of the History of Computing, 6–25.

Hayles, N.K. (1999). How we became posthuman: Virtual bodies in cybernetics, literature, and informatics. Chicago: University of Chicago Press.

Hayles, N.K. (2002). Writing machines. Cambridge, MA: MIT Press.

Henning, M. (2006). The rise and fall of CORBA. Queue, 4(5), 34.

Henriksson, R. (1998). Scheduling garbage collection in embedded systems. PhD thesis, Lund, Sweden: Lund University.

Hillis, D. (1999). The pattern on the stone: The simple ideas that make computers work. New York: Basic Books.

Hopper, G.M., & Mauchly, J.W. (1953). Influence of programming techniques on the design of computers. Proceedings of the IRE, 41(10), 1250–1254.

DOI: 10.1002/asi

Kawadia, V., & Kumar, P.R. (2005). A cautionary perspective on cross-layer design. IEEE Wireless Communications, 12(1), 3–11.
Kirschenbaum, M.G. (2008). Mechanisms: New media and the forensic imagination. Cambridge, MA: MIT Press.
Knoespel, K., & Zhu, J. (2008). Continuous materiality through a hierarchy of computational code. Théorie, Littérature, Epistémologie, 25, 235–247.
Leonardi, P. (2010). Digital materiality? How artifacts without matter, matter. First Monday, 15(6–7).
Levinson, M. (2006). The box: How the shipping container made the world smaller and the world economy bigger. Princeton, NJ: Princeton University Press.
Mackenzie, A. (2010). Every thing thinks: Sub-representative differences in digital video codecs. In C.B. Jensen & K. Rödje (Eds.), Deleuzian intersections: Science, technology, anthropology (pp. 139–162). New York: Berghahn Books.
MacKenzie, D.A. (2006). An engine, not a camera: How financial models shape markets. Cambridge, MA: MIT Press.
Malafouris, L., & Renfrew, C. (Eds.) (2010). The cognitive life of things: Recasting the boundaries of the mind. Cambridge, UK: McDonald Institute for Archeological Research.
Manovich, L. (2007). The language of new media (8th ed.). Cambridge, MA: MIT Press.
May, T.C., & Woods, M.H. (1979). Alpha-particle-induced soft errors in dynamic memories. IEEE Transactions on Electron Devices, 26, 2–9.
Mayer-Schonberger, V. (2009). Delete: The virtue of forgetting in the digital age. Princeton, NJ: Princeton University Press.
McGee, W.C. (1959). Generalization: Key to successful electronic data processing. Journal of the ACM, 6(1), 1–23.
McKenzie, D.F. ([1986] 1999). Bibliography and the sociology of texts. Cambridge, UK, New York: Cambridge University Press (Original work published 1985).
McKusick, M.K., Joy, W.N., Leffler, S.J., & Fabry, R.S. (1984). A fast file system for UNIX. ACM Transactions on Computer Systems (TOCS), 2(3), 181–197.
McKusick, M.K., & Quinlan, S. (2009). Gfs: Evolution on fast-forward. Queue, 7(7), 10.
Miller, D. (Ed.) (2005). Materiality. Durham, NC: Duke University Press.
Miller, D. (2009). Stuff. Polity.
Negroponte, N. (1995). Being digital. New York: Knopf.
Parnas, D.L., Clements, P.C., & Weiss, D.M. (1984). The modular structure of complex systems. In Proceedings of the Seventh International Conference on Software Engineering (pp. 408–417). Washington, DC: IEEE Press.
Patterson, D. (2010). The trouble with multi-core. IEEE Spectrum, 47(7), 28–32, 53.
Paul, G. (2009). Foundations of digital evidence. Washington, DC: American Bar Association.
Pleszkun, A.R., & Thazhuthaveetil, M.J. (1987). The architecture of lisp machines. Computer, 20(3), 35–44.
Preston, B. (1998). Why is a wing like a spoon? A pluralist theory of function. The Journal of Philosophy, 95(5), 215–254.
Preston, B. (2000). The functions of things, a philosophical perspective on material culture. In P. Graves-Brown (Ed.), Matter, materiality and modern culture (pp. 22–49). New York: Routledge.
Radin, G. (1983). The 801 minicomputer. IBM Journal of Research and Development, 27(3), 237–246.
Rappaport, T.S., Annamalai, A., Buehrer, R.M., & Tranter, W.H. (2002). Wireless communications: Past events and a future perspective. IEEE Communications Magazine, 40(5), 148–161.
Ritchie, D.M., & Thompson, K. (1974). The UNIX time-sharing system. Communications of the ACM, 17(7), 365–375.
Rosenheim, S. (1997). The cryptographic imagination: Secret writing from edgar poe to the internet. Baltimore: Johns Hopkins University Press.
Russell, A.L. (2006). 'Rough consensus and running code' and the internet-osi standards war. IEEE Annals of the History of Computing, 28(3), 48–61.
Sethi, R. (1996). Programming languages: Concepts and constructs. Reading, MA: Addison-Wesley.
Shannon, C.E. (1949). Communication in the presence of noise. Proceedings of the IRE, 37(1), 10–21.
Snyder, R.L., Jr. (1952, December). Devices for transporting the recording media. In Proceedings of the Review of Input and Output Equipment Used in Computer Systems (p. 15). Los Alamitos, CA: IEEE Computer Society.
Star, S.L., & Ruhleder, K. (1996). Steps toward an ecology of infrastructure: Problems of design and access in large information systems. Information Systems Research, 7, 111–134.
Ulrich, K. (2007). The architecture of artifacts. In Design: Creation of artifacts in society. Available at: http://www.ulrichbooks.org
Ungar, D., Blau, R., Foley, P., Samples, D., & Patterson, D. (1984). Architecture of SOAR: Smalltalk on a RISC. In Proceedings of the 11th Annual International Symposium on Computer Architecture (pp. 188–197). New York: ACM Press.
Von Burg, U. (2001). The triumph of ethernet: Technological communities and the battle for the LAN standard. Stanford, CA: Stanford Business Books.
Warner, J. (2009). Materializing communication concepts: Linearity and surface in linguistics and information theory. In P. Turner, S. Turner, & E. Davenport (Eds.), Exploration of space, technology, and spatiality: Interdisciplinary perspectives (pp. 196–213). Hershey, PA, and New York: Information Science Reference.
Wing, J.M. (2006). Computational thinking. Communications of the ACM, 49(3), 33–35.
Yates, J.A. (2005). Structuring the information age: Life insurance and technology in the twentieth century. Baltimore: Johns Hopkins University Press.
Zimmermann, H. (1980). OSI reference model—The ISO model of architecture for open systems interconnection. Communications, IEEE Transactions on [Legacy, Pre-1988], 28(4), 425–432.
Zittrain, J. (2009). The future of the internet—And how to stop it. New Haven, CT: Yale University Press.

DOI: 10.1002/asi