Deployment Scalability in Exposed Buffer Processing

Micah Beck
Min H. Kao Dept. of Electrical Engineering and Computer Science
University of Tennessee
Knoxville, Tennessee, USA
Email: mbeck@utk.edu

Abstract—**Deployment scalability was introduced to capture a very general notion that is often a goal of shared infrastructure. It refers to the ability of the infrastructure's basic service to grow across many boundaries that might constrain it, through the acceptable application of resources, and while maintaining it at a standard level. In order to achieve deployment scalability the Internet architecture created a common infrastructure interface, or "spanning layer," at the Network layer of the communication stack. This architecture was adopted to achieve two central goals: 1) The spanning layer virtualizes the variety of local services, enabling interoperability through the adoption of a common model; and 2) it provides an abstraction that hides the complex and dynamic topology and behavior of local infrastructure, thereby restricting the ability of clients to inspect or control local resources. The central design choice of the Internet Architecture is to make end-to-end datagram delivery the "bearer service" that defines its spanning layer. By contrast, Exposed Buffer Processing creates a more general platform by defining an *underlay* platform that provides the resources for implementing networking, storage, and computation. Exposed Buffer Processing implements a) a low-level "data plane" that provides fundamental persistence, transfer, and processing functionality and b) a higher-level programmable "control plane" that defines a variety of more global services.**

The Internet is often referred to as a scalable network, but there are many different variants of scalability that can apply to a network. The most intuitive is node scalability, meaning that the total number of nodes can be increased by applying more resources. While it is not explicit, there is an implication that the resources applied should be close to linear so that practical resource bounds are not reached quickly. Some networks may have a hard upper bound on node scalability (as in a local area network based on a single hub with a fixed number of possible connections) and others may have resource requirements that impose practical limits (such as networks with tight latency constraints).

The notion of *deployment scalability* was introduced to capture a notion that is quite general but is often a goal of shared infrastructure [1]. It means the ability of an infrastructure's basic service to grow across many boundaries that might constrain it with an acceptable application of resources while maintaining the performance of that service at a standard level. Thus, the classical Internet architecture exhibits a high degree of deployment scalability because it can grow in the total number of nodes (up to the limit of its address space) and connections, and because it also can cross geographical, administrative and other boundaries that constrain some other models. A very different example of an infrastructure architecture with a high degree of deployment scalability is the Unix/POSIX kernel interface, which can be deployed on a wide range of devices in many different environments supporting a variety of services.

While this definition of deployment scalability is admittedly still quite vague, it expresses the goal of designing interfaces that have the tendency to be widely and voluntarily adopted, sometimes called "viral growth." Such viral growth tends to occur when the value of interoperability and code reuse outweighs the benefit of developing and supporting more specialized alternatives. Deployment scalability has enabled the Internet Protocol and the Unix kernel interfaces to dominate the fields of networking and operating systems almost to the exclusion of competitors.

A more technical characterization of deployment scalability that relates it to the classical hourglass model of layered systems has been offered [1]. The suggestion is that the intuitive characterization of the concept given above is a property of service interfaces that simultaneously support many applications and also have a wide variety of implementations. Achieving this combination of many possible applications and many possible implementations is challenging to achieve because of the impact of the Deployment Scalability Tradeoff (discussed below). Whether this more technical definition is sufficient to account in practice for the more intuitive result is an open question.

When we consider the introduction of standards in complex services and systems, it is now understood that a design that attracts voluntary adoption is far more desirable than one that is legislated through legal authority or imposed through market power. Thus a high degree of deployment scalability is an important, and perhaps a necessary attribute of any candidate standard. However, designing and reaching consensus on new interfaces that exhibit deployment scalability has proven difficult. There are two important countervailing forces: path dependence and the Deployment Scalability Tradeoff.

- *Path dependence* is "the tendency of institutions or technologies to become committed to develop in certain ways as a result of their structural properties or their beliefs and values" [2].

- *The Deployment Scalability Tradeoff* asserts that ”there is an inherent tradeoff between the deployment scalability of a specification and the degree to which it is logically weak, simple, general and resource limited [1].

In the remainder of this paper, we motivate and present Exposed Buffer Processing (EBP) as a candidate service that allows for greater generality than the Internet's best-effort datagram delivery service while also taking account of the Deployment Scalability Tradeoff (DST). EBP seeks to overcome the tendency toward stagnation due to path dependence by offering the possibility of benefits that outweigh the costs of disruptive change. It achieves the properties of simplicity and generality required by the DST through the *convergence* of services typically thought of as occupying separate technological niches: storage, networking and computation. The physical foundations of all three are combined in a common set of operations defined on a very general abstraction of persistent “buffers.” The most radical aspect of EBP is the adoption of a common abstraction of the *local* resources of nodes as the basis of interoperability in service creation. This defines a common platform that is at once extremely general—Internet best effort datagram delivery is one option among others—and also *logically weaker* (in the sense referred to by the DST) than globally defined distributed services.

I. The Internet Architecture Has Failed

The Internet architecture created a common infrastructure interface (spanning layer) at the Network layer of the communication stack in order to achieve two central goals:

- It virtualizes a variety of local services, enabling interoperability through the adoption of a common model.
- It hides the complex and dynamic topology and behavior of local infrastructure by providing an abstraction that restricts client visibility into local resources.

The central design choice of the Internet Architecture is to make end-to-end datagram delivery the “bearer service” that defines its common “spanning layer” [3]. This is a “stateless” communication service, meaning that the state of each intermediate node (router) is a combination of hardware, administratively installed system software, and so-called “soft state” (e.g., cached routes) which can be recalculated. There is no state from outside the network (or “hard state”) that is specific to the traffic flowing through the node. The choice of a stateless service enables each datagram to be routed independently along a changing path through the network topology.

While stateless service is a very powerful and direct approach to topology hiding, it is not the only possible choice. Compelling arguments have been made that this design is necessary in order to achieve “deployment scalability”. However, the stateless model imposes limitations on the nature of services that the Internet can implement, and these limitations have frustrated the ambitions of many important application communities. In response, these communities, equipment providers, and carriers have developed “*ad hoc* workarounds” and “barnacles” [4] that have encrusted and that are compromising the Internet Architecture.

A. Point-to-Multipoint Communication

Many of the most ubiquitous and important Internet applications are based on the transfer of data from a single source to a large number of recipients. Email lists and Network News predated the Internet and became the basis of asynchronous bulletin boards, forums, and social media. A succession of file transfer and streaming mechanisms grew to comprise a large portion of mass and high transfer Internet traffic, starting with FTP, Gopher, and HTTP.

The inability of unicast datagram delivery to scale up to the level of traffic required to support simultaneous point-to-multipoint communication on a global scale has been evident since the early days of the Internet. IP multicast was developed and promulgated to meet this challenge, but it suffers from a number of impediments that limit its deployment scalability. One issue is that it requires the maintenance of flow-specific state at a tree of intermediate nodes, albeit in recomputable soft state. Another is that different protocols have been required for sparse and dense applications, complicating configuration and use. Finally, IP multicast is a synchronous communication protocol, which requires that sender and receiver interact simultaneously whereas many of the largest applications are less- or a-synchronous.

There has been a succession of responses to the application community requirement for point-to-multipoint communication. Asynchronous access to static files and objects were addressed by FTP and Web source file mirroring and then shared caching of responses. However, these approaches suffer from two weaknesses:

- they require the deployment of server or cache resources proportional to the maximum simultaneous transfers, and
- they are most effective when client requests are distributed in a manner that is informed by knowledge of the network topology.

These two factors restricted the deployment scalability of such solutions, but also strongly inclined Content Delivery Networks (CDNs) to use knowledge of and control over network topology to achieve performance and efficiency.

B. Active Services

The effectiveness of static source file mirroring and response caching began to wane with the increasing use of a number of active content technologies, including server-side scripting, stateful Web sessions, and response encryption. All of these had the effect of making each response unique, even responses to identical requests. The network service model was changing from accessing stored objects to the invocation of unique and dynamic services.

This resulted in a movement toward server replication, which allows client requests to be directed to one of a set of servers whose responses are considered equivalent. The tendency toward increasingly stateful application sessions (e.g., shopping carts) increased the need for stability in the choice of server. This gave another impetus toward application service providers asserting control over the choice of server. The concomitant need for coherence between application servers in turn increased the degree of connectedness and synchronization required between servers.

These factors have led to a move from distributed Content Delivery Networks to more highly centralized Cloud Data Centers and Cloud Networks connecting them. The core of these application server networks may or may not use the public Internet, and they are increasingly implemented using CDN- or Cloud-specific interconnects. The emerging application infrastructure uses HTTP over TCP/IP for client access compatibility; but it delivers services over an architecture that exposes the lower layer topology of the wide area network to service providers, emanating from a core network that may be completely private and proprietary. *In other words, the uniform interoperability of the Internet Architecture has been abandoned.*

II. Splitting the Internet Atom

A number of architectural modifications or alternatives have been proposed to enhance the functionality of the Internet by inserting so-called “middleboxes.” A middlebox is a programmable device with storage that lies along the data delivery path [5]. A related approach is the creation of overlay networks whose intermediate nodes are servers that use the Internet Protocol between themselves and for edge access.

These trends have resulted in the loss of interoperability and deployment scalability in the implementation of important services, such as commercial Web sites and multimedia streaming. One important aspect of the lack of deployment scalability is the cost and difficulty of implementing such overlays. Consequently they are being built and operated by powerful corporations to support profitable services, but are not widely implemented at reasonable cost or operated in the public interest.

Another approach to the convergence of storage, networking, and computation is possible, through the definition of a common *underlay* platform that provides all of these resources. This suggests an architecture that is implemented as 1) a low-level “data plane” that provides fundamental persistence, transfer, and processing functionality, and 2) a higher-level programmable “control plane” that defines a variety of more global services. These layers correspond loosely to the Link and Network layers of the Internet stack, but with the data plane being generalized to include persistence and processing and the control plane supporting heterogeneous services.

In this model, the data plane is the common spanning layer that provides interoperability (portability) in the implementation of services. Furthermore, the control plane has full visibility of the resource topology of the data plane and can make choices on a per-service basis about how and to what degree that topology is exposed to its clients. We refer to this separation of virtualization (definition of a common spanning layer) from topology hiding (on a per-service basis) as “splitting the Internet atom” (see Figure 1).

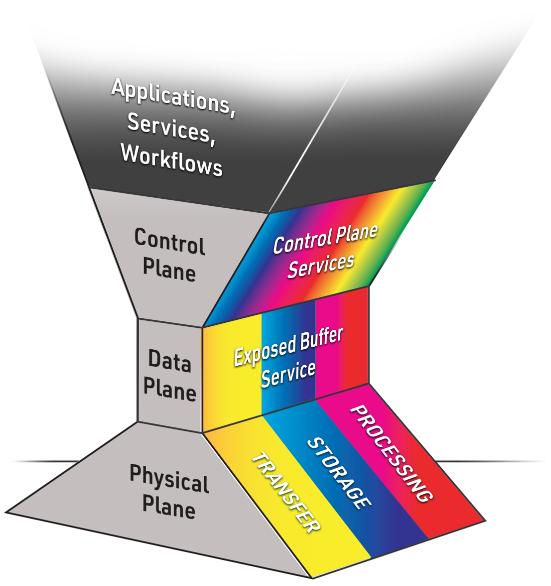

Fig. 1. The EBP Layered System ”Anvil”.

It is clearly *possible* to adopt a common model of the data plane as a basis for low-level interoperability and then to use it to implement variations of or extensions to the Internet (or to define other very different services) in the control plane. One class of such possibilities is content-addressed services [6]. However, the suggestion raises a number of questions about whether this is a good idea. Important issues that immediately arise include performance, security, and deployment scalability of the architecture.

Performance is the most difficult of these issues to address in a “paper architecture.” The current peaks of Internet performance have been achieved through specialized and optimized engineering enabled by decades of investment and economies of scale. Even so, there are some aspects of more-specialized circuit-based communication (which preceded the Internet) that the Internet has never matched, as is evidenced by pixelation and buffering in the delivery of multimedia content. The degree to which a platform based on interoperability at the data plane and heterogeneity in the control plane could eventually rival the performance of dedicated Internet infrastructure is unclear. Security and deployment scalability can be more directly analyzed in terms of the design.

III. Weakness is a Virtue

As the experience of the Internet illustrates, shared infrastructure presents a challenging design space. Adopting stateless datagram delivery at the spanning layer provides both interoperability and topology hiding, but it also rules out the use of storage and processor resources at intermediate nodes in support of important services. However, decades of analysis and discussion, informed by experience and reason, have led to the formulation of the Deployment Scalability Tradeoff (DST), a principle that can help in navigating that design space.

The DST tells us that there is a correlation between the deployment scalability of a spanning layer and the degree to which it is logically weak, simple, general, and resource limited. The DST is informed by classical analyses of the Internet Architecture, such as that of Saltzer, Reed and Clark in“End-to-End Arguments in System Design” [7], and by reflection on other layered systems, including the Multics/Unix tradition of operating system design.

We can relate the DST to the design of the Internet's end-to-end datagram delivery architecture as follows. A datagram delivery service is simple and general, but in aggregate a flow can represent a significant allocation of network resources. The imposition of a size limitation on individual datagrams (the Maximum Transfer Unit) by each router is a means by which the utilization of link bandwidth and router forwarding resources can be interrupted, facilitating (but not enforcing) fair contention. However, *logical weakness* in the design of the Internet's datagram delivery service is the main tool for ensuring its deployment scalability.

Logical weakness is a concept that may not be intuitive to some readers. The “logical” aspect refers to the fact that the specification of a service (the API) *can* be expressed in terms of a system of formal logic, but such a formulation would also be very detailed and difficult to apply. Thus we typically consider most commonly used API specifications to be less-formal approximations to such a logical description.

Within formal systems, the notion of “logical weakness” follows directly from the definition of implication. In less formal terms, logical weakness can simply be thought of as ”assuming less” or ”making fewer guarantees”. This intuitive meaning of weakness is familiar to software designers.

The Internet datagram delivery service is weak because it makes very few guarantees. There is no upper bound on latency (delivery time) or on the variance of latency between endpoints over time or on the probability of in-order or eventual delivery. There are no guarantees of privacy or non-malicious behavior on the part of endpoints or intermediate nodes. There is no guarantee of stability in forwarding decisions or of persistence in the state of intermediate nodes along a path. Many suggested extensions or modifications to the Internet spanning layer have been vetoed essentially (but not explicitly) because they increase the logical strength of the network service specification.

Some stronger guarantees have been adopted within the Internet community even though they are not strictly speaking part of the architecture. For example, it is expected that delivery latency will be sufficiently limited and predictable to enable effective end-to-end signaling between endpoints for the purposes of fault tolerance and flow control. Erratic or malicious behavior by endpoints or intermediate nodes may result in blocking by manual or automated security mechanisms. Such stronger terms of service may be agreed on within an Autonomous System or in peering Service Level Agreements, but are not required universally. This has resulted in some loss of interoperability at the Transport and Application layers of the Internet stack, and has slowed the penetration of widespread Internet connectivity in parts of the world with infrastructure challenges.

IV. Local is Weaker

One reason to insulate Internet clients from Link Layer characteristics (such as connectivity topology) is the volatility of such information. A weak service that exposes highly volatile information is of limited utility, and this can lead clients astray if it is interpreted as if it were stronger. Virtual topology metrics, such as the round trip time of Network Layer datagram delivery are inherently quite weak but are somewhat predictable over short periods of time.

Many sophisticated high volume services require access to low-level resource topology in order to apply policies and optimize the allocation of resources. CDNs and Cloud networks do use acess to network topology to better meet application requirements. This approach strengthens the specification of such overlay networks and makes them expensive and difficult to operate, limiting deployment scalability.

One way to reduce the weakness of a service without weakening or eliminating any guarantees is to confine it to a locality. This is not immediately intuitive in all contexts but can be seen in a simple example:

> A service S defined on the nodes of a network consisting of 100 nodes N_0 ... N_{99} delivers messages reliably between any two nodes N_i and N_j. Now, consider another service S' defined on the same set of nodes whose specification states that messages are delivered reliably only between pairs of adjacent nodes N_i and N_{i+1}. Everything that is true of S' is true of S, in other words, S' is weaker than S.

This principle can be applied to explain why a service like reliable multicast can have a substantial degree of deployment scalability within LANs even if it has properties that are too strong to enable wide deployment scalability in a global network.

The notion of deployment scalability in a local service may seem counter-intuitive, since there may be no requirement that different localities interoperate, in contrast to a global communication network. Interoperability in local *non-communication services* is commonly called *portability*, and portability is the basis of common software platforms such as POSIX.

V. Local is More Secure

One concern about programmable and stateful network services is that they enable more powerful attacks than stateless transfer networks that can leverage additional resources. Within the limitations of our communication network, it is clear that a more local service is vulnerable to fewer denial of service attacks by a single malicious node than the globally connected one.

A data plane service that does not implement global connectivity must rely on control plane services to connect localities. This means that the responsibility for ensuring a level of security appropriate to a control plane service is the responsibility of that service. Some services may require very little security because they present a minimal attack surface. Others may require very strong security mechanisms and will suffer in their deployment scalability as a result. Some strong services, such as the classical Internet's unrestricted global datagram forwarding, may be deemed too dangerous for the control plane of very sensitive nodes to deploy.

This may seem to some to be reminiscent of the Internet's end-to-end approach that leaves security to the application layer and which has resulted in today's disastrous cybersecurity environment. There are two important differences to consider:

- end-to-end Internet security requires every application to handle traffic from any node connected to the global Internet, and
- the implementers of the control plane are fewer and must be more technically sophisticated than the implementers of many Internet applications.

The intent is that judicious care taken in the design and implementation of control plane services can enable more secure distributed environments, even if it does not ensure them. It enables (but does not require) every control plane node to act as a service-specific firewall.

VI. Exposed Buffer Processing

Exposed Buffer Processing (EBP) is a converged approach to shared infrastructure whose design is based on the Deployment Scalability Tradeoff (DST) [8]. The EBP architecture can be implemented using the current Internet stack, although not adhering to all of its original architectural principles (in the same way that Content Delivery Networks do not). The current implementation of Exposed Buffer Processing is the Internet Backplane Protocol, which is a full overlay implementation that does adhere to Internet principles (specifically, isolating overlay routing from Link Layer topology information and mechanisms) [9]. There is an evolutionary path from IBP to more native forms of EBP that access lower layers of the Internet stack which is beyond the scope of this paper. Alternatively, it is possible to implement a form of EBP directly on the local layer (a so-called ”clean-slate” approach) but this presents deployment challenges in current infrastructure environments.

IBP is encapsulated as a set of RPC-over-TCP calls, and so it does not have the inherently local characteristics of EBP (although these can be imposed using firewalls or SDN routing). The core functionality of EBP is implemented in a few simple and general calls:

- **Allocate** a buffer on a specific node, specifying the size and duration explicitly. The client receives capabilities (random keys) that are the only names by which the allocation can be referenced and which also implement per-allocation access control.
- **Write** data to a buffer or **read** data from a buffer. This allows clients which are applications to implement higher level operations using the data plane directly.
- **Transfer** of data between two buffers on one data plane node or between two nodes by specifying the keys of both the sending and receiving buffers. These data plane nodes must be adjacent (reachable through the connecting network).
- **Transform** data stored in a set of buffers on a single data plane node by invoking the execution of an operation on that node. Such operations are bound to a global namespace and must be implemented by the node through some mechanism other than invocation. This means that operations need not imply the use of on-demand or mobile code mechanisms.

All of these calls and the services they implement are best-effort, meaning that they make no guarantees of completion, integrity, or correctness, either immediately or over time. All resource allocations, including buffer space, transfer bandwidth, and computational resources, can be capped by the operator of the data plane node (in analogy to the network Maximum Transfer Unit). While data plane nodes may implement strong services and provide guarantees, all knowledge of and control over network QoS is implemented by control plane services.

A. Heterogeneity at the Network Layer

The idea of heterogeneity at the Network Layer has usually been considered in the conventional context of globally connected data transfer networks. Since the Network Layer is traditionally responsible for global forwarding of datagrams, then one way of enforcing interoperability in communication is by placing the spanning layer there. Adopting this approach also locks into place the assumption that communication is implemented only through the forwarding of datagrams.

In the context of Exposed Buffer Processing, the Control Plane is responsible for a variety of potentially non-local services, of which global forwarding of datagrams is one. From this point of view, the delivery of datagrams could be a means of accessing services other than data transfer (e.g. remote procedure call) and data transfer could be implemented through mechanisms other than datagram delivery (e.g. access to named objects). The possibilities are endless, and they may not interoperate. The coherence that has been enforced by placing the Internet Protocol suite at the thin waist of the communication stack is not enforced by the more general platform architecture.

There are two important points to consider in evaluating such a possible future.

- If it is in the interest of the community, coherence can be imposed by mechanisms other than constraining the network architecture. An example of such a development has been the limited heterogeneity that has occurred at the Transport Layer of the Internet stack. The perceived value of ”TCP-friendliness” to the community led to the maintenance of tight control over the approval of new Transport Layer protocols by the IETF, which in turn has governed their use on public networks. The authority under which such constraints have been applied is not legislated but community-based.

- The constraints placed on communication have resulted in the development of middle boxes and overlay networks that leverage standards for edge access but implement strategies other than end-to-end datagram delivery in the core. Such “barnacles” have had the effect of leading to the private and proprietary implementation of the core network, putting it out of reach of any but the largest commercial vendors who then sell access to their clients on terms that they control.

B. Homogeneity at the Local Layer

It is only possible for the community to assert control over issues like interoperability in communication and other services if there is a shared platform. The variety of potentially non-local services is so large that current efforts to define standards at the node level are constrained by a perceived need for specialization. Adopting a common model of resource management at the local level would create a highly generic model of a processor-based device, enabling a much greater degree of commonality in the low-level software.

The idea of adopting a common device model can be found in many domains when the communities that use those devices value interoperability. Historically, the computer industry has tended to adopt de-facto device standards at different levels. The earliest may have been the IBM System/360 architecture, which dominated commercial mainframes for decades. The IBM PC architecture followed in the emerging area of personal devices.

The Java Virtual Machine (JVM) strategy can be seen as the most ambitious attempt to create a common machine model to supplant physical system architecture as the basis of interoperability. Java was originally developed in Sun Microsystems’ Oak project as a model of set-top box functionality, where the primary goal was interoperability across physical devices, rather than high levels of performance or innovative human-machine interfaces. It was adopted as a means of projecting edge computation into Web browsers where interoperability was similarly central. However, the JVM was redirected as the central mechanism of a strategy of replacing the PC model for desktop applications, and later on edge and embedded devices of all kinds. In service of this goal, the JVM was extended with specialized packages and libraries that increased its complexity, reduced its stability and security, and reduced its appropriateness as a universal device model.

Another striking development toward a common machine model has been the near-universal adoption of the Intel x86 Instruction Set Architecture (ISA) as a low-level model of computational resources. This model is not only highly local—it is restricted to the processor control logic, datapath, and register and address translation—it is also quite general. But its complexity has resulted in a concentration of device production on a small number of industry players working from the same templates. A form of portability typically predicated on x86-based virtual machines and low-level storage and connectivity is Infrastructure-as-a-Service (IaaS): ”The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.” [10]

Adopting a common model of system architecture (buffer management, local data transfer, and invocation rather than a detailed definition of operations) would create interoperability in low-level system management functions. This would not determine the ISA of instruction execution (or require that a single ISA be chosen). It would also support innovative models of processing, including functional and highly data parallel models that can be supported by Graphics Processing Units, Field Programmable Gate Arrays, or other accelerators, within a common model of buffer control and management.

There is a long track record of open source communities adopting low-level system models including processor and buffer management. The primary example of a community where a common platform has been unifying is operating system community, which came together on the adoption of the Unix/POSIX operating system kernel. While the OS is a higher level model than low-level buffer management, the development of a minimal and orthogonal kernel interface enabled a revolution of interoperability in application and service creation software. The POSIX model now forms the basis of program portability from mobile phone apps and embedded devices to personal computers and storage and computer clusters. The POSIX model is the basis of the movement toward OS-level interoperability using Linux containers [11].

A better analogy to interoperability in low-level system functionality can be found in the emergence of open source router firmware (e.g. OpenWRT) and file servers (e.g. Luster, OpenNAS). A choice of processing models is much more relevant when alternative and new ISAs can be developed, so the emergence of open source processor cores is also relevant (e.g. RISC-V). Other examples of common system models developed commercially in order to create proprietary communities of interoperability include Roku and Amazon Alexa in the home sector.

VII. Examples

The potential for service creation generality offered by the Internet architecture to distributed applications that build it was not fully appreciated by the Internet’s contemporaneous critics, and perhaps not even by its creators. Minimal design may enable the construction of new service ecosystems supported by interoperable infrastructure, but it does not actually bring them into being. To better explain what EBP would mean for service creators, we present two examples of how it can be used to address current problems in the design of distributed services.

A. File Distribution Systems

File systems are a classical example of the encapsulation of the management of primitive buffers (typically disk or SSD blocks) within a much more complex and heavyweight file abstraction. A file is not just a collection of data blocks, but a complex data structure (e.g., the Unix inode) maintained on disk and in memory. The implementation of directories layers even more structure and mechanism on top of the file abstraction. File systems do not operate by simply writing data provided by client read and write operations into storage blocks. Complex data transformations, including encryption, compression, and RAID encoding require that computation be applied, sometimes intensively. The management of concurrency between simultaneous users of files and their system resources requires sophisticated concurrency control and fault tolerance of a stack of complex mechanisms. In large high performance file systems, the use of burst buffers and tertiary storage creates a distributed state even within a data center.

The problem of implementing file access for a community distributed across the wide area network has in the past been addressed in one of two ways:

- In distributed file systems, the control mechanisms of the file system are extended across wide area network links, generalizing the communication and coordination that applies in local area networks. This creates a single extended domain within which resources and operations are scheduled.
- In federated file systems, an overlay is created which uses the high-level client interfaces of multiple local file systems (perhaps of different designs) and presents a unified service interface to its own clients. This requires the translation of a single overlay model onto multiple local systems, taking account of the challenges of spanning administrative domains and wide area resource topology.

While there are many challenges to each of these approaches, they have in common the composition of the file system silo and the Internet architecture, which hides communication topology. Typically, this is addressed by leveraging the fact that the number of localities is small, that they are widely separated, that their separation within the network does not change much and is known to system administrators, and that they do not inhabit competitive or adversarial domains. These strong assumptions allow topology to be managed manually by system administrators or even by end users. Unfortunately, these assumptions do not generally hold as the scope and reach of such systems are scaled up.

Content Delivery Networks address this issue with distributed file systems in two ways:

- By specializing the distributed service to particular remunerative applications (e.g., Web sites, shopping, and streaming) that fit their business model, and
- By taking responsibility for knowing and even controlling the topology of the server and access networks (e.g., by use of topologically sensitive DNS resolution).

Exposed Buffer Processing enables the design of a file distribution system that is implemented at least partially in the control plane, using disk blocks, network buffers, and process pages that are all modeled as interoperable buffers. Implementation in the control plane means that the resource topology describing storage availability, network characteristics, and processor load is fully visible. Furthermore allocation of and movement between these different primitive resources can be directed by control plane services at a very low-level and at a very fine granularity.

The construction of a robust and scalable file distribution system is a complex problem that works by building up many layers, both within the control plane and at the application layer. An important aspect of this layering is that the resource and information sharing policies applied at every level differ. The lowest level services in the control plane make use of the local resources of nodes, but they can also define the terms of connectivity. No ”default” datagram forwarding model is imposed on every service, although services are free to adopt common protocols if they meet their needs.

A good example of the importance of enabling specific services to control policy within their own stack is illustrated by the regulation of storage services across the wide area in the Millennium Copyright Act [12]. In that law, store-and-forward communication networks are allowed to forward copyrighted files at will but can only store them under specific stringent conditions. A provision allows networks to store files, and even cache them, but not for an ”unreasonably long” period of time. This presents a challenge for publicly shared infrastructure, such as a file distribution system.

These issues arise in the adaptation of institutional policy to the creation of distributed data facilities across academic, research, and other public institutions. Each locality is legally responsible for the data it manages, and each has its own set of policies and procedures. Networks can span institutions because they are largely immune from the application of policy due to their highly constrained functionality. The layered approach to the construction of data sharing services enabled by Exposed Buffer Processing allows for the application of a graduated approach, with specialization and limitation being added to the higher layers, while the lower layers are more directly informed by topology and reflect local policy.

B. *Edge Data Analysis*

The “data deluge” is creating huge aggregate data sources located throughout the network, including edge devices such as sensors and cyber-physical systems. The classical compute center model requires that this data be moved to centralized locations and ingested into storage systems before it can be processed by applications and generate responses. There is a broad consensus that edge computing is necessary to reduce the volume of data through summarization and analysis as well as to localize some control loops and enable timely responses.

One difficulty in projecting computation to edge environments is the process-oriented model of computation. In conventional operating systems such as Linux, a large collection of operating system code and data structures must be present in a node as well as the specific user code and data and any files that will be acted on. This operating system and application state is comprised of storage blocks and memory pages as well as any network buffers required for communication and coordination during execution. Typically, all of this state must be accessible on the local node in order for the system to function.

There are a number of strategies for creating and using the amount of state that edge computing requires:

- One approach is to place a minimal hypervisor in the node and require transfer of the entire operating system and application processes (Infrastructure as a Service). This minimizes long term management of state at the edge node but creates a burden on the capacity and stability of the edge network. It also does not provide an obvious way to store data that has a lifetime beyond the duration of the hypervisor image.
- A different approach (Linux containers) is to install the operating and file system state at the edge node more permanently, perhaps including long-lived application files and long-running processes. This increases the burden of managing the node and may make sharing among applications more complicated.

Exposed Buffer Processing enables a different approach, which begins with the observation that the application that *must* execute in the edge environment may be very limited, and may have specific resilience requirements and characteristics. For instance, a sensor may require a monitor that calculates a moving average which can then be periodically relayed to a more central node for analysis and correlation with other sensors. The average itself may be a relatively simple and limited operation, performed on an input buffer and an ”accumulator” buffer which holds a summary of stream state, producing a new value for the accumulator. The average may be robust to the occasional loss of input data, and it may be possible to recover even from loss of the accumulator buffer by refreshing from a recent checkpoint.

In this scenario, it may be possible to schedule the averaging operation, the flow of data, and the checkpointing all from a control plane node that is connected to a number of sensors, requiring no presence of the control plane functions in the sensor-adjacent node. If particular aspects of the control loop must be localized at the most remote point of the edge environment to minimize latency, it may be possible do so by deploying a very limited control operation on the edge node, and monitoring and managing it as necessary from a more stable but less proximate node that implements more complex control plane functionality.

The approach just described could work for very lightweight edge functions that have very weak operating environment requirements, but what if a more conventional virtual machine is required to execute a stored program? A stored program is simply a buffer that presents one input to an interpreter. That interpreter may be a natively executed Instruction Set Architecture, but it may also be a Java Virtual Machine or a P4 accelerator. Such more substantial execution mechanisms may be installed at edge nodes, and their state requirements can be managed either by remote managers operating in the control plane or by localized managers that actually run in the edge node. Nothing in the EBP architecture rules this out. But creating a more conventional virtual machine that requires a much greater localization of state in buffers local to the node does increase the complexity of the system. Certain strong characteristics of highly localized computational nodes may be impractical or deemed not worth the effort. Greater generality of implementation may have to be achieved at the cost of certain application functionality, as described by the Deployment Scalability Tradeoff.

The reader may find that this description of how computation can be assembled out of storage, memory, and communication buffers to be unfamiliar. By way of contrast, the construction of the silos hides the components that comprise their high-level functions, enabling users to view them as opaque components. At the end of the last century, users of communication circuits may have been similarly surprised by the complexity of the TCP protocols that endpoints were forced to implement when using the best effort IP datagram network. Today, most users and application developers take the correct functioning of TCP for granted, as it is once again hidden, but now by layers of software at network endpoints. Encapsulation is a tool which simplifies the support of specific functions, but it need not be a straightjacket that restricts innovation.

VIII. Validation

Validation of the design of wide area infrastructure is a very challenging topic. This is due in part to the goal of deployment scalability, which is defined in terms of the ultimate acceptance of the design and its wide utility in application domains not necessarily anticipated in advance. It is also due to the fact that wide area information infrastructure tends to be a highly exclusive undertaking. While successive epochs may overlap, there is generally a single dominant model at any given time. During the period when telephone and radio frequency broadcast media were dominant, the idea that packet networking would arise as a force for convergence that would ultimately dominate was widely unthinkable. In the current period of packet networking and Cloud data centers, the idea that further convergence could arise from the convergence of storage, networking, and computation silos may seem similarly implausible to many.

Various aspects of the proposed Exposed Buffer Processing model have been validated in a number of different ways. As described above, the Internet Backplane Protocol and Logistical Networking are overlay implementations of Exposed Buffer Architecture which have been successfully used to build a wide variety of different services and to display experimental performance in overlay, as described elsewhere [8], [13]. However, that record of success was achieved over the past 15 years, so the implementation details and performance levels do not match the current vision of EBP as a unifying mechanism at a much more fundamental level. It may seem a lot to ask of a field that advances so rapidly in implementation technology to extrapolate the lessons of past decades to design for the future. Reestablishing the fundamental validity of the model experimentally would take a development effort comparable to the original overlay implementation– millions of dollars applied in a multi-year project. Technological tradeoffs do not occur on a timetable, and design insights gained in a previous era may remain relevant even as times change.

Another mode of possible validation is the application of formal reasoning to the design process. In the case of EBP, a search for the formal basis for End-to-End Arguments [7] led to the formulation of the Deployment Scalability Tradeoff, the core of which is expressed in the Hourglass Theorem [1]. Expressed in the language of Program Logic, this theorem provides a theoretical basis for the intuition of operating system and network designers that the ”weakest” common interface should be used to support a required set of applications while also maximizing deployment scalability. Such an abstract foundation may seem inaccessible to some infrastructure architects, but it is supported by many historical examples taken from as the design of the Unix operating system and the Internet stack. Of course, there may be doubts about whether abstract principles that help explain the historical evolution of successful and unsuccessful infrastructures are applicable as validation of a new paradigm at a different layer of the infrastructure service stack.

Ultimately, the question for the skeptical reader is whether the radical approach to convergence that we suggest is plausible. We do not ask readers to conclude that we have the ultimate protocol or implementation of the principles involved, but only entertain the idea that following this design approach has the potential to lead to a new epoch in innovation and interoperability. The question is not whether the efficacy of this architecture has been proved, but whether investigation and development in this direction deserve the attention of a community that has for decades been frustrated by the constraints imposed by their own early successes.

IX. Related Work

Exposed Buffer Architecture grew out of responses to the limitations of stateless networking. Efforts to address application demands which are not well addressed by the conventional client/server paradigm have led to a sequence of solutions from FTP mirroring to Web caching, middle boxes, and network-embedded virtual machines running on multi-tenant infrastructures: PlanetLab, GENI Racks, and most recently Network Function Virtualization. At the same time, the demand for utility storage and computing services has given rise to a sequence of solutions based on distributed data centers ranging from early online services (e.g., Compuserve and AOL) to the computational Grid and most recently the Cloud and National Research Platform. Edge and Fog services for the Internet of Things combine the functionality typically found in data centers with widespread deployment in embedded environments. Exposed Buffer Processing uses the resources of network nodes to solve problems of data logistics (combined movement, storage and computing) in a common, primitive, interoperable form rather than through the high-level services of silos. In that sense, EBP proposes a reconsideration of design choices that are at the foundations of those silos, allowing for the construction of services that blur conventional boundaries to enable flexible and efficient solutions to problems of data logistics.

X. Conclusions

Application portability/interoperability is the key to building user communities at all levels of the Information and Communication Technology (ICT) stack. The adoption of open, uniform standards is a key enabler of interoperability. However, there are many barriers to the introduction of open standards into a mature, competitive environment dominated by proprietary technologies.

Voluntary adoption of a common model due to its inherent deployment scalability is a way to overcome these barriers. Achieving deployment scalability in a low-level model of system resources is particularly challenging. The Deployment Scalability Tradeoff points to a design strategy that has proven successful in the acceptance of the two most ubiquitous service interfaces in the modern ICT environment: the Unix/POSIX kernel and the Internet. Exposed Buffer Processing is a blueprint for applying the same analysis to the development of a weak, simple, and generic model of the nodes that comprise myriad distributed systems.

Acknowledgement

The description of EBP grew out of (ongoing) conversations with Martin Swany, and this paper was greatly enhanced by the contributions of Terry Moore and Elaine Wenderholm.

References

[1] M. Beck, “On the Hourglass Model,” *Communications of the ACM*, vol. 62, no. 7, pp. 48–57, 2019.

[2] I. Greener. (2019, Jan) Path dependence. [Online]. Available: https://www.britannica.com/topic/path-dependence

[3] D. D. Clark, “Interoperation, Open Interfaces and Protocol Architecture,” *The Unpredictable Certainty: White Papers*, no. 2, pp. 133–144, 1995.

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

[5] B. Carpenter, “Middleboxes: Taxonomy and Issues,” Internet Requests for Comments, RFC 3234, February 2002.

[6] *Proceedings of the 6th ACM Conference on Information-Centric Networking, ICN 2019, Macao, SAR, China, September 24-26, 2019*. ACM, 2019. [Online]. Available: https://dl.acm.org/citation.cfm?id=3357150

[7] 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

[8] M. Beck, T. Moore, P. Luszczek, and A. Danalis, “Interoperable Convergence of Storage, Networking, and Computation,” in *Future of Information and Communication Conference*, 2019, pp. 667–690.

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

[10] P. M. Mell and T. Grance, “Sp 800-145. the NIST definition of cloud computing,” Gaithersburg, MD, USA, Tech. Rep., 2011.

[11] The Linux Foundation. (2020) Open container initiative. [Online]. Available: https://opencontainers.org/

[12] United States. Congress. Senate. Committee on the Judiciary, *The Digital Millennium Copyright Act of 1998: report together with additional views [to accompany S. 2037)*. [Washington, D.C.?]: [U.S. G.P.O.,], 1998. [Online]. Available: https://www.copyright.gov/legislation/dmca.pdf

[13] M. Beck, T. Moore, N. H. French, E. Kissel, and M. Swany, “Data logistics: Toolkit and applications,” in *Proc. of the 5th EAI International Conference on Smart Objects and Technologies for Social Good*. ACM, 2019, pp. 61–66.