Locating Interoperability in the Network Stack

Micah Beck Terry Moore
Logistical Computing and Internetworking Laboratory
Computer Science Department, University of Tennessee
{mbeck, tmoore}@cs.utk.edu

ABSTRACT

The conventional view within the Internet community is that IP is the appropriate basis for interoperability in the network stack. However, recent developments in IP networking and in approaches to the link layer have generated increasing pressure towards the creation of non-interoperable network domains. In this paper we explore one avenue of attack on these problems, using the End-to-End Principle as a guide to help locate a point within the network stack where the required degree of interoperability may be found. Our conclusion is that the appropriate location of interoperability is in a buffer management layer located between the link and network layers, which we call the *transit layer*. We argue that a transit layer protocol that abstracts the particular characteristics of different intermediate node resources (in the dimensions of data transfer, storage and processing) while being more general and sitting below the network layer in the stack, could exhibit greater deployment scalability and provide a broader foundation for network interoperability than IP.

1. INTRODUCTION

David Clark's seminal characterization of the Internet's goals and its underlying architecture make it clear that interoperability was the key to its mission [7]. The Internet addresses the challenge of creating an "…effective technique for the multiplexed utilization of existing interconnected networks…" by deploying a *common service* (viz. "… a packet switched communication facility …which implement[s] a store and forward packet forwarding algorithm") through which all the networks that it encompasses can transfer data interoperably. IP datagram service thus provides the interoperability that embodies the Internet's "top level" goal.

Given this goal, the more deployable the common service is, the more valuable it will be. *Deployment scalability* is our term for the relationship between an increase in the "size" of a network or distributed system's deployment, as measured by some set of quantifiable dimensions, and the concomitant changes in the size or degree of some other set of attributes considered important to the system's success, such performance, reliability, and operating cost. Clearly the scalability of the Internet, in this sense, has been remarkable. It has grown dramatically in many of the dimensions along which the deployment size might be measured, including the number of intermediate nodes deployed, the number and size of different networks subsumed, the number of organizational borders crossed, the extent of geographical distances spanned, the number and diversity of services supported. At the same time, despite this incredible growth in every direction, its performance, reliability and many other valued properties have been either preserved or improved over time.

In spite of, and partly because of, this success, the network community has arrived at a crossroads at which a number of competing forces are converging [8]. On the one hand, the Internet is faced with the demand to support a much wider class of applications and services than it was designed for. At the same time, the explosive improvements in fundamental technologies require that these new applications and services be implemented under dramatically different conditions. This has led to a situation in which the ability of IP, or any single network layer service, to provide a basis for universal interoperability is in doubt.

We believe that defining a common service, based on the broadest possible level of commonality, is a prerequisite for enabling a community to create a shared information infrastructure that can effectively meet a broad diversity of its needs and aspirations. If it is realistic to believe that this task can be successfully undertaken once again in the current, more diverse network environment, we feel that it is too important not to try. In this paper we explore one avenue of attack on this architectural problem, using the End-to-End principle as a guide to help locate the point within network stack where the required degree of interoperability can be found.

2. E2E AND SCALABILITY

The End-to-End (E2E) Principle was developed as a methodology for understanding the impact of assigning functionality to the layers of a communication stack in the delivery of data from one network endpoint (the sender) to another (the receiver) [13]. Over the years since these ideas were first introduced and the name was applied to them, they have engendered no small amount of controversy. There are many different formulations of them, for example, some ranging quite far from the networking domain. Some have seen E2E as normative

Univ. of Tennessee Computer Science Dept. Tech. Report UT-CS-04-520, Feb. 2004

principles telling network architects what they may and may not do; others have seen them as a set of arguments to help designers understand networks better. But there is no doubt that E2E arguments have guided the development of the Internet and served to make it the world's most scalable computer networking system.

Like its originators [13], we see the use of the E2E Principle in networking as equivalent to a more general class of layering arguments that arise in the design of computer systems of all kinds. The multiplicity of interpretations of E2E stems from the fact that a specific case is being considered, rather than the more general principle. (Hence the existence of what are sometimes refered to as a class of "E2E arguments"). Our formulation of the general principle, which we continue to call E2E for the sake of familiarity, is as follows:

> **The E2E Principle**: When designing a service that is implemented using a shared infrastructure, there is an inherent tradeoff between the service's *scalability* on that infrastructure, and both i) its specialization to a particular class of applications, and ii) the value or scarcity of the resource consumed to provide it.

In our view, there are three primitive services that network intermediate nodes, as key elements of the shared infrastructure, can make available as resources to enable the creation of network services:

1. transferring data between neighboring nodes
2. storing data, and
3. applying transformational operations to data

From this perspective, these fundamental services are all at an architectural level that is below the network layer stack, since they are required in order to implement end-to-end IP datagram delivery service.

Given our formulation of the E2E Principle, which treats it as a hypothesis about fundamental limits of scalability in the face of service specialization, and cost and/or scarcity, one can see why attempts to make use of the local storage or processing services on the intermediate nodes have sometimes been interpreted as *prima facie* "violations" of E2E. Some have viewed any creation of visible data state within the network this way, for example, because of the belief that storage of value to users. In that case the E2E Principle tells us that the need for reliability would result in a commensurate lack of scalability in the resulting network, since reliability and indefinite allocation can monopolize valuable resources. Similarly, transforming data at intermediate nodes may be viewed as a violation of E2E. The idea is that processing requires security, scheduling, code portability and other expensive and specializing services that, according to the principle, will result in a system with little scalability.

Over the years various attempts have been made to avoid or mitigate the discipline implied by E2E requirements. With the explosive Internet boom of the late 1990s, the idea arose that IP infrastructure would now remain fixed, but overlay infrastructure would grow up on top of it to allow the creation of additional services. This had the advantage of letting the designer of overlay services off the hook regarding the scalability of their service, since they were designing a service only for some community of users, not for the entire network. The communities were usually the sites of a single enterprise or the paying customers of some specific online business. While this has led to some very valuable enterprise, intranet and overlay solutions, it has not added to the ability of the public infrastructure to respond to the needs of public events, or to meet the unexpected needs of user communities who have not arranged for overlay infrastructure to be deployed on their behalf.

Another situation in which one might adopt an approach that E2E predicts is less scalable occurs when one is working in an environment where some of the approaches that would enhance scalability are simply impractical. For example, the developers of Interplanetary Networking (IPN) found that the delays inherent in end-to-end signaling on a planetary scale made end-to-end data integrity checking and retransmission of data impractical, and so they were forced to adopt a rigorous system of hop-by-hop reliability based on database technology which they call "custody transfer" in analogy to post office procedures [5]. In this case, it was seen as necessary to forfeit scalability in order to implement the application at all.

With such experiments at cheating E2E, various attempts to wring profitability from non-scalable services, and cases where architects were forced to turn away from it, those who believed E2E was a fundamental tradeoff were left with no way to expand scalable networking to encompass new classes of resources. While some have believed that this showed that it was time to "go beyond" E2E in order to make further progress, we argue that a more careful application of the E2E to problem of service creation at the intermediate node may yield better results.

3. LOCATING A LAYER FOR A COMMON SERVICE

In his seminal 1988 paper "The Design Philosophy of the DARPA Internet Protocols," D. D. Clark, after identifying the Internet's top level goal of providing a foundation for network interoperability, identified seven "second level" goals for creating an effective Internet architecture [7]. The top three (in order of importance) are as follows:

1. "Internet communication must continue despite loss of networks or gateways."

2. "The Internet must support multiple types of communications service."
3. "The Internet Architecture must accommodate a variety of networks."

There is rough agreement that during the first decade and a half or so of its existence, the original design of the Internet Protocol satisfied all three of these requirements to a remarkable degree. But while all the evidence is that the design of IP, in particular the insistence on weak semantics, continues to be successful in achieving the goal of survivability, the situation with respect to the second and third objectives is a matter of widespread debate. To the extent that the design of IP is still well adapted to the application demands at the transport layer above it (goal 2), and to the technological demands of the network technologies available at the link layer beneath it, (goal 3) we would expect that the community would still choose the benefits of interoperability in spite of pressures to make requirements that cannot be accommodated while preserving IP as the common network layer protocol. However, there is significant evidence that, as the Internet continues to expand, the build up of pressures from both directions at once is having the opposite effect.

In the case of goal 2, there are a variety of communication services required by application communities that are not well supported by IP. As a result, there has been a succession of movements to create non-interoperable alternatives [2, 14, 15], a proliferation of conflicts in various areas over which a number of mutually exclusive design choices should be taken [8], and the creation of application specific overlays.

In the case of goal 3, there are a variety of link layer technologies that violate the assumptions required to implement IP at the network layer: that an immediate end-to-end forwarding path exists between endpoints, that the maximum round trip time is not excessive and that the end-to-end packet drop probability is small [10]; or that the underlying physical transport is packet rather than circuit based, as in some forms of optical networking. Some in the community see this effect as so inescapable that they have proposed simply abandoning the goal of a common service and replacing it with a common system of metadata to describe and control the inherent heterogeneity emerging at network layer [9].

Thus, IP is gradually loosing its effectiveness in the role of the common service, both in terms of its ability to support new types of application services and its ability to integrate new types of networks. The gradual balkanization that this is producing exposes the limits of IP's deployment scalability. Now suppose we wanted a network that could scale beyond these limits, and that we believed such a network was possible. How would we design a new common service in order to achieve it? .

The obvious place to turn for guidance would be the E2E Principle. As we have remarked above, our interpretation of the E2E Principle tells us that a network service that is sufficiently generic and which does not consume resources that are too valuable or scarce will be scalable. The problems emerging in the ability of IP to serve as the unifying protocol shows that either

- It is not generic enough, or
- It is consuming resources that are too scarce or too valuable

We rule out the latter as a source of non-scalability in IP, as well as for other solutions we discuss later which adhere strongly to the use of *weak semantics* and *limited service* to bounds the maximum consumption of resources by any atomic service implemented at the intermediate node. In the case of IP datagram delivery, this takes of form of best effort reliability and limited MTU. Other examples of solutions that adhere to this approach include Logistical Networking and Ephemeral State Management (sec. 5.1).

Thus, turning to increasing the generality of IP, if we look at what services applications want that IP doesn't model, chief among them are storage and processing of data *in transit*, i.e. at intermediate nodes. On the other hand, these services, unlike IP datagram delivery, are inherently *local* to the intermediate node, rather than being defined across a homogeneous network layer. Thus, they are more analogous to the link layer, which connects *adjacent* nodes. Thus, we propose to generalize the view of layer 2 to include local storage and processing services. We call this more general layer, which includes link, storage and processing as coequal elements, the *local layer* (Figure 1).

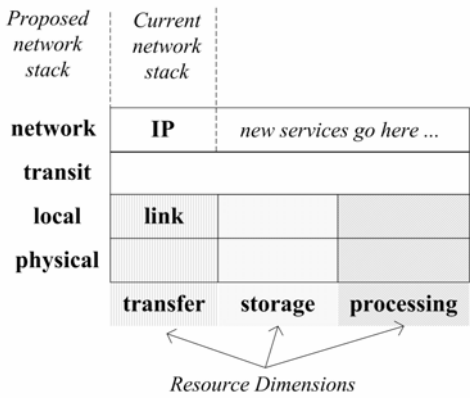

Figure 1: Location of the transit layer in the network stack

The local layer exhibits the same extreme heterogeneity as the link layer in each of its three elements or dimensions. Storage "paths" can be as different as fast access RAM and ultradense physical media. The processing "paths" are complex operations implemented on devices as dissimilar

as microprocessors or FPGAs. The key point to notice, however, is that all of the operations of the local layer can be modeled as providing services of various kinds *to arrays of bytes of data stored in transit* at the intermediate node.

For this reason, a network protocol capable of expressing a broad class of operations on *byte arrays* located at the intermediate node *can abstract a more general class of lower layer services than IP does*. By choosing to model operations that are either local to the intermediate node or restricted to operating on nodes connected by adjacent links, the protocol can avoid implementing routing or wide area algorithms of any kind. Because network layer protocols, such as IP, can in fact be implemented on top of it, we would situate the new protocol between the local and network layers in the current network stack. Since a protocol at this layer would provide an abstraction of services for data that is *in transit* at the intermediate node, we propose to call it the *transit layer* (see Figure 1).

Our position is that a transit layer protocol that abstracts the particular characteristics of different technologies at the local layer, while being more general and sitting below the network layer in the stack, would exhibit greater deployment scalability and provide a broader foundation for network interoperability than IP.

3.1 A Note on Network Transparency

One of the important ideas in the architecture of the Internet is the notion of network transparency. If we considered, as the original designers of the Internet did, that the function of a communication network is to pass data unchanged between endpoints, then would be natural to compare the network to a transparent window, through which an image passes unchanged. Of course this is far from the truth of the Internet, or any other network that is switched above the physical layer. The network is full of complex mechanisms that implement the high level function of transporting data unchanged. In the terms of information hiding, the transparent network is an opaque mechanism, a somewhat unfortunate discord in imagery.

While this might seem to be merely a rhetorical problem, when we consider a generalized network that can transform data as well as transport it unchanged, the rhetoric of transparency as an unconditional virtue can get in the way. A network that completely encapsulates the richness of its resources at the local layer cannot use those resources to build new functionality at the network level. If we focus instead on the idea that the network should appropriately abstract the resources of the local layer in their *diversity*, while still exposing the buffer management mechanisms of the transit layer in order to allow the construction of new network layer services, we have a model that can accommodate both abstraction and adaptability.

Thus, on this analysis, network transparency is one solution to the more general problem of abstracting away from the link layer, but it oversimplifies by leaving out the storage and processing components of the local layer. The problem is then to define a layer which exposes those components in a manner that is sufficiently general and yet yields to practical implementation.

4. DEFINING THE TRANSIT LAYER

In describing transit layer services, we need a basic unit of data on which operations can be performed. At the link layer the fundamental unit is the packet and the fundamental service is the delivery of packets between adjacent nodes. At the IP network layer, the unit is the datagram, and the fundamental service is the delivery of datagrams between network endpoints.

We can model a packet or datagram formally as a 3-tuple <t, a, P> consisting of :

- a *payload* (array of byte values) $P = v_0, v_1, \ldots$
- a *network location* specified by a link or network address a, respectively
- at a *global time* t

The basic operation defined on a datagram (packet) <t, a, P> by the link (network) layer is transfer:

> *Transfer*. Data is copied from one network address to another, yielding a new buffer <t+ε, a', P> where ε is a function of the static and dynamic network topology.

Because the transit layer models services provided at the intermediate nodes, we extend our model of the basic unit of data to a 4-tuple <t, a, b, P>, which we call a *byte array*, and which includes a *local buffer name* b within the node. The buffer is a physical storage resource, which can hold a payload value unchanged with some degree of inherent reliability over time. In our model of the transit layer, buffers are also the starting and ending points for operations on those payloads.

We categorize operations on these buffers three fundamental buffer operations types:

1. *Transfer*. Data is copied from one network address to another, yielding a new byte array <t+ε, a', b', P>, where ε is a function of the static and dynamic network topology.
2. *Storage*. Data is stored until a later time, t', yielding a new byte array <t', a, b, P>
3. *Processing*. One (or more) byte array(s) on a single node serve as inputs and/or outputs to an operation, yielding one (or more) byte array(s) with a new value <t+ε, a, b, P'> where ε is a function of the operation and the data.

We apologize to the reader for this highly abstract and unsatisfying characterization of transit layer operations as

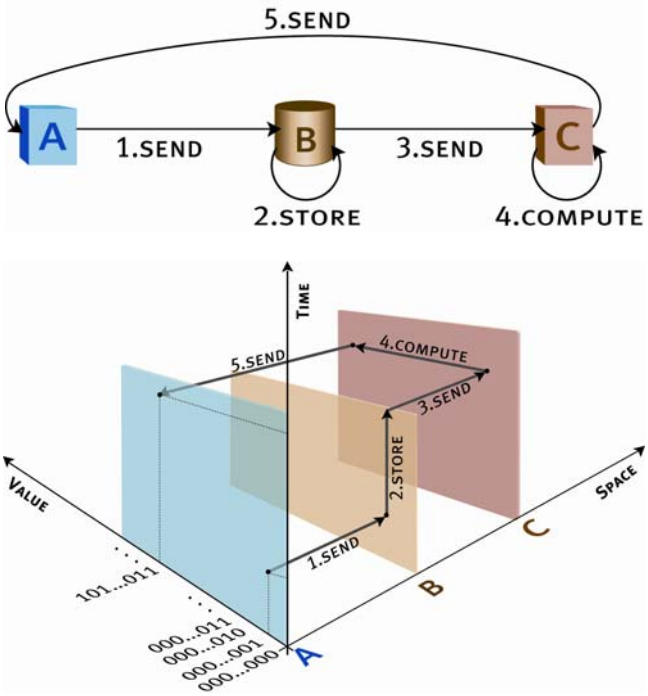

Figure 2: Transit layer connectivity in the transfer, storage, and processing dimensions.

arbitrary transformations of byte array attributes, without specific examples. The possible interpretations cover a wide range. At one extreme byte array could be a small vector, a transfer a wireless packet hop and processing could be a single ALU operation per word; or, at the other extreme, a byte array could be a gigabyte sized region of disk, a transfer could be an optically switched transfer at Tbps rates, and processing could be a visualization kernel massively parallelized on a shared memory multiprocessor. While the framework seeks to reach toward such extremes, the reader can think of examples of moderate operations as primitive but time-sliced virtual machine instructions for implementing active routing, distributed computation, and data centric applications, such as those described in the Active Networking and Grid literature [11, 12].

The interpretation of transfer, storage and processing as operations that act on byte arrays in the temporal, spatial and value dimensions is illustrated in Figure 2. While there is no logical reason why there should not be other categories of operation which, for instance, combine storage and transformation, current technology primarily supports these three basic types. Other important categories of operation support services such as synchronization, and again those can be combined with the three fundamental categories listed here. These other operations have been omitted from the current discussion for the sake of clarity and brevity.

We consider every byte array to have a neighborhood in time, space and value, consisting of other buffers to which it can be transformed in a single transit layer operation. It is important to note that the transit layer neighborhood is not assumed to be fixed over time or even stable for long periods. As with link layer instability in the Internet, extreme instability at the transit layer may interfere with the correct functioning of some network layer routing protocols. However, the intent is to model a dynamic environment in which neighborhoods change over time, and where end-to-end paths through these neighborhoods may exist only by taking advantage of spatial, temporal and transformational dimensions.

If the spatial dimension of the transit layer neighborhood is defined by link layer adjacency, then the temporal dimension is defined by the storage technology that implements the buffer itself. A RAM buffer is not likely to maintain a value for as long as a buffer implemented on disk. Storage provisioning is thus analogous to temporal connectivity.

Finally, the transformational dimension of the transit layer neighborhood is defined by the processing capabilities of the intermediate node on which the buffer is stored, as well perhaps as the connectivity between the storage resource in which it resides and that processing resource. For instance, a RAM buffer may have access to fine grained operations implemented by a processor attached to the memory, while a large buffer held on tape may not have any processing connectivity at all.

Processing connectivity is likely to be the most irregular dimension because of the complexity of high performance computing devices and the irregularity of the operations they implement compared to data storage and transfer. Settling on a universal set of operations, like the problem of agreeing on a single processor or virtual machine model, set of operating system primitives, or any other API, may be difficult or impossible, depending on the area of application.

It is our position that there should be a subset of functionality sufficient to implement a range of network layer services that can be decided through community decision-making processes such as the IETF. Experiences with related community defined programming interfaces such as Linux and Java have provided a lot of experience with the creation of common computing interfaces, and the transit layer is in many ways simpler because processing is so orthogonal to network and transfer functions. High level APIs will be programmed on top of the transit layer, not built inside it.

However, given the fact that some application communities will inevitably require features that cannot be standardized, we anticipate the emergence of service discovery and negotiation protocols at the network layer to enable the use of transit layer nodes with an interoperable core that nonetheless do not implement identical sets of operations. Because of lesser but still significant diversity in storage services and increasing diversity in transfer

services, the same may also be true for those dimensions of the local layer.

Another important feature of any layer of the network is that it defines a mechanism for the sharing of resources between the participants. In the case of the transit layer, the participants in the protocol are either on local host or are adjacent at the link layer, and so can the provisioning of resources can be quite finely modulated to the nature of the link layer connectivity. When the link layer defines a private or highly trusting community, the transit layer can provide free access to significant levels of resources in all dimensions.

This is especially important when we generalize the local layer to storage and processing resources because, although it is possible to create wide area communities that share such resources freely, it is more common to find a need to impose controls such as user authentication, quotas and billing, which can put significant barriers in the way of communication and collaboration. Locating the sharing of such resources at the transit layer allows the controls necessary for wider sharing to be located at the network layer and specialized to the nature of network layer services.

5. RELATED APPROACHES TO ENRICHED RESOURCE NETWORKING

The design approach that we propose in this paper spans the application of modern computer system architecture and engineering principles to data transfer, storage and processing resources on scales ranging from micro to macro, but we have skirted some important questions, such as how to deal with synchronization. The conceptual tools that will be required to make this approach work are taken not only from networking, but also from among the most influential systems of the past several decades in areas including operating systems, virtual machines, pipelined and parallel processor scheduling, fault tolerant and high performance storage and computing. In lieu of a comprehensive section covering these topics, we present some recent projects, which have taken similar or alternative design paths. The development of a detailed proposal for a transit layer protocol is also beyond the scope of this paper, and we do not present any of the following as specific proposals for a transit layer protocol.

We discuss two categories of approaches to the creation of services at intermediate nodes: those which *expose* all data state created at the intermediate node to direct inspection and manipulation by higher layers, and those which create state which is *encapsulated*, i.e. it is accessible only to the service implementation. In addition, we briefly discuss peer-to-peer approaches that directly create new network services at the end-points.

5.1 Exposed Approaches

5.1.1 Logistical Networking

The research team led by Beck and Plank at the University of Tennessee's Logistical Computing and Internetworking (LoCI) Laboratory has for more than five years been developing the Internet Backplane Protocol (IBP), an overlay implementation of functionality closely related to that required of the transit layer [3, 4].

The fundamental categories of operations implemented by the Internet Backplane Protocol are:

- allocate a persistent storage buffer on an intermediate node (either the local node or one that is adjacent at the link layer); storing data to, loading data from, and managing the control state of such buffers
- transfer data between buffers, and
- process data stored in some set of buffers under the control of a specific predefined operation

The definitions of the transit layer operations in Section 5 are taken from the semantics of the core categories of IBP functionality, but stripped of much of the complexity required by the specifics of storage resources management and the specific difficulties of an overlay implementation (as discussed below).

Of these three categories of IBP functionality, a significant amount of the complexity of IBP is focused on the first category. It derives from the creation of persistent state visible to the client and the need to manage it in a manner that protects clients from one another (using a large, randomized namespace of capabilities) and that allows the intermediate node not to overtax its storage resources (best effort leases, maximum allocation size). Data transfer and processing are completely stateless, manifesting their effects solely by modifying the values stored in their storage buffer arguments.

The protection of the depot's data transfer and processing resources is enabled by the enforcement of MTUs for transfer, and maximum units of computational work for processing. It is worth noting that to the extent that the access of local layer services by network services is more trusted than the access of host resources by IBP clients, the problem of enforcing resource sharing, which has been a difficult one in Logistical Networking, may be more tractable in a transit-layer implementation,

The IBP intermediate node, or *depot*, is implemented at the application level. It is accessed using a TCP-based client/server protocol that leverages a variety of data transport mechanisms between nodes. The IBP depot is the core infrastructure for the LoCI project. As would be expected, additional E2E services (e.g. aggregation, fault tolerance, compression, replication, and optimization of data transfer performance) are implemented in libraries that execute at network end points.

Because the control of processing resources within the IBP depot is still immature, more autonomous services (e.g. resource discovery, routing, and active data management including error recovery) are currently implemented as self-contained processes that run on end systems. However, any transit layer protocol must have, as a primary design requirement, the ability to overcome the latencies imposed when local operations are initiated by the network layer. The design of IBP allows for such optimization of the client/server interface without a loss of scalability.

5.1.2 Ephemeral State Processing

Calvert, Griffioen and Wen [6] have developed Ephemeral State Processing as a mechanism to maintain persistent state at IP routers and perform operations on it. Their approaches to accounting for the E2E Principle are similar to those taken in the design of IBP: storage allocations are limited in size and duration, instructions are restricted to a limited set installed on the router, and both functions are best effort. However the scale of their ephemeral state is orders of magnitude smaller than the storage supported by IBP: storage allocations are limited to single 64 bit words stored for 10 seconds; primitive operations analogous to individual machine instructions act on one or two stored words. While this greatly reduces the problem of scalability, it also restricts the applicability of their approach to very simple services

5.2 Encapsulated Approaches

5.2.1 Active Networking

Active Networking has focused on the creation of services at the intermediate node by the specification of a node operating system, which is a service layer that is encapsulated within the intermediate node itself. With such a node OS in place, code implementing a new service at the network layer can then be injected into the active intermediate node and executed there, creating and accessing local state. The key difference between the Active Networking approach and what we describe as the transit layer is that the latter exposes the entire stored data state of the intermediate node for management by higher layers of the stack, and *may not choose to allow the indefinite allocation of a storage or processing resources by the higher layers*.

The transit layer approach, which combines exposed state and weak semantics, can place a much greater burden on the implementation of the network service to manage the state of multiple intermediate nodes in providing service to endpoints. While some of these elements are present in certain Active Networking projects, maximizing their impact as design criteria is central to the E2E philosophy that motivates and informs the design of the transit layer.

5.2.2 Remote Procedure Call and Job Execution

Various Distributed Computing systems implement remote procedure call mechanisms or job execution, and the processing component of the transit layer bears some resemblance to those mechanisms. Remote procedure call usually bundles the movement of data from a client to a computational server, the invocation of a (usually heavyweight) service, and the return of results. Job execution may allow inputs to be prestaged and results to reside at the server. The bundling of data movement with processing, the transfer of arbitrarily complex programs to the execution platform and the encapsulation of state management within an extended execution all create highly complex system behaviors that sometimes have little survivability. The important point to make is that execution of a call (job) *to completion* is a form of *indefinite* allocation of storage and *potentially unbounded* allocation processing resources for the purposes of state management, unless the possible nature of the call (job) is appropriately circumscribed in its use of such resources. The use of specialized state management systems such as checkpointing can enhance survivability at the expense of generality. The experience of the Distributed Computing community with the deployment of these systems does not recommend this approach as a model for massively scalable infrastructure on the level of the Internet. Of course, it was never intended to be so.

5.2.3 Network Attached Storage

The area of Network Attached Storage (NAS) started from an approach focused around server appliances based on the Network File System, but has evolved toward storage devices that are more specially adapted for direct attachment of disk resources to the network. Although protocols such as iSCSI and FCIP have allowed abstraction away from some physical device characteristics, the lack of adequate protection between users makes it more appropriate to think of them as technologies at the local layer (perhaps combining transfer and storage components) rather than at the transit layer, in spite their overlay implementation on top of TCP. However, more storage protocols with more sophisticated allocation models, such as T10 [1], and hybrid storage/processing technologies, such as intelligent disks, continue to move NAS in the direction of transit layer functionality.

5.3 Peer-to-Peer

In the past several years, a number of peer-to-peer services have developed that implement application services using the storage, processing and data transfer resources of a community of network endpoints. Because of the lack of control that the members of such communities exercise over the other members, participants in the protocols must of necessity make very weak assumptions about the behavior of their "peers." They must allow for the fact that peers may be broken, may perform

poorly or inconsistently, may lie about their identity, or may act in a malicious manner, perhaps disabling or stealing the resources of other members of the larger network community.

The necessarily weak assumptions of peer-to-peer protocols are an approximate model to the transit layer of a scalable network. Protocols devised to implement complex processing in peer-to-peer networks may therefore have applicability in the transit layer. But the transit layer should not be as dangerous a place as the peer-to-peer environment, since access to it by end users can be controlled through a combination of link layer connectivity and controls implemented at the network layer. Thus, even algorithms that perform well only in relatively restricted peer-to-peer environments may find applicability in more public transit layer networking domains.

6. CONCLUSION

In this paper, we have revisited interoperability as the basis for the creation and maintenance of shared information technology infrastructure. We have briefly examined the current problems facing the deployment of new application services and the incorporation of new network technologies in an environment where interoperability is based on the use of IP as the common protocol. Using the End-to-End Principle as our guide, we have reached the conclusion that a more general protocol, modeling storage and processing resources on the intermediate nodes, as well as a greater variety of link layer transports, could support a greater diversity of services at the network layer. Thus, interoperability based on the adoption of a common protocol at a transit layer, created between the link and network layers of the current stack, could be expected to include communities that are now attracted to network layers that are incompatible with the IP standard. It could also be expect to enable the utilization of link layer technologies that are not well suited to the transport of IP datagrams.

However, our architectural argument may seem to have skirted some important issues, chief among them the problem of performance. E2E arguments usually allow for cases when application performance requirements have to be explicitly taken into account in making architectural choices. Even scalability may have to be sacrificed when performance requirements are paramount. While our proposed transit layer model provides an abstraction of the local resources of the intermediate node that can integrate diverse low level technologies and enable the creation of diverse network layer services, there is a very real question as to whether any such model can hope to perform at the incredible levels achieved by today's core Internet routers. In principle, routers could be built on virtual machines that transfer data in large segments as well as succinct datagrams. But when high performance video streams at aggregate bandwidths of terabits per second need to reach their destinations with minimal jitter, predictability seems to demand some degree of specialization in the forwarding of datagrams at the network layer.

Does this mean that our analysis is naïve when viewed from a realistic economic and engineering perspective? The answer may depend on the ultimate economic and societal importance of new applications and activities that need the flexibility that the transit layer would enable, compared to the current importance of applications that are very sensitive to latency incurred at the forwarding node. It may also depend on the ability of network engineers, faced with the challenges of building intermediate nodes of such great generality, to optimize and pipeline the regular cases that neither need complex services, nor can afford to take unnecessary detours through so many levels of the stack.

Ultimately, there may be important communication domains where the transit and network layers are collapsed for reasons of performance. Indeed, today's Internet can be viewed as one such region. However, in future such regions may exist as isolated domains within a more generalized network that is able to serve an extremely diverse community of interests. While engineering advances may deliver performance levels once thought to be unattainable for a scalable network, some desirable applications may remain out of reach as part of the price to be paid for more universal interoperability. It's a familiar story, and we believe the time may have come to play it out again on an exciting new stage.

7. Acknowledgments

Through his leading role in the conception and development of the Internet Backplane Protocol and the entire Logistical Networking storage middleware stack, Jim Plank has contributed fundamentally to the ideas presented in this paper. The success of Logistical Networking as an applied research project in computer systems is due largely to Jim's, talents, knowledge and experience. As co-director of the Logistical Computing and Internetworking Laboratory, he provided indispensable leadership and vision in driving our research collaboration.

Early contributions were also made to this work by Martin Swany and Rich Wolski, who participated in seminal architectural discussions and the original design of IBP. The work of the IBP implementation team – Alex Bassi, Wael Elwasif, Erika Fuentes, Sharmila Kancherla, Jeremy Millar, and Yong Zheng – also contributed to the architecture as it matured.

Other members of the Logistical Computing and Internetworking Laboratory, including students and research staff, have worked with IBP, developed other elements of the Logistical Networking middleware stack, and provided important insights and critical feedback that helped shape these ideas. We particularly would like to

acknowledge the contributions of Scott Atchley, Ying Ding, Dusty Parr and Stephen Soltesz.

Finally, the authors gratefully acknowledge the early support and ongoing collaboration with Jack Dongarra and his team at the Innovative Computing Laboratory.

8. REFERENCES

[1] *www.T10.org*.

[2] D. Alexander, W. Arbaugh, M. Hicks, P. Kakkar, A. Keromytis, J. Moore, C. Gunder, S. Nettles, and J. Smith, "The SwitchWare active network architecture," *IEEE Network*, May/June, 1998.

[3] M. Beck, T. Moore, and J. S. Plank, "An End-to-end Approach to Globally Scalable Network Storage," in *Proceedings of ACM Sigcomm 2002*. Pittsburgh, PA: Association for Computing Machinery, 2002, pp. 339-346.

[4] M. Beck, T. Moore, and J. S. Plank, "An End-to-End Approach to Globally Scalable Programmable Networking," in Future Directions in Network Architecture (FDNA-03), an ACM SIGCOMM 2003 Workshop, Karlsruhe, Germany, August 25-29, 2003.

[5] S. Burleigh, A. Hooke, L. Torgenson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, "Delay-Tolerant Networking -- An Approach to Interplanetary Internet," *IEEE Communications Magazine* June, 2003.

[6] K. L. Calvert, J. Griffioen, and S. Wen, "Lightweight Network Support for Scalable End-to-End Services," in *Proceedings of ACM Sigcomm 2002*. Pittsburgh, PA: Association for Computing Machinery, 2002.

[7] D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *ACM CCR '88*, vol. 18, no. 4, pp. 106-114, August, 1988.

[8] D. D. Clark, J. Wroclawski, K. Sollins, and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet," in *Proceedings of ACM Sigcomm 2002*. Pittsburgh, PA: Association for Computing Machinery, 2002.

[9] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, "Plutarch: An Argument for Network Pluralism," in Workshop on Future Directions in Network Architecture (FDNA-03), Karlsruhe, Germany, August, 2003.

[10] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," Intel Research, Berkeley, IRB-TR-03-003, February, 2003.

[11] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufman Publishers, 1999, pp. 677.

[12] L. Lehman, S. Garland, and D. Tennenhouse, "Active Reliable Multicast," in *Proceedings of IEEE Infocom,*, 1998, pp. 581-589.

[13] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277-288, November, 1984.

[14] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wethrall, and G. J. Minden, "A Survey of Active Network Research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80-86, January 1997, 1997.

[15] D. Wetherall, J. Guttag, and D. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," in IEEE OPENARCH, San Francisco, CA, April, 1998.