Interoperable Convergence of Storage, Networking, and Computation

Micah Beck, Terry Moore, and Piotr Luszczek
mbeck@utk.edu, tmoore@icl.utk.edu, luszczek@icl.utk.edu
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN 37996
Accepted to *Future of Information and Communications Conference (FICC) 2019*

***Abstract*—In every form of digital store-and-forward communication, intermediate forwarding nodes are computers, with attendant memory and processing resources. This has inevitably originated efforts to create a wide-area infrastructure that goes beyond simple store-and-forward, a facility that makes more general and varied use of the potential of this collection of increasingly powerful nodes. Historically, these efforts predate the advent of globally routed packet networking. The desire for a converged infrastructure of this kind has only intensified over the last 30 years, as memory, storage, and processing resources have both increased in density and speed while simultaneously decreasing in cost. Although there is a general consensus seems that it should be possible to define and deploy such a dramatically more capable wide-area facility, a great deal of investment in research prototypes has yet to produce a credible candidate architecture. Drawing on technical analysis, historical examples, and case studies, we present an argument for the hypothesis that in order to realize a distributed system with the kind of convergent generality and deployment scalability that might qualify as "future-defining," we must build it from a small set of simple, generic, and limited abstractions of the low level resources (processing, storage and network) of its intermediate nodes.**

1. Introduction

A variety of technological, economic, and social developments — most notably the general movement toward Smart Cities, the Internet of Things, and other forms of "intelligent infrastructure" [1] — are prompting calls from various quarters for something that the distributed systems community has long aspired to create: A next-generation network computing platform. For example, the authors of a recent Computing Community Consortium white paper, writing with the US "Smart Cities" initiative [2] in view, express the research challenge as follows:

"What is lacking—and what is necessary to define in the future—is a common, open, underlying 'platform', analogous to (but much more complex than) the Internet or Web, allowing applications and services to be developed as modular, extensible, interoperable components. To achieve the level of **interoperation** and innovation in Smart Cities that we have seen in the Internet will require *federal investment in the basic research and development* of an analogous open platform for intelligent infrastructure, tested and evaluated openly through the same inclusive, open, consensus-driven approach that created [the] Internet." [3] [Emphasis in source]

The experiences of the last two decades have made the distributed systems community acutely aware of how elusive the invention of such a future-defining platform is likely to be [4]. Achieving this vision has been the explicit or implicit ambition of a succession of well funded and energetically pursued research and development efforts within or around this community, including Active Networking [5], Grid Computing [6], PlanetLab [7], and GENI [8], to name a few. Although these broad efforts have produced both valuable research and useful software results, nothing delivered so far has achieved the *deployment scalability* necessary to initiate the kind of viral growth that everyone expects such an aspirational platform to exhibit. At the same time, chronic problems with network hotspots were an early and persistent sign that the Internet's stateless, unicast datagram service had scalability limitations with respect to data volume and/or popularity. This fact has led to increasingly sophisticated and increasingly expensive technology "workarounds," from the FTP mirror sites and Web cache hierarchies of the early years, to the content delivery networks (CDN) and commercial Clouds we see today.

The central idea of this paper is that the appropriate common service on which to base an interoperable platform to support distributed systems is an abstraction of the low layer resources and services of the intermediate node, i.e., a generalization of the Internet stack's layer 2. The "Internet Convergence" of the 1990's developed the "hourglass" paradigm, with a best-effort datagram delivery as the common service, or "spanning-layer," at its narrow waist [9]; we believe that the paradigm required by the data saturated world now emerging in edge/fog environments is more accurately pictured as an "anvil" (Figure 1), with a common service interface that exposes storage/buffer, network, and processor resources in a programmable way. Drawing on technical analysis and historical examples, we argue that in order to build distributed systems with the kind of interoperability, generality and deployment scalability that might qualify as "future-defining," we must implement them using a small set of simple, generic, and limited abstractions of the data transfer, storage and processing services available at this

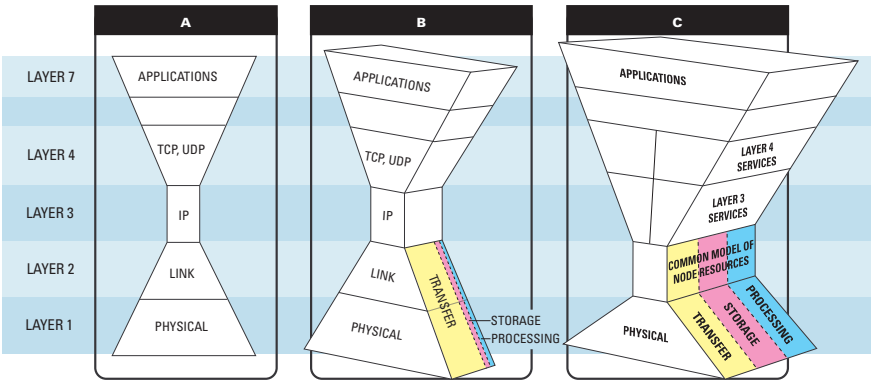

Figure 1. The Hourglass vs. The Anvil

layer. In our model, these abstractions all revolve around the fundamental common resource, the memory/storage buffer.

2. Background

Given the inclination of computer scientists to add features, the fact that every form of digital store-and-forward communication (including the Internet) has intermediate forwarding nodes that are computers, with attendant memory and processing resources, makes attempts to create a wide area infrastructure with services beyond simple store-and-forward inevitable. Such efforts to make more general use of these increasingly powerful nodes— a *generalized converged network*, in our terminology—predate the advent of globally routed packet networking (e.g. `uux` [10]). The exponentially increasing density and speed, and rapidly decreasing cost of memory, storage and processing resources over the past 30 years has only intensified the desire to define and scalably deploy a converged infrastructure of this general description. Yet despite the general consensus that it should be possible to do so, this aspiration has remained unfulfilled.

One problem is that the goal of converged networking runs in the opposite direction of the traditional architectural approach of the Internet design community, which insists that services other than datagram delivery must be kept out of the Network Layer of the communication protocol stack. This community maintains that the ability of the Internet to function properly and to continue growing globally depends on keeping this common service layer "thin", in the sense that it provides services that are simple, generic and limited [11]. From this point of view, services other than datagram delivery should be implemented in systems connected to the Internet as communication endpoints. Various rationales supporting this point of view are collectively referred to as "End-to-End Arguments" [11].

Since a router that has substantial system storage (i.e. other than network buffers) and generalized computational resources (i.e. other than forwarding) is neither difficult nor expensive to build, there have been numerous efforts to resist this orthodox point of view. The simple fact that storage and computational resources can be provisioned and located throughout the network at reasonable cost stimulates efforts in this direction. Moreover, the apparent opportunity to create such a powerful distributed infrastructure presents a temptation that is inherently difficult for computer scientists and engineers to resist. These facts, however, do not make it a good idea to add extensions to the fundamental service of the global Internet, nor do they ensure that if it is built, service creators and users will adopt it in sufficient numbers to enable economic sustainability beyond the prototype stage. Indeed, while a number of plausible network service architectures have been defined that can provide access to such distributed resources [5], [12], the widespread deployment of extended services on a converged wide area infrastructure has proved elusive.

Perhaps an even more compelling reason for the continued drive to create such a converged infrastructure is that some important distributed applications cannot be efficiently and effectively implemented through decomposition into two components, one implemented by a "thin" datagram delivery service in the core of the network, and the other implemented at "fat" endpoints. For example, some applications require an implementation that is sensitive to the location of storage and computation in the network topology. Point-to-multipoint communication was an early and obvious example. Using simple repetition of unicast datagram delivery was viewed as too inefficient by early Internet architects, but an efficient tree could be built only through the use of network topology information. Such low level information was seen as inappropriate for users of the "thin" and stable Network layer to access. Thus, multicast was added to Layer 3, fattening that thin layer with services that seemed to address this issue. However, IP multicast has proved difficult to standardize and has failed to achieve the universal deployment of "simple, generic and limited" unicast IP datagram delivery.

But problems with lack of generality in the intermediate nodes were manifest even in highly successful Internet applications. The early growth of the Internet was fueled by applications that seemed to fit the unicast datagram delivery model well enough: FTP and Telnet. Of these, the one-to-many nature of FTP, albeit asynchronous, created a problem in the distribution of popular and high-volume files. Ignoring

the implications of topology led to ineffective use of the network, with hotspots at servers that attracted high volumes of traffic and unnecessary loads placed on expensive and overburdened wide area links. The result was the creation and management of collections of FTP mirror sites [13], and the ubiquitous invitation for users to "choose a mirror site near you", which meant the use of approximate information about network topology by the end-user, at a level above even the Internet stack's Application Layer.

The advent of the World Wide Web exacerbated the problem of indiscriminate access to servers with no reference to network topology or even geography. Mirror sites for file download proliferated, and redundancy in the storage of all high-traffic Web content became a necessity. A Network layer that hides topology from its clients is, after all, an inherently inadequate platform on which to build high traffic globally distributed systems. The need to work around this reality gave rise to automated Web caching [14], [15] and server replication [16], [17], which were precursors to modern Content Delivery Networks [18], [19].

It should be noted that although both Web caching and server replication are obvious examples of the convergence of networking and storage, they also require computation in the implementation of policy and server-side processing; and so in fact they represent convergence of all three fundamental computational resources. We examine the approach to convergence that they represent in more detail in section 5 below. Following a different strategy, Logistical Networking, discussed in section 6.2, implements a convergence of networking and storage service that avoids the need for general computation by minimizing policy and other server-side processing [20], but was later extended to include limited server-side operations [21].

3. The Convergence Spectrum

The interplay between technological divergence and convergence is a dialectic with a long history. In the area of computing and communications, there was an early divergence in the conception and implementation of several different information technology resources. Because of the phenomenon of path dependence [22], such divergence has tended to be self-reinforcing, leading to a set of familiar technology *silos*, such as data transmission and broadcast using radio frequency signals, virtual circuits, switches and gates and magnetic or solid state storage cells. The success of the Internet in the 1990's provided the foundation for the substantial or partial convergence of various traditional telecommunication silos–telephony, broadcast television, etc.–in this century [23], but the fundamental silos at the base of computing–storage, processing, networking–have remained as entrenched as ever.

The early divergence of basic computational resources has given rise to conceptual, technological and organizational silos corresponding to correspondingly isolated communities. Formal models and methods of reasoning have been adapted to deal with the complexity and specific issues of each niche. For example Boolean logic is a useful model of solid state

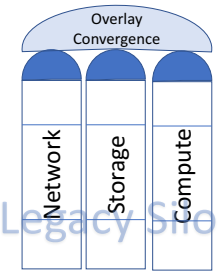

Figure 2. Overlay Convergence of Legacy Silos

circuits, and "stateless" communication is a useful model of wide area data network built out of switches and FIFO line buffers.

The development of silos has been an enabling strategy for modeling and optimization of these quickly evolving technological fields. However, they have also led to the creation of service stacks, or silos, with highly specialized services at the top layers (see Figure 2). But because the low level resources that these silos encapsulate can only be accessed through high level services, this inevitably tends to create barriers to the flexible and efficient use of constituent low level resources *in combination*.

The problem with silos as a strategy for dealing with the complexity and specialization of disparate underlying technologies has become more pronounced due to the evolution of low level systems toward general mechanisms that utilize processors or digital logic controlled by software, firmware or by hardware designed using computerized tools. Such generality in low level mechanisms holds out the possibility of the implementation of highly efficient system architectures, with optimizations that span traditionally disparate resources. The challenge is to bridge or eliminate the existing silos, or, in other words, to implement *convergence*.

We say that a service interface (i.e., an API) is *converged* if it gives unified access to multiple low-level resources (or services) traditionally available only through isolated service silos. Historical examples of system design that leverage convergence include the auto-increment register, direct memory access I/O and vector processing.

When the goal is to achieve convergence for a service interface using previously non-interoperable resources, there are two fundamentally different ways to go about it: *overlay convergence* which combines silos at a layer above their high level services, and *interoperable convergence*, which strives to unify their foundations. These two strategies lie at the ends of a spectrum along which a variety of familiar examples can be arrayed.

Overlay convergence is the most common approach lying at one end of the spectrum and creating a high level interface that provides access to a number of traditionally separate service silos. We term this approach *overlay convergence* because it typically involves the creation of a service that provides unified access to the existing service silos from above, through their high level client interfaces (see Figure 2). By contrast, at the other end of the convergence spectrum

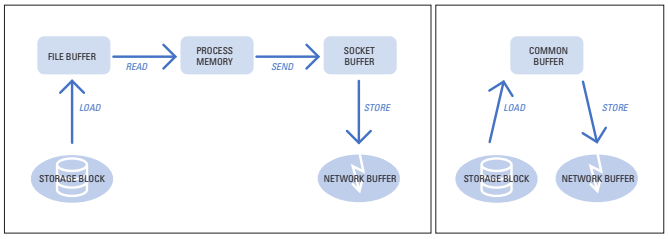

Figure 3. Read-Send vs. Sendfile

is what we call *interoperable convergence*. We say that a platform is interoperably converged if it minimizes the imposition of unnecessary high-level structure or performance costs when applying different low-level services, so that those underlying common resources can be accessed without incurring the overhead and restrictions that are associated with complex and specialized service silos.

Both overlay and interoperable forms of convergence seek to create a common service, or *spanning layer*, which supports a generalized set of applications requiring resources that were previously segregated. The purpose of such a spanning layer is to enable interoperability in the support of this rich category of applications [9].

Some examples that fall along this spectrum and illustrate these different approaches include the following:

- The BSD kernel created an overlay convergence of Unix process and local file management with local and wide area networking through the addition of the *socket* related system calls. While some calls that act on file descriptors such as `read()` and `write()` were extended to operate on sockets, the level of integration is mainly syntactic and does not extend deeply into integration of common functions such as buffer management.
- Following the implications of this example, in order to move data stored in a file to a TCP stream in UNIX, it was originally necessary to move it into a process' address space using the `read()` system call and then inject it into the TCP stream using `send()` (see Figure 3). A more interoperable approach is a combined `sendfile()` system call was added as an extension to Linux that allows data to be transferred from storage into a kernel memory buffer and from there directly to the network without moving it to process memory or using a dedicated network data buffer. However this buffer management solution is applicable only in quite specific scenarios. We thus characterize it as a *workaround*.
- A distributed file system converges storage and data movement in a more interoperable manner. These resources are traditionally available through local file management and networked file transfer tools.
- A database system can store a set of tuples without order, but traditional data movement tools operate on files. Thus, it is necessary to serialize a set of tuples as a file in order to send it to a remote database system. The file is transferred serially, using TCP with retransmission to keep the serialized data in order. A somewhat interoperable approach would generate the serialized stream representing the tuple set on demand, rather than creating and storing it as a complete file. A more interoperable approach would be to implement a specialized protocol that takes advantage of the lack of natural sequentiality in the tuple set to perform retransmission out-of-order. This might require additional work to ensure that the new protocol was "TCP-friendly" when used in public shared networks.
- A data analysis system (such as MapReduce [24]) traditionally consists of a deep data store and a dedicated compute resource such as a cluster or a shared-memory parallel computer. Visualization typically requires data to be moved from the data store to the compute resource which then returns its results to the data store. User access then requires that the visualization output be moved to and interpreted by a human interaction system. A more interoperable approach would allow computations to be applied to the data in the data store (in-situ), and for the user to interact with the results of that computation directly as it occurs.

4. Deployment Scalability

Because it is impossible to evaluate alternative strategies without identifying a criterion for success, we introduce the concept of *deployment scalability* as the goal of creating converged infrastructure. We define *deployment scalability* as widespread acceptance, implementation and use of a service specification. The workarounds we have described build overlay converged network but they are not interoperable and cannot achieve deployment scalability.

In a recent paper [25], Beck makes an argument for a fundamental design principle underlying systems that exhibit deployment scalability:

> **The Deployment Scalability Tradeoff** There is an inherent tradeoff between the deployment scalability of a specification and the degree to which that specification is weak, simple, general and resource limited.

The terms "simple, generic and resource limited" are derived from the classic paper "End-to-End Arguments in System Design" by Saltzer, Reed and Clark which discusses them in the context of Internet architecture. The term "weak" refers to logical weakness of the service specification as a theory of program logic, and is due to Beck's partial formalization of the arguments in that paper. Stating this principle as a tradeoff is a further refinement of the usual interpretation of the original paper as an absolute rule (or principle) requiring or prohibiting particular design choices [26].

The classic example of the application of the End-to-End Principle, from which its name is derived, is the location of the detection of data corruption or packet loss or reordering in the TCP/IP stack [11]. The scalability argument for end-to-end detection of faults is that removing such functions

from the spanning layer makes it weaker, and therefore potentially admits more possible implementations. Because fault detection can be implemented above the spanning layer, the set of applications supported is not reduced.

The evolution of process creation in Unix teaches a similar lesson. In early operating systems it was common for the creation of a new process to be a privileged operation that could be invoked only from code running with supervisory privileges. There were multiple reasons for such caution, but one was that the power to allocate operating system resources that comprise a new process was seen as too great to be delegated to the application level. Another reason was that the power of process creation (for example changing the identity under which the newly created process would run) was seen as too dangerous. This led to a situation in which command line interpretation was a near-immutable function of the operating system that could only be changed by the installation of new supervisory code modules, often a privilege open only to the vendor or system administrator.

In Unix, process creation was reduced to the `fork()` operation, a logically much weaker operation that did not allow any of the attributes of the child process to be determined by the parent, but instead required that the child inherit such attributes from the parent [27]. Operations that changed sensitive properties of a process were factored out into orthogonal calls such as `chown()` and `nice()`, which were fully or partially restricted to operating in supervisory mode; and `exec()` which was not so restricted but which was later extended with properties such as the *setuid* bit that were implemented as authenticated or protected features of the file system. The decision was made to allow the allocation of kernel resources by applications, leaving open the possibility of dynamic management of such allocation by the kernel at runtime, and creating the possibility of "Denial of Service" type attacks that persists to this day.

These two classical examples of interoperable convergence point to a significant issue. Changing the low level services on which existing silos are built requires the redesign and reimplementation of complex higher level service stacks. The influence of path dependent thinking and the pain of abandoning "sunk investments" explain the natural tendency of service provider communities to develop *workarounds* that preserve widely deployed lower level services.

5. Web Caching and CDNs: A Case Study in Overlay Workarounds

During what might be called the "Internet Convergence" in the 1990's, the generality and scalability of the Internet's datagram delivery model gave rise to the idea of using it to implement the convergence of broadcast, telephony and data services [23]. The emergence of unicast datagram delivery as the only universal Internet service (discussed in Section 2) has meant that the underlying capabilities of analog connectivity mechanisms to implement true broadcast and to provide quality of service guarantees through resource reservation are not accessible to Voice over IP and Streaming Media over

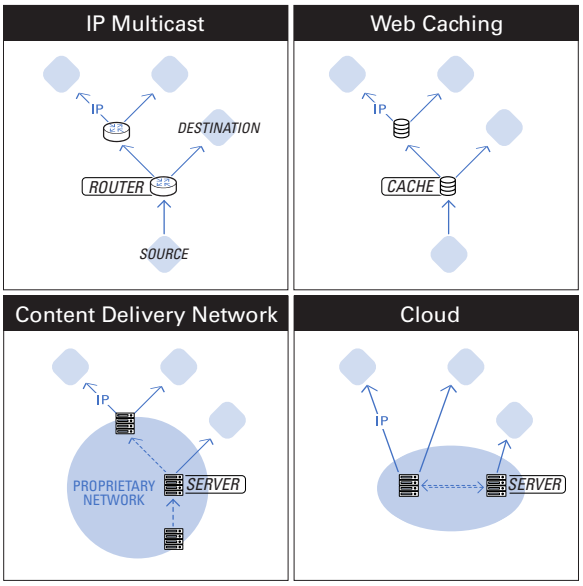

Figure 4. Overlay Workarounds Addressing Point-to-Multipoint Distribution

IP protocols. In spite of such limitations, the convenience and cost benefits of convergence workarounds continue to dominate the commercial development of these services.

But the absence of a universal point-to-multipoint communication mechanism within the common Network layer of the Internet also left a large class applications without native support, and this, in turn, has generated a whole series of overlay workarounds (see Figure 4). For instance, the distribution of static Web pages (those that require only minimal rewriting of stored HTML pages) can be viewed as a form of point-to-multipoint application. A browser cache uses moderate storage resources in the network endpoint to capture the delivered Web page and associated metadata and minimal processing to implement the cache policy and mechanism. A proxy cache uses larger scale storage and has a greater processing load, which is supplied by a substantially provisioned network intermediate node. The convergence of resources in Web caches led to an architectural development in which application-specific proxies are uploaded to the a "middlebox" platform which implements both caching and general processing.

Web caching played a pivotal role in the expansion of the Web as a global data distribution service during the period when intercontinental data links were too expensive to allow unfettered access by academics. A hierarchical system of large scale caches was developed and deployed in US Research and Education Networks [14], [15] and use of national caches to access Web data across intercontinental links was made mandatory in many countries.

In spite of its effectiveness in reducing the traffic loads due to delivery of static Web pages, the popularity of intermediate caches has waned dramatically in the past decade. There are several reasons for this trend including:

- The correctness of Web caching relies on lifetime metadata being provided by origin servers which is

often missing or inaccurate.
- The growth of dynamic Web applications means that many Web objects are not cachable.
- The lack of an accurate and universal mechanisms for reporting views interferes with the dominant business model of Web advertisers.
- Reliance on a complex cache infrastructure decreases the control of the implementer of a Web service over the Quality of Service experienced by customers.

Many of these factors stem from the implementation of Web caching on top of the HTTP application protocol, albeit with some modifications to increase control over intermediate and browser caches by origin Web servers. Cache networks are an overlay which accesses Web services from the top of the protocol stack and thus does not allow the degree of fine-grained control that is required for seamless convergence.

An alternative approach is to start from the source, and to replicate the functionality of the Web server on multiple network nodes. Manual procedures for FTP mirroring led to automated mechanisms like Netlib [13], and high traffic Web sites gave rise to sophisticated cluster and geographically distributed server replication schemes [16], [17].

Commercial content Delivery Networks have approached the problem in a somewhat different way, using HTTP and streaming protocols for client access almost unchanged. This is analogous to the way that online services (e.g. Compuserve and AOL) and ISPs used telephone services. CDNs have instead focused their innovation on the underpinnings of the Internet in order to improve the effectiveness. replication. They use a combination of server side caching, distributed file and database systems and complex streaming and synchronization protocols implemented on proprietary international networks of application-specific servers.

CDN Web and DNS servers may be implemented as applications processes, but by using lower layer Internet mechanisms through now-commonplace layering violations (such as topology-sensitive DNS resolution), they use knowledge of network topology and other low level information that is intended to be encapsulated within the Network Layer of the Internet architecture. Modern extensions to the Network layer may allow CDN's to be implemented without such violation of layering, but at the expense of creating a "fatter" and less generic Network layer (see Section 4).

Commercial CDNs are thus a kind of Chimera, patched together from proprietary components and standard, low level components of the Internet. They create a proprietary, specialized network with their own services as the spanning layer, using the Internet as tools in their implementation and as a means of reaching end users. This view is supported by the trend toward using private or non-scalable mechanisms to implement internal communication among centralized and distributed CDN nodes. Since currently emerging paradigm for edge computing "...extends the CDN concept by leveraging cloud computing infrastructure..... [28]," the non-interoperable nature of these overlay approaches undermines the coherence of Internet-based information service ecosystems. As we argue below, *Exposed Buffer Processing* offers a more interoperable and unified foundation for such next-generation ecosystems.

6. Exposed Buffer Processing: An Approach to Interoperable Convergence

While operating system interfaces such as POSIX provide access to storage, networking and computing services, they do so in ways that conform to the traditional silos.

- File system calls do not have explicit access to general networking or computation resources.
- The sockets interface does not provide access to general storage and or computation resources.
- The POSIX process management functions do have only the minimal necessary overlap with storage and network functions (notably specifying an executable file image in the `exec()` system call).

However, the core resource that is used to implement all these silos is the persistent memory or storage buffer.

- In storage, storage blocks or objects are used in the implementation of higher level file and database systems, along with RAM memory buffers that are used to improve performance, enable application/OS parallelism and allow for flexible exchange of data with other operating system data structures.
- In networking, buffers are used at the endpoints for much the same reasons as storage, and are used at intermediate nodes to allow for asynchrony in the operation of store-and-forward packet networking.
- In computing, memory pages make up process address spaces, are also used to enable asynchrony in inter-process communication, and hold all other operating system data structures used in the implementation of functions on behalf of processes.

So although convergence of storage, networking and computation is possible through conventional operating system interfaces using the generality of the user process as a gateway between silos, a more interoperable approach is to expose a common abstraction of the underlying resource that all of these high level silos operate on, namely persistent storage blocks or memory buffers. We call this approach to convergence *Exposed Buffer Processing*.

6.1. Core EBP functionality

Exposed Buffer Processing is a general architectural idea that can have different implementations. The original implementation takes the form of the Logistical Networking stack described in Section 6.2 encapsulates the service as remote procedure call over TCP. Certain aspects of this implementation would have to be redesigned in an implementation directly over a datagram-oriented protocol.

- `allocate` – This call allocates storage capacity into which data can be stored. It is possible for this allocation to be performed implicitly (as part of the `store` operation described below) or explicitly (as a reservation

of resources which subsequent `store` operations can use.) An allocation call specifies several parameters that limit characteristics such as duration that limit its effect. An important attribute of an allocation is a local name by which stored data is referenced in subsequent calls.
- `store` and `load` – These calls allow EBP clients to store data in an allocated buffer or to load data that has previously been stored.
- `transfer` – This call transfers data between buffers specified by name on the same or on different depots.
- `transform` – This call applies a named operation to a set of buffers on a single depot, potentially transforming the data of one or more of them. The operation name is local to the depot and must have been defined by an code upload operation which is specific to the implementation of the depot. Operations will be assigned names through a client-community process that ensures that names are used in a sufficiently consistent manner.

6.2. Logistical Networking as EBP in Overlay

Over the past 15 years the Logistical Networking project [20], [21], [29] has worked to define an approach to Exposed Buffer Processing that is implemented as an overlay on the Internet. An examination of the key components of that implementation provides an EBP proof of concept:

- **Internet Backplane Protocol (IBP)**: IBP is a generalized Internet-based storage service that is encapsulated as remote procedure call over TCP. IBP was designed to be simple, generic and limited following the example of the Internet Protocol (IP) [11]. It is a best effort service, its byte array allocations are named only by long random keys (capabilities) and represent leases whose duration and size are limited by the individual intermediate node (in analogy to the IP MTU). The intermediate node that implements IBP is called a *depot*, and it is intended as a storage analog to IP routers. In many ways IBP is closer to a network implementation of `malloc()` than a conventional Internet storage service like FTP, and in addition every IBP allocation is a lease of storage resources which can be limited in duration. IBP has been implemented in both C and Java.
- **exNode**: Because IBP is such a limited service, the abstraction of an allocation that it supports does not have the expressiveness of the file abstraction that users typically expect of a high level data management system. The exNode is an abstract data structure that holds the structural metadata required to compose IBP allocations into a file of very large extent, with replication across IBP depots, identified by their DNS name or IP address [30]. The exNode can be thought of as an analog to the inode used in early Unix file system implementations. The exNode has both standard XML and JSON sequentializations.
- **Logistical Runtime System (LoRS)**: The exNode can be used as a file descriptor to implement standard file operations such as **read** and **write**. The Logisticsal Runtime System (LoRS) uses the exNode to implement efficient, robust and high performing data transfer operations. Some of the techniques used in the implementation of LoRS are comparable to those used in parallel and peer-to-peer protocols [31].
- **Logistical Distribution Network (LoDN)**: While the exNode implements topological composition of IBP allocations to implement large distributed and replicated files, it does not deal with the temporal dimension introduced by IBP's use of storage leases. LoDN is an active service which holds exNodes and applies storage allocation, lease renewal and data movement operations as required to maintain policy objectives set by end users through a declarative language and manageable by an intuitive human interface.
- **Network Functional Unit (NFU)**:The NFU was introduced as a means to allow simple, generic and limited in-situ operations by a depot to data stored in its IBP allocations. The NFU has been used in numerous experimental deployments, and has been shown to enable robust fault tolerance and high performance is a wide variety of applications [32], [33], [34]. However, the middleware stack that supported such experimentation has never been fully integrated with the deployed versions of LoRS and LoDN or the Data Logistics Toolkit (discussed below), and so the NFU has never been used in a persistent large scale deployment.

6.3. "Packetization" of Storage and Processing

One way to characterize EBP's simple, generic, and limited design philosophy for the abstractions of common spanning layer services is to say that it extends the idea of "packetization" from the domain of networking, where it has proved so remarkably successful, to the domains storage/memory and processing as well. Unfortunately, this contradicts the impulses many designers who have historically relied on the more complex, specialized and virtually unbounded services. The relevant contrasts between packet-based and circuit-based approaches are familiar and clear in realm of **Networking**:

- **Size**: Circuit-based networks allow an unbounded amount of data to pass over a persistent circuit, in analogy to an electrically connection, masking the underlying digital implementation in terms of MTU-limited packets. The Internet exposed the MTU and required endpoints to concatenate packets into streams.
- **Failure**: Circuit-based networks provide Quality-of-Service (QoS) guarantees sufficient to enable application developers to either ignore occassional communication faults or to fail catestrophically when they are detected. The Internet exposed the possibility of failure by dropping faulty packets and by exporting a best effort service, requiring endpoints to detect and respond to failures.
- **Locality Independence**: Circuit-based networks can allocate resources and maintain state along a specific path from sender to receiver, helping to ensure fast forwarding and providing a stable platform for implementation of auxilliary services. The Internet allows

every packet in a connected flow to be forwarded along a different path, putting the burden for maintaining stability on the packet routing scheme and ruling out connected services that require the maintenance of state, but enabling great resilience in the face of failures and changes in topology.

The similarities between between networking and storage make the the analogous set of contrasts relatively easy to work out for the realm of **Storage**:

- **Size**: File-based models of storage allow a very large amount of data (assumed by many applications to be virtually unbounded) to be stored as a single linear data extent. Logistical Networking (i.e., EBP in overlay) exposes a maximum storage allocation size imposed by the storage resource (analogous to the Internet Protocol's MTU) requiring endpoints to explicitly concatenate allocations into files.
- **Failure**: File and database systems provide QoS guarantees sufficient to enable application developers to either ignore occassional storage faults or to fail catestrophically when they are detected. Logistical Networking exposes a simple failure model (faulty operations terminate with unknown state for write-accessible storage) and by exporting a best effort service, requiring endpoints to explicitly detect and respond to failures.
- **Locality Independence**: File-based models of storage can allocate resources and maintain state on a well-connected "site" to manage fault tolerance and replication in terms of where "copies" reside. Logistical Networking allows every allocation comprising a file to be managed independently, potentially spreading them across topologically seperated nodes, moving and storing data on a fine-grained basis as called for by applications (e.g., data streaming).

Finally, although computation can and often does transform the data on which it operates, an analogous set of contrasts can none less be worked out for the realm of **Computation**:

- **Size**: Process-based computation allows an unbounded amount of processing to be performed one or a set of closely-coupled threads. The Network Functional Unit (i.e., EBP in overlay) exposes a unit of processing that can be limited in many resource dimensions, including elapsed clock time, CPU cycles consumed, RAM allocated during execution and I/O activity performed, requiring a runtime system to concatenate limited resources to create an unbounded virtual execution model.
- **Failure**: Process-based computation provides QoS guarantees sufficient to enable application developers to either ignore occasional processing faults or to fail catastrophically when they are detected. The NFU exposes a simple failure model (faulty operations terminate with unknown state for write-accessible storage) and exports a best effort service, requiring endpoints to explicitly detect and respond to failures.
- **Locality Independence**: Process-based computation can allocate resources and maintain state on a set of well-connected processors, enabling successive time slices to execute sequentially in a manner that leverages continuity of operating system and application data state. The NFU allows every allocation comprising a process to be managed independently, potentially moving them and the memory/storage allocations that comprise the state of supervisory and application data state as required (eg fault tolerance and load balancing).

6.4. EBP Below the Network Layer

The argument for creating a converged layer to support the Internet and other global distributed services is compelling. The need for distributed systems to have access to and control over low layer network characteristics including topology and performance is clear in the steps that have been taken to work around the stricture that forbids such direct access in the Internet architecture.

We propose the creation of a platform based on a common service similar to IBP but which models the networking capabilities of the Link Layer. We use the term Exposed Buffer Processing for this as-yet-unrealized service. The central idea of this paper is that the appropriate platform for the creation of distributed systems is some form of EBP. We emphasize that EBP need not follow the design of IBP, as long as it takes appropriate account of the Deployment Scalability Tradeoff. We offer experience with IBP as an overlay form of EBP for the consideration of the community.

7. Applications of EBP

7.1. Scientific Content Delivery

Dissemination of data is one of the fundamental challenges of modern experimental and observational science. There is a general move toward the open sharing of raw data sets, enabling replication of analyses, cross-cutting studies, innovative reexamination of previously collected data and historical examination of collection and analysis techniques [35], [36]. In many case the data collected is large and observation is continuous, as in remote data from satellites and other sensors [37], experiments such as the Large Hadron Collider [38], or broad harvesting of multimedia content [39]. The resources required to make such data streams instantaneously and persistently available can exceed the centralized capabilities of institutions or government agencies.

Commercial CDN or Cloud solutions may be too expensive, and may not adequately serve the entire global user community (see discussion of the Digital Divide below) and may not adequately support the publication by users of secondary data products resulting from their processing of raw data. However, the ICT resources required to address such problems may be affordable, and the community of user institutions may be capable of hosting them in a distributed manner. Using shared EBP infrastructure, we can build a distributed, federated content management system using the resources of the content provider and user communities

7.2. Digital Divide and Disasters

Modern network services take full advantage of the strong assumptions that can be made about the implementation of the Internet in the industrial world. It is common for services to rely on continual low-latency datagram delivery, always-connected servers, stable and uninterrupted datagram routing paths and high bandwidth connectivity to take just a few examples. Services implemented at Cloud Computing centers are among those that place great demands on the Internet backbone and "last mile" connectivity to edge networks.

Many services can be decomposed into synchronous and asynchronous components, and different "Data Logistics" strategies applied to each part [40]. Techniques used in Content Delivery Networks, including caching and prestaging can be applied on a fine-grained and even per-client basis. It is sometimes the case that the entire service can be implemented using edge resources. In other cases there is a component that can only be implemented using synchronous end-to-end datagram delivery across the backbone, but requires only low bandwidth. In some cases analysis of the application combined with reconsideration of the truely necessary characteristics of the service delivered to the end-user can reduce the need for high quality synchronous connectivity to the vanishing point. In a sense, reliance on strong network assumptions is often used to trade off unnecessary reliace on excellent network infrastructure for ease of development. This is a useful strategy for those who can afford and support the necessary infrastructure.

Today, some environments cannot support strong network assumptions, even when local IT resources are available. Examples are communities isolated through geography, economic (poverty, discrimination) or political circumstances (famine, war), or social factors. Disasters create environments where infrastructure is disrupted even in the most advanced societies. The recent response of modern network technologists has been to bring fixed or mobile wireless technology (satellite, 4G) into remote locations and to the scene of disasters or to create complex wireless infrastructures based on continuous aviation drones such as Google's baloon-based project Loon [41] and Facebook's drone-based project Aquila [42]. By contrast, using a mix of interoperable heterogeneous synchronous and asynchronous data transport integrated into a flexible platform to support a variety of distributed applications can be cheap, robust and easily deployed.

7.3. Big Data and Edge Processing

One of the inexorable trends in the collection of data is the emergence of large scale online sensors and instrument that produce data that must be subjected to volume-reducing processing before it can be passed over the network. Growing trends in sensor networks, the Internet of Things, and Smart Cities will severely exacerbate this problem, to say the least [43]. The historical approach of sending all such data to computation centers that are either self-contained or connected to their peers through heroic networking that may be private or even proprietary in nature is no longer sufficient to address the total size, globally distributed generation, and need for use by applications that we see today [44]. An alternative possible using EBP is to apply limited edge processing on the in the edge network using a converged infrastructure that can also store and transport data.

7.4. In-locus Data Analysis

Data Analytics (DA) has emerged as a new paradigm for understanding unreliable and varying environments. It goes beyond logging, reporting, and thresholding to perform meaningful analysis of large scale data sources that are networked through dynamic and distributed infrastructure. (The stage before batch or streaming analytics take place is often called "data assimilation".) DA is capable of extracting latent knowledge and providing insight from field sensors, computational units, and large mobile networks. At the same time, the number of these data sources and the resulting ingest rate are growing dramatically with increased edge hardware capability (resolution and sampling rate) and hybridization (multi-messenger and multi-sensor data acquisition). This requires new algorithmic approaches that closely integrate the network, I/O, and computational software stacks to lower the overheads and provide non-trivial data metrics at the edge. The emerging field of approximate and/or randomized algorithms position themselves perfectly in this role as they combine new methods for matrix approximation via random sampling that have recently been developed by the Applied Mathematics and Machine Learning communities.

Due to the recent interest [45], [46] in randomized and approximate algorithms, such methods have become a much better fit in an inherently unstable and constantly changing distributed environments by attaching a probabilistic measure to the result. In fact, there are many statistical techniques in the Randomized Linear Algebra class of algorithms that lend themselves perfectly to utilize the convergence principles of in-locus computing (as manifest in IBP's best effort Network Functional Unit operations as discussed in Section 6) and respond algorithmically to assimilate the inherent failures that naturally occur in a widely distributed system at the scale that we target. The iterative nature of most approximate methods allows us to incorporate erroneous response from a sensor or a network transmission and gradually remove the malformed data from the multidimensional subspace that is being worked on. Similarly, an intermittent lack of response from a sensor or a network element may naturally be incorporated as a sampling and selection operator that is triggered by a system-reported event as opposed to the classical method that uses a pseudo random number generator (PRNG) as an unbiased projector or selector. Also, the probabilistic nature of the approximate algorithms allows us to weigh the data sources based on their history of reliable responses and the quality of the data they delivered (if a measure of quality can be obtained, from, for example, a duplicate sensor). High quality sensors and network connections will, over time, gain large weights and thus render them highly probable

to be approximately correct as envisioned by the Probably Approximately Correct (PAC) learning framework [47].

8. Conclusions

In this paper, we have argued that interoperable convergence of storage, networking and processing is necessary in building a platform to support distributed systems which exhibits deployment scalability, and that the most effective implementation is a form of Exposed Buffer Processing at a layer below that which implements the Internet. Our argument rests on practical historical examples of the problems caused by the Internet's lack of expressiveness and an argument based on a partially formalized design methodology that the spanning layer of any converged infrastructure must be simple, generic and limited.

References

[1] E. Mynatt, J. Clark, G. Hager, D. Lopresti, G. Morrisett, K. Nahrstedt, G. Pappas, S. Patel, J. Rexford, H. Wright *et al.*, "A national research agenda for intelligent infrastructure," *arXiv preprint arXiv:1705.01920*, 2017. [Online]. Available: http://cra.org/ccc/resources/ccc-led-whitepapers/

[2] Networking, I. T. Research, and D. N. Program, "Smart and Connected Cities Framework," 2015, https://www.nitrd.gov/sccc/materials/scccframework.pdf.

[3] K. Nahrstedt, C. G. Cassandras, and C. Catlett, "City-scale intelligent systems and platforms," *arXiv preprint arXiv:1705.01990*, 2017. [Online]. Available: http://cra.org/ccc/resources/ccc-led-whitepapers/

[4] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, April 2005.

[5] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *Computer Communication Review*, vol. 26, pp. 5–18, 1996.

[6] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, ser. Advanced computing. Computer systems design. Morgan Kaufmann Publishers, 1999, pp. 47–48.

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: An overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, Jul. 2003. [Online]. Available: http://doi.acm.org/10.1145/956993.956995

[8] R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds., *The GENI Book*. Springer, 2016.

[9] D. D. Clark, "Interoperation, open interfaces, and protocol architecture," *The Unpredictable Certainty: White Papers*, no. 2, pp. 133–144, 1995.

[10] "Uux(1p) posix programmer's manual," IEEE/The Open Group, 2013. [Online]. Available: http://www.unix.com/man-page/posix/1p/uux/

[11] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, Nov. 1984. [Online]. Available: http://doi.acm.org/10.1145/357401.357402

[12] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and issues," RFC 3234, Feb. 2002, network Working Group. [Online]. Available: https://tools.ietf.org/html/rfc3234

[13] J. Dongarra, G. H. Golub, E. Grosse, C. Moler, and K. Moore, "Netlib and NA-Net: Building a scientific computing community," *IEEE Annals of the History of Computing*, vol. 30, pp. 30–41, Apr. 2008.

[14] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical Internet object cache," in *IN PROCEEDINGS OF THE 1996 USENIX TECHNICAL CONFERENCE*, 1995, pp. 153–163.

[15] D. Wessels and k claffy, "ICP and the Squid Web cache," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATION*, vol. 16, pp. 345–357, 1998.

[16] D. Kirkpatrick, "IBM's olympic fiasco department of groundless optimism," *Fortune Magazine*, September 9 1996. [Online]. Available: http://archive.fortune.com/magazines/fortune/fortune_archive/1996/09/09/216607/index.htm

[17] M. Beck and T. Moore, "The Internet2 distributed storage infrastructure project: An architecture for internet content channels," in *Computer Networking and ISDN Systems*, 1998, pp. 2141–2148.

[18] R. Buyya, M. Pathan, and A. Vakali, Eds., *Content Delivery Networks*. Springer, 2008.

[19] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance Internet applications," *SIGOPS Oper. Syst. Rev*, 2010.

[20] M. Beck, T. Moore, and J. S. Plank, "An end-to-end approach to globally scalable network storage," in *In ACM SIGCOMM 2002*, 2002.

[21] ——, "An end-to-end approach to globally scalable programmable networking," in *Future Directions in Network Architecture*. ACM Press, 2003, pp. 328–339.

[22] P. A. David, "Path dependence: a foundational concept for historical social science," *Cliometrica*, vol. 1, no. 2, pp. 91–114, 2007.

[23] D. G. Messerschmitt, "The convergence of telecommunications and computing: What are the implications today?" *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1167–1186, 1996.

[24] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[25] M. Beck, "On the Hourglass Model, End-to-End Arguments, and Deployment Scalability," *Communications of the ACM*, vol. to appear, 2018.

[26] "Will the real end-to-end argument please stand up?" http://mercury.lcs.mit.edu/ jnc/tech/end_end.html.

[27] D. M. Ritchie and K. Thompson, "The Unix time-sharing system," *Communications of the ACM*, vol. 17, pp. 365–375, 1974.

[28] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[29] J. S. Plank, A. Bassi, M. Beck, T. Moore, D. M. Swany, and R. Wolski, "Managing data storage in the network," *IEEE Internet Computing*, vol. 5, no. 5, pp. 50–58, Sep. 2001. [Online]. Available: http://dx.doi.org/10.1109/4236.957895

[30] M. Beck, D. Arnold, R. Bassi, F. Berman, H. Casanova, T. Moore, G. Obertelli, J. Plank, M. Swany, S. Vadhiyar, and R. Wolski, "Logistical computing and internetworking: Middleware for the use of storage in communication," in *In 3rd Annual International Workshop on Active Middleware Services (AMS)*, 2001.

[31] J. S. Plank, S. Atchley, Y. Ding, and M. Beck, "Algorithms for high performance, wide-area, distributed file downloads," LETTERS, Tech. Rep., 2002.

[32] H. Liu, M. Beck, and J. Huang, "Dynamic co-scheduling of distributed computation and replication," in *IEEE International Symposium on Cluster Computing and the Grid*, May 2006.

[33] M. Beck, H. Liu, J. Huang, and T. Moore, "Scalable distributed execution environment for large data visualization," *IEEE Explorer*, Nov. 2007.

[34] H. Liu, "Scalable, data-intensive network computation," Ph.D. dissertation, University of Tennessee, Knoxville, 2008.

[35] O. J. Reichman, M. B. Jones, and M. P. Schildhauer, "Challenges and opportunities of open data in ecology," *Science*, vol. 331, no. 6018, pp. 703–705, 2011.

[36] R. Kitchin, *The data revolution: Big data, open data, data infrastructures and their consequences*. Sage, 2014.

[37] S. S. Board, N. R. Council *et al.*, *Landsat and Beyond: Sustaining and Enhancing the Nation's Land Imaging Program*. National Academies Press, 2014.

[38] I. Bird, "Computing for the Large Hadron Collider," *Annual Review of Nuclear and Particle Science*, vol. 61, pp. 99–118, 2011.

[39] M. Breeding, "Building a digital library of television news," *Computers in libraries*, vol. 23, no. 6, pp. 47–49, 2003.

[40] M. Asch, T. Moore, R. M. Badia, M. Beck, P. Beckman, T. Bidot, F. Bodin, F. Cappello, A. Choudhary, B. R. de Supinski, E. Deelman, J. Dongarra, A. Dubey, G. Fox, H. Fu, S. Girona, M. Heroux, Y. Ishikawa, K. Keahey, D. Keyes, W. T. Kramer, J.-F. Lavignon,

Y. Lu, S. Matsuoka, B. Mohr, S. Requena, J. Saltz, T. Schulthess, R. Stevens, M. Swany, A. Szalay, W. Tang, G. Varoquaux, J.-P. Vilotte, R. W. Wisniewski, Z. Xu, and I. Zacharov, "Big data and Extreme-Scale computing: Pathways to convergence – toward a shaping strategy for a future software and data ecosystem for scientific inquiry," *The International Journal of High Performance Computing Applications*, vol. 32, pp. 435–479, July 2018.

[41] "Project Loon," https://x.company/loon/.

[42] "Facebook takes flight," https://www.theverge.com/a/mark-zuckerberg-future-of-facebook/aquila-drone-internet.

[43] S. Banerjee and D. O. Wu, *Final report from the NSF Workshop on Future Directions in Wireless Networking*. National Science Foundation, 2013.

[44] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[45] H. Avron, P. Maymounkov, and S. Toledo, "Blendenpik: Supercharging LAPACK's least-squares solver," *SIAM Journal on Scientific Computing*, vol. 32, no. 3, pp. 1217–1236, 2010.

[46] P. Drineas and M. W. Mahoney, "RandNLA: Randomized numerical linear algebra," *Communications of the ACM*, vol. 59, no. 6, pp. 80–90, 2016.

[47] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, pp. 1134–1142, 1984.