Exposed Buffer Architecture for Programmable and Stateful Networking

Micah Beck
University of Tennessee
Knoxville, Tennessee 37996
Email: mbeck@utk.edu

Terry Moore
University of Tennessee
Knoxville, Tennessee 37996
Email: tmoore@icl.utk.edu

Although the movement to create a standard platform for Network Function Virtualization (NFV) was initialized in 2012 by a white paper from a vendor consortium [1], much of the intellectual impetus for the movement was provided seven years earlier by an impassioned call to "[Overcome] the Internet Impasse through Virtualization" published by leaders of the networking research community [2]. These researchers were alarmed by the fact that the current Internet architecture had become, as a result of its own astonishing success, so deeply entrenched that proposed substantial changes (e.g., for security and QoS) had become effectively undeployable in real world environments, even for the purposes of experimentation and testing. Internet Service Providers were instead being forced to deploy a diverse ensemble of "... *ad hoc* work-arounds, many of which violate the canonical architecture (e.g., middleboxes) ... in order to meet legitimate needs that the architecture itself could not. These architectural barnacles ... may serve a valuable short-term purpose, but significantly impair the long-term flexibility, reliability, and manageability of the Internet." Fifteen years later, this "impasse," or state of stagnation in network innovation, remains essentially unchanged [3].

Network Function Virtualization offers Internet Service Providers (ISPs) a way to address many issues raised by the growing menagerie of middle- and edge-box "barnacles" by implementing the functions they provide in software, thereby making it possible to combine them more seamlessly with core network functionality running on commodity hardware. Specifically, NFV creates a space within an IP router to execute processes that implement a variety of functions while maintaining the performance of packet forwarding along the fast (NFV-independent) routing path. The Exposed Buffer Architecture (EBA) described below shares with NFV the general goal of enabling greater network innovation, but takes a more radical approach to the "Internet impasse," one which we believe opens up a path for significant innovations that conform in a much more principled way to the architecture of the underlying platform.

As advocated by Anderson et al. [2], EBA begins by directly confronting one highly plausible source of the problem, namely, the specific virtualization that is *inherent* to the Internet architecture. In the Internet stack below the Network Layer, the Link layer models services that are local to network nodes, or that connect them in local area networks. Aggregating these low level resources in the implementation of IP to create wide area services serves two different purposes:

- It virtualizes the 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 premise of EBA is that we can separate these two tasks. First we define a *local* virtualization of lower layer services and resources (including storage and processing) to enable interoperability. Then this local virtualization enables heterogeneity of higher level services, to be defined at the Network layer. This preserves interoperability of implementation while avoiding the "ossification" that comes from imposition of a uniform abstraction at the Network layer; global forwarding becomes a choice, not an intrinsic necessity.

As its name suggests, the design of the Local layer virtualization on which EBA builds focuses on the *buffer*, a data persistence resource of limited size that captures values generated by adjacent devices within a system and holds them until they can be delivered to others (perhaps with very low latency). Buffers are the fundamental building block of all sequential digital circuits. Familiar higher level abstractions, such as data storage, movement and computation, are implemented using them. Thus EBA can account for all of the resources of the network node, including the storage and processing resources (normally excluded in representations of the "network stack") which are used for system configuration and the calculation of forwarding routes. (see Figure 1). The hypothesis on which EBA is based is that higher level operations can be implemented with the greatest level of flexibility and efficiency by expressing them in terms of primitive buffer operations and then aggregating them explicitly to create higher level services. By exposing storage and computational resources that are now used to implement the network, but which are hived off from alternative uses by traditional mechanisms and policies, EBA creates a converged infrastructure model in which transfer, persistence and processing functions can be combined freely to enable a wide variety of programmable and stateful services and protocols.

The alternative is the current approach: construct higher level abstractions that encapsulate their constituent buffers and operations on them and then compose those abstractions in overlays or middleboxes (either physical or virtual). By enabling network layer datagrams to be inspected and modified by services at intermediate nodes, NFV seeks to provide routing configurability, programmability and stateful services by marrying end-to-end datagram delivery with some form of process management.

As we will describe in this paper, EBA is a much more direct approach which uses that fact the datagrams are not physical entities but abstractions (distributed processes) that aggregate local operations on local buffers along a path. Services defined on intermediate nodes are also abstractions of operations performed on memory pages implemented through instruction execution. NFV is an effort to deal with the separation of ICT silos. EBA constructs those from a small number of general and interoperable of buffer operations. Defining an infrastructure in terms of buffer operations rather than higher level silos of storage, networking and computation provides a more direct path to convergence of ICT silos.

I. Background

Modern digital communication descends from telephony, which had the circuit as its central abstraction. Originally, a telephone circuit was actually a conductive path that carried an analog electrical signal end-to-end from microphone to speaker. A succession of developments virtualized this physical model, starting with the introduction of switches and amplifiers, leading to digitization and packetization. The transition from telephony to Internet communication exposed end-to-end packet delivery as a more general and flexible abstraction through the use of which a variety of wide area communication services, including telephony, can be implemented.

Modern computing has followed a similar trajectory, starting with analog devices whose control was virtualized through digitization and then managed through timeslicing. The original model of obtaining the use of an entire dedicated machine lives on in the modern abstraction of an operating system process. Like virtual circuits, processes are implemented by aggregating smaller resource-limited units (timeslices) managed to create a legacy abstraction.

Exposed Buffer Architecture provides an approach to converged communication and computing by taking advantage of the commonality in their implementation. If we analyze the implementation of the process abstraction, we find that it consists of the management of state stored in buffers of different size and characteristics (e.g., registers and pages). From this point of view, packets delivered end-to-end and processes that transform data on a single node are closely related, and can be unified by expressing them both in terms of a generic buffer abstraction.

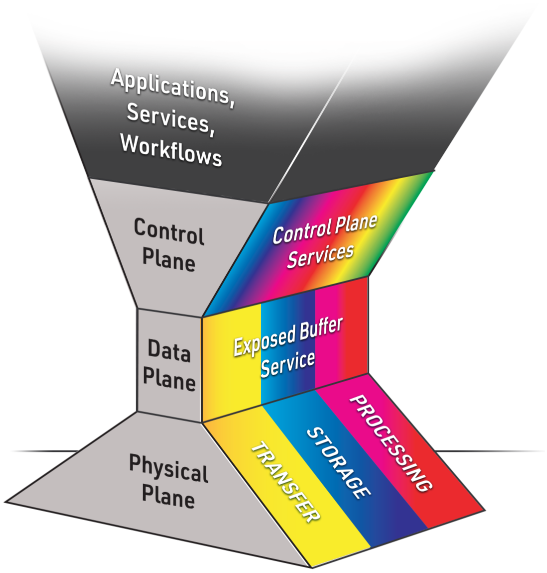

Fig. 1. The EBP Layered System ”Anvil”.

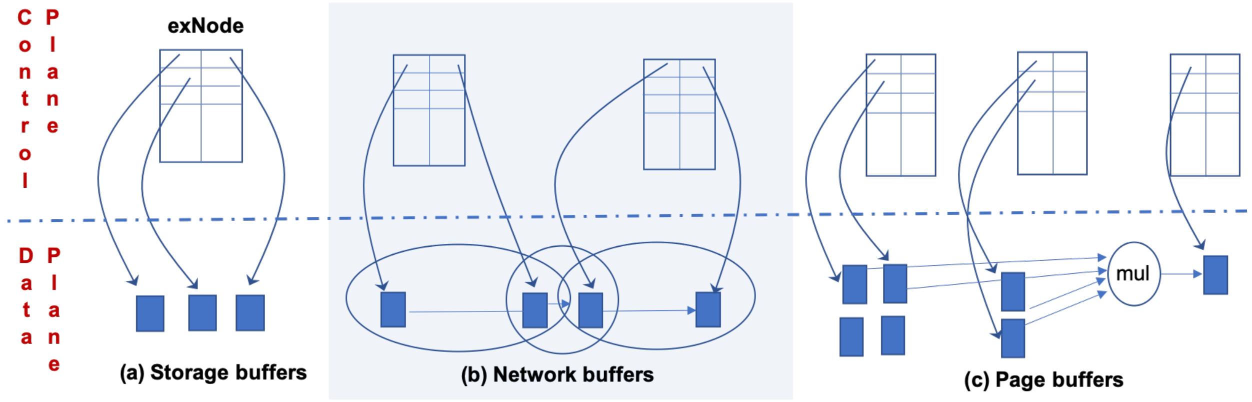

Fig. 2. Structural metadata in the EBA control plane uses a common interface to buffers in the EBA data plane to manage diverse operations.

II. Exposed Buffer Architecture

The fundamental concept in Exposed Buffer Architecture is that all computer systems can be expressed as the aggregation of operations on a generic abstraction of persistent data buffers. Such buffers have many implementation technologies, different sizes, are connected to other buffers in different ways and have a variety of transformative operations defined on them.

However, the systems can all be described in terms of a small set of primitive functions:

- allocation of a buffer,
- storage and retrieval of data in a buffer over time,
- transfer of data between locally connected buffers, and
- transformation of data in a set of buffers that have some common locality

The smallest buffers typically considered in the design of large scale systems are single bit cells, and the largest are contiguous storage extents consisting of gigabytes. Connectivity ranges from the inputs and output paths of FIFO buffers to highly interconnected local area networks. Storage duration might be measured in microseconds or in years. Transformation operations range from bitwise Boolean functions to complex pattern recognition. All can be described in terms of a uniform abstraction and many can be implemented using interoperable mechanisms, (subjects to performance constraints).

From this point of view, a datagram is characterized as the transfer of data from an originating buffer, through a series of buffers either internal to network nodes or connecting buffers in adjacent nodes, to eventually reach a destination buffer. Forwarding is a local operation that connects input and output buffers, or connects those buffers to the memory pages of the router.

Computation and processing, on the other hand, can be characterized as the local transformation of a set of buffers that implement the pages of a process address space, including the stored program. Some of those buffers may reside in an operating system, and may be shared in the implementation of multiple concurrently active processes. Other buffers are special purpose registers or memories (e.g., caches)connected directly to fixed datapaths and control functions. Such specialized and optimized implementation interferes with interoperability, but these elements are still recognizable as buffers.

Conventional implementation strategies distinguish datagrams in flight from the pages of an executing process in ways that then require complex workarounds to allow them to be combined. In Network Function Virtualization, a datapath is created from the datagram forwarding path to the address space of processes implemented using timeslices of a conventional ISA. These timeslices implement complex compound operations on the state of the process and the datagram. In the NFV model, the aggregation of those timeslices implements a virtualization of the network node.

NFV implements a restrictive datapath in order to maintain the performance of one function, datagram forwarding, while enabling the possibility of additional network processing. Exposed Buffer Processing is an approach that first maximizes generality and interoperability and only then considers how performance can be achieved, either through the use of optimizations that leverage converged network resources (data logistics) or by optimization and possibly acceleration along the fast datagram forwarding path.

III. Exposed Buffer Processing

Exposed Buffer Architecture‘ [4] adopts a model in which resources that implement persistence and transformation of buffers local or adjacent to a node are considered as resources of the physical layer. The layer above the physical layer (a generalized ”local layer”) implements a virtualization of those general resources through a minimal API or local area protocol. Within Exposed Buffer Architecture, a datagram is modeled as a set of buffers that are allocated along a path of nodes connected by local area hops. Forwarding is modeled as a transformation of the datagram header that results in a forwarding choice which is then the basis of an update transformation to the header and a buffer transfer across a single hop. The buffers along the path may be allocated dynamically (FIFO buffering is interpreted as a highly transient form of buffer allocation) or persistently (as in reservation for the purpose of QoS).

Within Exposed Buffer Architecture, a process is modeled as a set of buffers (pages) that can be transformed by computation on the nodes where they are allocated. Transformations may represent operations within inputs and outputs (potentially functional) that are executed in dataflow fashion or following order specified by program dependences. If a process is expressed as a stored program augmented with operating system services (e.g., a POSIX process) then the pages of the program are modeled as buffers and the operating system state are additional buffers that are accessible only through the intervention of the control plane.

Within Exposed Buffer Architecture, a file is modeled as a set of buffers (storage objects) that are allocated from within volumes located at nodes where data is placed. Data encoding for error detection and correction and redundancy for performance are implemented through the control plane. Fault tolerance and performance enhancing algorithms as well as data migration and placement according to other policies are similarly not implemented by the buffer service. However, specific operations such as calculating hashes or reconstructing data when storage corruption occurs may be implemented within the buffer service using buffer transformations.

A. *The Data Plane*

As described above, the EBP service defines a means of allocating, transferring and transforming data buffers using resources local to the node or within a LAN. That can be thought of as a passive execution mechanism similar to the datapath of a processor, but extending to transfer among adjacent nodes. We call the infrastructure that implements this service the data plane.

B. *Security in Exposed Buffer Networks*

The need to provide secure wide area service under the Internet model has been a long-standing problem with no obvious solution in sight. The reason for this lies in the choice of end-to-end datagram delivery as the universal service that defines the common ground of the Internet. If we consider routers to control access to all the resources of the Internet that are shared in the wide area, then the means by which (subnets consisting of) routers share those resources is through the exchange of routes (e.g., through BGP). Once a route has been advertised to a peer, a router has little basis on which to make admission decisions, and transit through the router is provided to all. Transit through a router implies that it will allocate whatever resources it has access to from the next router on the path, in effect acting as a full proxy for the sender.

In contrast the local buffers and operations on them comprise the resources of an Exposed Buffer Architecture. Securing the resources of a local node is an easier task because the mechanisms used do not need to scale to the entire community of network users, only to those with direct access to the node. In Exposed Buffer Architecture, access to the resources of non-local nodes is done through services implemented in the control plane, through service-specific peering mechanisms and agreements that can be specialized to the community served. Thus, global routing need not be supported by any service on a particular node, if it chooses not to participate in the kind of unmediated communication service that defines the Internet.

C. *EBP in Overlay: The Internet Backplane Protocol*

Exposed Buffer Architecture is a general approach to the design of systems, and applies to many implementation strategies at many scales. For example, one interpretation of RISC architecture is that in decomposing multicycle operations into microoperations connected by explicit register operations, it is an application to datapaths within a processor of the same approach suggested by Exposed Buffer Architecture. On a different scale, the use of replicated servers in Content Delivery Networks accompanied by topology-aware DNS resolution is also an application of Exposed Buffer Architecture, albeit one that is then hidden from end users through spoofing of the URL display in browsers (which is a continuing basis for social engineering attacks).

Exposed Buffer Processing is the name for an (as yet undefined) implementation of the architecture within 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) [5]. As we discuss in Section V-B, there is an evolutionary path from IBP to more native forms of EBP that access lower layers of the Internet stack or directly on the local layer (a so-called ”clean-slate”).

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:

- **Allocating** a buffer on a specific node, specifying the size and duration exclicitly. 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 directly.
- **Transfer** of data between two buffers on one data plane node or between two by specifying the keys of both sender and receiver. These data plane nodes must be adjacent (reachable through Internet routing).
- **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 do not necessarily 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 QoS is implemented by control plane services.

IV. Structural Metadata and the Control Plane

There are applications that can make direct use of the resources of the data plane to allocation, move and transform data. However, those functions are quite primitive and local, and in general must be aggregated to implement abstractions that are more easily usable by application developers. IBP clients can also implement control plane services which they then offer to applications or to other services.

The analogy in Networking is the aggregation of IP best effort datagram delivery into connected TCP streams. In computation, it is the aggregation of primitive operations or VM time slices into orchestrated collections of threads. In storage, it is the aggregation of disk blocks or storage objects into file systems or databases.

The primitive resources of the data plane are named and controlled using capabilities, which act as pointers that are either local to a node or in the case of transfer, dereferenced across a LAN adjacency. These capabilities can be organized into data structures, along with additional control-oriented information for the purpose of implementing higher level abstractions and functions. We call these data structures *structural metadata*.

In Networking structural metadata is found in packet headers and connection state. In computation, it is the process descriptor, memory management and other conrol registers. In storage, it is the inode, B-tree or other storage data structure used to organize and access data blocks. (Figure 2)

A. The exNode

The most widely used data structure created to organize structural metadata in the form of IBP capabilities is the exNode [6]. Modeled after the function of the Unix file system's inode data structure, the exNode aggregates multiple IBP allocations to implement a large contiguous data extent addressed linearly. It also aggregates redundant storage allocations to implement RAID-like fault tolerance either locally or using storage distributed across the wide area network.

B. Logistical Networking

The role of structural metadata is just as important in Networking as in Storage. In the conventional Internet stack, such metadata is distributed in packet headers as well as maintained in TCP stream state at endpoints. In Logistical Networking, we can use the exNode as a representation of stored data and more transient data structures within the control plane to represent and control data as it moves through the network. Because of the separation between the control and data planes in EBP architecture, structural metadata may remain physically centralized even as it represents data that is distributed and mobile within the data plane.

C. Converged Computation

The structural metadata associated with a computational process is typically held by an operating system in the form of memory management data structures (page tables), task descriptors, hardware contexts (the contents of registers) and other miscellaneous data structures. Typically, these are expressed in terms of physical addresses and other highly local information that ties each task to one node or to a shared-memory environment.

When data being computed on takes the form of file-like extents, the exNode can take on the role of a page table (much as in memory-mapped I/O). However as with networking, data sometimes has to be managed or moved during computation in ways that may be better represented by other data structures. In cases where computation is applied to stored data over long periods of time, it may be useful to generalize the exNode to represent computationally ortiented data management. Examples are the localized expansion of data (e.g. decompressed, unpacked or transposed) for faster access or transformation).

D. Distributing the Control Plane

Separating structural metadata from ”payload” data has a number of advantages. The metadata is typically much smaller (less than 1% of the data data size, depending on representation) and so can be managed flexibly and efficiently. To the extent that EBP is implemented using a local (or in the case of IBP, wide area) networking environment, a centralized control plane process can manage data stored in a large number of individual data plane nodes. This can expedite implementation of the control plane, although the resulting systems are limited to environments in which the data plane is sufficiently well connected to the more centralized control plane to enable such separation.

A further stage in the implementation of the control plane is to enable the distribution of control functions. While this complicates the implementation of the control plane, it also generalizes its functionality and enables deployment in a wider variety of network environments. A distributed control plane is more scalable, facilitating services in which control functions use more substantial resources. Even more important, in environments where communication between local area domains (subnets) needs to be restricted for security or policy reasons, peering between those domains can occur through coordination between control and data plane nodes at the borders. Thus control plane functions can be implemented in a more centralized or more distributed manner according to specific performance and policy requirements.

A further evolution of the implementation of the control plane is using data plane resources to store metadata and to perform transformations. An example would be storing exNodes in data plane storage allocations, represented by a higher level exNode. That mode of storage could be supported by data plane operations which search through the exNode data structure and extract metadata. The categories of control plane functions that might then be supported using data plane resources could expand to enable the control plane to remain less resource-intensive and thus amenable to a more compact implementation.

V. Shared Platform Architecture

If networking, processing and storage are all higher level services whose current implementation are comprised of operations on persistent buffers, then what is the motivation for expressing them all in terms of a common primitive buffer service? Our primary motivation is to define an architecture for a shared infrastructure for the implementation of services that span those traditional ICT silos that will be as widely used as possible for as long as possible.

A. The Deployment Scalability Tradeoff

In a stack of services higher layers are implemented in terms of lower layer ones. A ”spanning layer” is a layer that virtualizes those below it. Layers above it must be expressed in terms of the spanning layer. The prototypical example of spanning layers are the Internet Protocol Suite at the Network layer of the Internet stack that virtualizes local area services. Another example of a spanning layer is the Unix/POSIX operating system call interface which virtualizes the architectural resources of the host computer.

Any spanning layer creates a community of interoperability in applications that are expressed at layers above it on the stack. *Deployment scalability* is a term which denotes the tendency of a spanning layer to be widely and willingly adopted in ways that span boundaries (what one might call ”viral growth”) [7]. Another way of characterizing deployment scalability is that it means that adoption of the spanning layer offers benefits that outweigh those of using a specialized interface.

The Deployment Scalability Tradeoff 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 limited in its allocation of resources. This motivates EBP being a best effort service which has limits on all atomic resource allocations and requiring aggregation of those resources to meet specific reliability and performance goals. The Deployment Scalability Tradeoff is a principle which relates the design of a spanning layer to its deployment scalability.

B. *EBP Implementation Strategies*

Deployment of network architecture innovations is complicated by the dominance of the Internet Protocol Suite at the Network and Transport layers and the need for stability in both commodity and research networks that are critical important day-to-day operations.

Overlay solutions that are compatible with the architectural rules of the current network inhibit effective use of information and mechanisms that are restricted to the lower layers (e.g., routing and security policy). Native solutions can be deployed when access to the lower layers is possible, for example when the innovative service is being deployed by the owner of the physical infrastructure (e.g., Content Delivery and Cloud networks). Extensions of commodity networking environments which remain compatible with end user interfaces while allowing greater configurability or programmability of network nodes are possible (e.g., NFV). These approaches have not yet yielded infrastructure that exhibit a high degree of deployment scalability.

We propose an approach to deploying EBP that proceeds using a combination of overlay and native mechanisms. An EBP protocol can be encapsulated within IP packets, enbling it to share physical infrastructure with commodity Internet traffic, but virtual LANs or SDN can be used to keep EBP traffic from being routed between networks. In essence, IP is thus used as a convenient local area networking solution. To connect local EBP domains Internet connectivity between peering control plane nodes can be used. This approach enables dedicated control plane nodes to be deployed, but the EBP control plane can also be implemented within the container execution capability of NFV-enabled routers.

VI. 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-tenent infrastructures: PlanetLab, GENI Racks, and most recently NFV. At the same time, the demand for utility storage and computing services have given rise to a sequence of solutions based on distributed data centers from early online services (e.g., Compuserve and AOL) to the computational Grid and most recently the National Research Platform. Edge and Fog services serving 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-connected 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 isolated silos.

VII. Conclusion

In this paper we have presented Exposed Buffer Architecture, which addresses the primary goal of Software Defined Networking and Network Function Virtualization: augmenting and extending the flexibility of network services implemented at the Network layer. EBA proposes to move the virtualization layer that provides interoperability/portability in the implementation of applications to the local layer which is generalized by including storage and processing resources. This approach enables backward compatibility with current Internet Protocol clients in Network layer services, but does not require it. It defines a converged platform for implementing new services as alternatives to the Internet.

References

[1] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow world congress*, vol. 48. sn, 2012, pp. 1–16.

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

[3] J. McCauley, Y. Harchol, A. Panda, B. Raghavan, and S. Shenker, "Enabling a permanent revolution in internet architecture," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 1–14.

[4] 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.

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

[6] A. Bassi, M. Beck, and T. Moore, "Mobile Management of Network Files," in *Third Annual International Workshop on Active Middleware Services*, Aug. 2001, pp. 106 – 114.

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