Revitalizing the Public Internet By Making it Extensible

Hari Balakrishnan[1], Sujata Banerjee[2], Israel Cidon[2], David Culler[3], Deborah Estrin[4], Ethan Katz-Bassett[5], Arvind Krishnamurthy[6], James McCauley[7], Nick McKeown[8], Aurojit Panda[9], Sylvia Ratnasamy[10], Jennifer Rexford[11], Michael Schapira[12], Scott Shenker[10,13], Ion Stoica[10], David Tennenhouse[2], Amin Vahdat[3], Ellen Zegura[14]*

[1] MIT, [2] VMware, [3] Google, [4] Cornell Tech, [5] Columbia, [6] UWashington, [7] Mount Holyoke, [8] Stanford, [9] NYU, [10] UC Berkeley, [11] Princeton, [12] HUJI, [13] ICSI, [14] Georgia Tech

Contact:shenker@icsi.berkeley.edu

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
The authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

Abstract

There is now a significant and growing functional gap between the public Internet, whose basic architecture has remained unchanged for several decades, and a new generation of more sophisticated private networks. To address this increasing divergence of functionality and overcome the Internet's architectural stagnation, we argue for the creation of an Extensible Internet *(EI) that supports in-network services that go beyond best-effort packet delivery. To gain experience with this approach, we hope to soon deploy both an experimental version (for researchers) and a prototype version (for early adopters) of EI. In the longer term, making the Internet extensible will require a community to initiate and oversee the effort; this paper is the first step in creating such a community.*

CCS Concepts

• **Networks** → **Network architectures**;

1 Why try to change the Internet now?

It is well known that the current Internet's design has many shortcomings. For instance, it was not designed with security in mind, so it is vulnerable to a wide range of attacks such as distributed denial of service, route spoofing, and DNS hijacking. In addition, the Internet's service model of host-to-host packet delivery is not ideally suited to today's usage, which is predominantly content- and service-oriented and typically involves mobile and/or multi-homed clients. We cite these problems not as a criticism of the original architecture – whose generality and longevity have been astounding – but of our inability to evolve that architecture as its inevitable inadequacies have become apparent through changes in usage and expectations. Despite extensive research into alternative designs that would address the problems cited above, this architectural stagnation has persisted for several decades.

Before continuing, we clarify what we mean by architecture. Broadly construed, the Internet architecture encompasses the Internet's layered design and all of its constituent protocols. Here we adopt a narrower definition that only includes the end-to-end service model of the Internet (best-effort packet-delivery), but not the details of the protocols used to achieve it. With this definition, neither the transition from IPv4 to IPv6 nor the addition of new transport protocols are seen as architectural changes. In addition, we will use the term service, which has many meanings in the networking context, to mean the end-to-end functionality provided to hosts by the network portion of the Internet infrastructure.

While the architecture has remained unchanged, there have been significant technical developments that have enabled operators to build better networks[1] and improve application performance and security/privacy through better infrastructure management (*e.g.,* software-defined networks, network virtualization, network function virtualization), more flexible hardware (*e.g.,* programmable routers, SmartNICs), more secure and private protocols (*e.g.,* TLS, DNSSEC, RPKI, QUIC, oDNS), and more performant congestion control algorithms (*e.g.,* BBR [3], Annulus [20]). These innovations enable the Internet to do a better job of supporting the Internet's current architecture; however, they do nothing to transform that architecture.

This architectural stagnation has led many to conclude that significant change to the Internet's architecture is not possible. Some say that we tolerate the shortcomings of the Internet because it works sufficiently well, so there is no need for change. On the contrary, we contend that there is a need for change, but we continue to use the current Internet because we have no other choice: despite significant efforts, we have not been able to change the Internet's architecture, and there is no readily available substitute.

However, the first glimmer of an alternative has recently appeared in the form of an emerging generation of large private networks. Over the past decade, several cloud and content providers have constructed extensive high-capacity networks to handle their traffic. To extend the reach of these networks, they have deployed points-of-presence (PoPs) within tens of milliseconds of much of the world's population, and have also deployed an increasing number of off-network caches. As a result of these developments, a significant fraction of the Internet's traffic is either handled by internal caches or exchanged directly between a client's domain and one of these large private networks via their PoPs [2, 15].

What makes these new private networks functionally different from traditional carrier networks is that they typically apply additional processing in their PoPs to improve application performance. This in-network processing is fairly basic – consisting

*Authors in alphabetical order.

[1] Unless otherwise specified, we use the term "network" to refer to enterprise, campus, domain, or other networks within a single administrative realm.

mostly of caching (which has long been provided by CDNs), flow termination (which improves end-to-end delivery times), and load balancing (which improves application performance by avoiding overloaded servers). Thus, the network-level service provided by these new networks is not purely host-to-host IP packet delivery (though, to maintain backwards compatibility, it appears that way to clients).

While these private networks enhance customer experience, they also create a growing functional disparity between what we call the *public Internet*, whose service is limited to end-to-end IP-delivery, and these emerging private networks whose additional processing – for their own applications and those hosted on their clouds – provides substantially better performance and security.[2] As a result, the original vision of a unified public Internet is starting to fracture into a collection of private networks, each with its own proprietary service offerings.

While this is a worrisome development, it is also an opportunity in that these private networks can help improve the public Internet in two ways. First, they show how the Internet's basic service model can be augmented with additional services without changes in IP or other core protocols. While the ability to create an Internet with an extensible service model has been described conceptually in several research papers [9, 11, 14, 16, 18, 19], the existence of these private networks and their enhanced functionality makes a far more persuasive case.

Second, these private networks – with their extensive reach, high capacity backbones, and PoPs capable of in-network processing – provide an ideal platform on which to deploy a new and more flexible Internet architecture. We propose leveraging this platform to transform, in a fully backwards-compatible manner, the current Internet into one with an extensible service model where the adoption of new functionality does not require equipment upgrades nor substantial delays for standardization and deployment. In short, we believe there is an unprecedented opportunity to gradually but radically remake the Internet architecture through incrementally deployable changes.

This short essay explains why we believe such a transformation is possible by addressing *why* an architectural change is needed (Section 2), *what* principles should guide the design of this new architecture (Section 3), *how* these principles could be turned into a concrete design (Section 4) that would benefit users (Section 5), *who* among the stakeholders might benefit (Section 6), and *why* there is a chance of success (Section 7).

This last point is crucial because previous proposals for changing the Internet architecture in any fundamental way have come to naught (see [4] for a recent review of such proposals). However, as we articulate at the end of this essay, there are several reasons to believe that this time is indeed different. Most importantly, our goal is not to change the direction of the modern Internet (as defined by the designs being adopted by the new private networks) but to architecturally incorporate these trends in a way that is aligned with the incentives of major stakeholders, so that the public Internet remains vital. These private networks provide better support for modern applications by implementing more than just end-to-end packet delivery. *Our goal is to allow the public Internet to offer similar capabilities (and eventually much more) in an open (in terms of process and code) and extensible manner.*

2 Why should we care about Internet architecture?

These new private networks are providing superior service to billions of users without any change in the architecture: why isn't that sufficient? That is, why should we care about changing the Internet's architecture if the desired functionality is already being provided without an architectural change? There are two fundamental reasons.

First, these changes are not compliant with the current architecture; packets are being intercepted and processed in ways not consistent with the core IP protocol. Rather than banning such additional processing, which would be unwise given its proven utility in these private networks and there being no obvious way to achieve similar results within the architecture, we want to make this additional processing part of the standard Internet service model. These services would then be uniformly defined and automatically available to all Internet users.

Second, when these services become part of the architecture in an extensible manner, each application can explicitly invoke the desired service via a mechanism provided by the architecture. Since applications will only invoke those services they are compatible with, the functionality of these services can be fully general. However, when additional network functionality is deployed outside of the architecture, then one of the following conditions must hold: any such functionality must be backwards-compatible with all existing applications; or such functionality can only be applied to applications built or deployed by the entity running the private network (who can ensure that the applications are compatible with this additional functionality); or the functionality is deployed by the application provider itself (such as by using cloud-based proxies). The first option greatly limits what functionalities can be provided, the second limits who can provide them, and the third places a substantial burden on application providers.

There are thus three advantages for incorporating new functionality into the architecture: *universality* (as a required part of the architecture, these services are uniformly and automatically available as a basic part of the Internet service), *extensibility* (which allows applications to choose which services to use), and *generality* (which allows complete freedom in designing new services).

Note that the Internet's functionality has long been expanded through middleboxes (*e.g.,* firewalls, WAN optimizers, proxies, and the like). These middleboxes account for roughly a third of the network elements in most networks [22], yet there have been few calls for extending the Internet architecture to incorporate them (but see [23, 24] for some voices in the wilderness). In addition, enterprise networks often have local QoS and security mechanisms that are not part of the standard IP model. These local changes do not threaten the public Internet because they are intended solely for internal users. In contrast, these new extensive private networks are offering their privately-implemented services

[2]The improved security is due to the shorter interdomain paths, which reduce the chance of route hijacking

globally, through widespread deployment of PoPs, to those who use the applications offered by or hosted on these private networks. These services should be made universally available, so the public Internet, whose uniform service model enabled such rapid innovation, is not largely replaced by a set of private networks offering their own privately defined and implemented services.

3 What principles should guide us?

Our task is to design a new and more flexible Internet architecture, but what principles should we use to guide us? We propose four such principles that guide *what* these additional services could be (#1), *where* they are supported (#2), *how* they are initially deployed (#3), and what they are *not* (#4).

#1 Services should be layered on top of packet delivery. We already know how to deliver packets on the Internet, and further improvements in speed and reliability, though desirable, will not transform the Internet. Instead, we should focus on what other services the Internet should offer besides packet delivery, and those services should be deployed on top of packet delivery. This principle is merely a natural application of layering. Just as logical links (L2) are built on top of physical links (L1), and global best-effort delivery (L3) is built on top of L2, we should build all additional services on top of L3. For convenience, we will say such services belong to the *service layer* or L3.5 (as in [16]). Service implementations start at the service layer but may extend up through the application layer.

There is a wide range of possibilities for these services. EI would likely start with the functions currently deployed on private networks (*e.g.,* caching, flow termination, and load balancing). In time, additional functions could provide better security (*e.g.,* protection against DDoS), increased privacy (*e.g.,* anonymization of queries), additional delivery services (*e.g.,* better support for multipoint applications and multhomed clients), and use-case-specific performance improvements for streaming, IoT, gaming, AR/VR, and other common uses. Eventually, some new services could be more architectural in nature, such as providing support for information-centric designs where the interface revolves around publishing and retrieving content as in NDN [25] or DONA [13].

We call the set of additionally offered services the *Internet service model*. While our approach (described in the next section) can support a broad range of architectural proposals and performance enhancements, we envision the service model is not merely the union of all such possibilities but instead is a carefully curated set of services designed to meet a wide variety of application and user needs. This curation, involving discussion and ultimately approval of each service, will require some form of governance (akin to the IETF) that is yet to be determined. Note that an individual service may be composed of several component functions (*e.g.,* a service could combine flow termination and caching). However, we do not envision allowing hosts to invoke the composition of arbitrary services, as in many cases this may not be feasible (*e.g.,* a security-improving service and a multipoint delivery service may be inherently incompatible).

#2: Services need only be implemented at network edges. Good system design requires modularity. Here, we propose an edge-core modularity (as in MPLS), where all additional services need only be implemented at Internet edges on what we will call *service nodes* (SNs), and the Internet core merely provides packet delivery (L3). This limited deployment of service nodes seems sufficient, given the application support provided by current private networks, and also makes deployment easier. Some services may require support on hosts, and in some cases these could be implemented to allow hosts colocated on the same network to communicate directly rather than via a service node.

Thus, EI has a new layering configuration where switches implement layers 1 and 2; routers implement layers 1, 2, and 3; and service nodes and hosts implement all layers (including the new service layer). While service nodes and hosts are equivalent in terms of which layers they can implement, we expect the code deployed on service nodes at the service layer and above to be quite different from that deployed on hosts.

Where is this Internet edge where the service nodes are to be located? As the term edge implies, service nodes are within the Internet, not under host control, so host compromises do not affect their functioning. In addition, the host-to-service-node latencies should be relatively low, and the number of hosts being handled by any single service node should be small enough so the service node can handle per-flow state. These two constraints still allow wide latitude in where service nodes are placed, which will likely depend on the context (*e.g.,* whether they are within a cellular, enterprise, datacenter, or home network). For the purposes of this essay, we will consider the service node placements to be roughly equivalent to the PoPs of large private networks, the central offices (or regional aggregations points) of traditional carrier networks, or within large datacenters (to support the hosts therein). As 5G deployments mature, some service nodes could migrate to cellular base stations.

#3: Services should be initially deployed via open-source software modules designed to run on commodity servers. As has been amply demonstrated by these new private networks, and briefly argued in [16], software processing is sufficient to handle the loads at network edges. This is a major change from two decades ago, and it allows us to adopt this open-source approach to service deployment. Defining a new service not through a detailed specification (which is needed when requiring interoperability between independent proprietary implementations) but through an open-source software module deployed on commodity hardware will speed adoption and deployment by avoiding the delays inherent in hardware development/replacement cycles and in traditional standardization processes.

This does not preclude eventually providing hardware support for services to lower the cost/performance (either in general or for specific tasks such as cryptographic operations). However, these hardware implementations (be they ASICs, FPGAs, programmable routers, or anything else) must be compatible with the open-source code. To wit, the open-source software is the *de facto* standard, and all other implementations must interoperate with it. Thus, our agenda does not require programmable routers or other specialized hardware, but the efficiency of the resulting infrastructure could benefit from their eventual deployment.

#4: Services should focus on simple processing, not general computation. Edge computing [5] has become an important trend, but we do not want to confuse the simple processing (either packet-processing or handling basic requests such as caching) with the general computation offered by edge computing. Our focus here is only on the former as the latter involves a suite of issues that we do not address (*e.g.,* how to charge for computation and what is the right computational environment). However, while the instantiation of each service on a service node only involves simple processing, the control plane of these services could be quite general and supported by nodes other than service nodes, as would be needed to coordinate caches or control load-balancing.

4 What would a resulting design look like?

We now present a design, called the Extensible Internet (EI), that is consistent with these principles. We start by describing the general service paradigm in EI, followed by discussions of eventual deployment and incremental deployment scenarios.

4.1 General Service Paradigm

Interaction Pattern: For convenience, we focus on host-to-host communications when explaining this paradigm, but everything easily generalizes to other delivery models (such as would be needed for DTN [8] or NDN). Each host is associated with one or more service nodes through a discovery process (though usually only using one at a time, so in what follows we assume a single associated service node). For clients, this service node would typically be in a nearby PoP, while for servers within a cloud provider's datacenter the associated service node would be within the datacenter itself. When two hosts are communicating, the packets take a path which (when described at the service layer) goes from the source host, to its associated service node, to the service node associated with the destination host, and then finally to the destination host itself. The packets are tunneled in each hop of this process; the tunneling technology could be chosen on a pairwise basis, but for simplicity we will assume there is a single service-layer tunneling protocol built on IP. Note that this tunneling protocol, and the existence of service nodes (along with their service implementations), are the *only* architectural changes in EI and are sufficient to create the desired extensibility.

However, the Host-SN-SN-Host pattern does not always apply. If the two hosts are within the same L2 network or in different L2 networks but in the same administrative domain, then they could either communicate directly (just as in IP where two hosts can communicate directly without going through a router if they are part of the same L2 network) with the service-specific processing done by the host component of the service, or go through a single service node that they are both associated with, or go through two service nodes (if the hosts are associated with different service nodes). Which of these three cases apply will depend on the service itself (*i.e.,* whether it allows direct communication) and whether the two hosts share the same service node. In addition, some services may want to leverage additional service nodes along the path (*e.g.,* to improve caching or increase privacy), and the service itself can manage the routing between service nodes within a single network. While these other cases may arise in various deployment scenarios, for convenience we will focus on the canonical Host-SN-SN-Host pattern in the ensuing discussion.

Service nodes provide an architecturally-compliant way to insert simple processing inside the network above the IP layer. The fact that the processing occurs at only a few service nodes does not greatly limit what services EI can offer (though it has implications for the resulting performance). In fact, all proposals for new L3 architectures that assume a Host-Router-...-Router-Host paradigm could be implemented at the service layer.

Interfaces: Service nodes need to know which service to apply to an arriving packet. This can be handled by the equivalent of a "Next Protocol" field in the service-layer tunneling protocol. However, in some cases, the decision to invoke a specified service for a certain class of traffic might be made through a management interface between the host (or operator on the host's behalf) and the service node (*e.g.,* to filter incoming traffic for DDoS prevention). This invocation would specify the class of traffic, the intended service, and any configuration state required.

Hosts need to discover their associated service node. As we discuss later, this service node might be supplied by the host's network provider or by a third party. A simple discovery protocol can allow clients to discover nearby service nodes in either case.

Service Node Design: We assume that each service in the service model is instantiated in an open-source code module. The service node must provide an execution environment supporting a few basic primitives (*e.g.,* packet-in/out, read-write state, read-write configuration). This execution environment should be independent of its packaging so it can be supported natively, or in containers, or in VMs. In addition, the service node should handle various orchestration tasks for modules, such as invoking/killing, scaling up/down, failure detection/recovery, and diagnostics. While the execution environment interfaces and the service modules should be universal (except when supporting specialized hardware), the packaging and orchestration can vary between service nodes. Logically separating the execution environment from how it is deployed allows providers to initially use existing orchestration frameworks (for VMs and containers) while enabling an eventual transition to native deployments. Such a separation also allows primitives to be offloaded to hardware more easily.

4.2 Eventual Deployment Options

We now consider what a deployment might look like in the long-term by addressing three questions:

Who deploys the service nodes? To answer this, we define Last Mile Providers (LMPs) as those connecting end users (whether in homes or enterprises) to the Internet (*e.g.,* China Telecom, NTT, Comcast, Deutsche Telekom, Orange, Korea Telecom, and others), and the Cloud and/or Content Providers (CCPs) as those running one of the large emerging private networks (*e.g.,* Facebook, Google, Amazon, Azure, Alibaba, and others). The service nodes can be deployed by LMPs for their own users (presumably in their central offices), or LMPs can contract with one or more

CCPs so the LMP's users can access the CCP's service nodes (presumably in their PoPs). We call the set of providers offering service nodes the Service Node Providers (SNPs).

How are the service nodes interconnected? When two hosts in different networks exchange packets, we require that at the service layer packets go directly from a service node in the source network to a service node in the destination network without passing through service nodes in any intermediate transit networks. Thus, we require that, at the service layer, SNPs exchange packets directly. Moreover, we require that there be no settlement fees for these direct service-layer peerings.

The reason for this last requirement is that settlement fees can be used to charge application providers for access to customers (or the reverse), which can hinder competition (see [12] for a more detailed rationale). Moreover, each user is associated with a service node and is either directly or indirectly paying for the access to that node. Part of an SNP's service commitment is to carry packets to or from the next hop on behalf of its associated users. Therefore, if an SNP feels it is not being properly compensated for doing so, it should increase the charges to its own users for whom it is providing that service, rather than charging peers.

However, we do not expect every SNP to have the backbone infrastructure needed to directly connect to every other SNP at the L3 layer. If two SNPs have no direct L3 connectivity, they can choose to remotely peer at the service layer via a variety of methods including: setting up a tunnel using the existing Internet service, mutually contracting with a transit provider to provide such a tunnel, or both contracting with transit providers to reach the same IXP. How the SNPs share the cost of these remote peering arrangements and how they select between their multiple interconnection points (which presumably will be needed for resilience) is left to bilateral negotiations, though common industry practices might emerge.

Note that this design effectively decouples EI from BGP. BGP could continue to be used to provide interdomain routing at L3, but it is not exposed to EI, which only sees service-layer peering. This provides a much-needed separation of concerns between how packets are carried between domains (which can be any packet delivery technology), and how originating and destination domains deal with packets (which is determined by the Internet's service model). This would also allow the Internet to gradually diminish BGP's role without any change in EI.

What kinds of services are offered, and by whom? At an architectural level, there are a set of *public services* that must be offered at all service nodes. These services are curated and officially form the Internet's service model. There are no additional charges for these services, as they are part of the basic Internet offering.

In addition, we expect that, for a fee, SNPs will allow third-parties (such as application providers, companies selling specialized security services, and groups trialing services hoping to get them adopted as a public service) to offer *private services* at service nodes. These services are invoked using the mechanisms that are part of the architecture but using service identifiers that lie outside the public set of services.

Lastly, cloud providers that are also SNPs could additionally offer *hybrid hosting services*, using their datacenters to run their customers' backend application code and their service nodes to run associated application-specific service code. This combines a private service as above with traditional cloud hosting, and the resulting codesigns could eventually inform future public services.

4.3 Incremental Deployment Options

The deployment described above is what EI might look like in the long term. To get started on the journey towards that vision, we describe three steps that we hope to take in the short-term.

Host implementation: Because universal host support for EI is far in the future, we will produce code that can be embedded in applications (*e.g.,* web browsers and mobile apps) to execute the service node discovery protocol and tunnel packets to service nodes. We will release a version of the Chromium browser incorporating this, which will allow users and application developers to trivially redirect browser-based applications to EI.

Experimental deployment: We hope to use various research testbeds (such as FABRIC [7], EdgeNet [6], and Pronto [17]), along with donated resources from CCPs, to deploy a geographically dispersed set of service nodes. This would allow researchers to freely experiment with implementations of novel services.

Prototype deployment: We are working on a preliminary version of an open-source execution environment that can be run by various cloud providers. Cloud providers who are interested in participating in the prototyping exercise may allow this execution environment (and some service modules) to be run in their PoPs. For other cloud providers, the execution environment can be run on their cloud without any explicit involvement from the cloud provider (except to perhaps offer reduced charges). We hope to gather enough technical and financial support from a collection of cloud providers to create a fairly stable and widespread infrastructure. If this comes to pass, some early services could be stably deployed so application developers can depend on those services and design their software accordingly. In the beginning, this service would be free, and hopefully later turn into a commercially viable infrastructure (see Section 6).

5 How would this make users happier?

The ultimate test of an architecture is how well it, and the applications deployed thereon, meet user needs. EI's process for deploying new services is crucial in allowing it to address user needs. Once a service has been instantiated in an approved (by the curation process) open-source module, it can then be quickly deployed on all service nodes. In the current Internet, potential improvements must first endure a detailed IETF specification process because each protocol must allow independent interoperable implementations. Any L3-related proposals must then be deployed by individual domains, which can result in lengthy delays because such deployments often involve extensive and costly changes in domain infrastructure and operations. In EI, all service nodes must support all approved services, which merely requires installing the associated software module; thus, there will be no substantial time-lag between approval and deployment, thereby speeding the process of meeting user needs.

This is perhaps the most important advantage of EI: although many services we envision EI supporting can be implemented today using some *ad hoc* and/or proprietary combination of cloud and edge resources, EI allows the rapid adoption of such services in an open, universal, and extensible manner.

What do we envision these services to be? As previously mentioned, the first class of services offered will be those that are currently provided on the emerging private networks, perhaps combined with better support for multipoint, multihomed, and/or mobile delivery. More generally, as EI becomes the accepted architecture, application developers can leverage a growing set of universally available in-network services as they design their applications. We do not know exactly what service node support might best suit video streaming, gaming, IoT, AR/VR, and other common use cases, but we expect that at least some of these would benefit from such support. In addition, these public services will be augmented by private services that may be more application-specific.

The fact that service nodes can handle per-flow or per-user state, yet be unaffected by host compromise, could enable improvements in security. As mentioned above, EI could support DDoS prevention measures that allow hosts under attack to "shut-off" their attackers at each attacker's service node (as described in [1, 10]). Going further, a service node can use a capability-based mechanism and software attestation to limit the set of endpoints an application can communicate with (*e.g.,* ensuring that only an uncompromised webserver can send requests to a database server). As EI becomes established, researchers will doubtless find new ways to improve security via these service nodes.

These service nodes could also provide a public discovery service, redirecting clients to edge computation and private services (*i.e.,* functions that are not part of the architecture could register their existence with the architecturally-supported discovery service), easing the deployment of new functionalities by making them easier to discover and invoke.

In the longer term, EI could support services that are more architectural in nature. For instance, as mentioned above, a service could support an information-centric design such as NDN or DONA that provides an entirely new service abstraction. Or, less radically, one could create an architectural service that provides a more secure and private name resolution design to replace DNS (*e.g.,* similar in spirit to oDNS [21]). Of course, we cannot predict how these new architectures would improve the user experience, but our point is that we will have greatly improved our ability to deploy such architectures, so that future research could have a greater impact on the Internet service model than it does today.

6 Could this reshape the Internet?

Assuming we are able to achieve a prototype deployment of EI as described previously, what market forces would move us towards a more permanent deployment? EI would provide three benefits to CCPs. Through its support for explicit service invocation, EI will facilitate a growing market for private services that could generate additional revenue. EI also allows the CCPs to share the burden of providing the public services that create a better Internet for all. The desire to share this burden may particularly apply to content providers, where content is king and the necessary infrastructure to achieve good performance is not where they differentiate themselves. More generally, EI gives the CCPs a larger role in the Internet infrastructure, as they would likely be leading the way in deploying service nodes and the backbone interconnections among them.

EI would be a mixed blessing for the traditional carriers. While it makes the CCPs more of a threat, EI would also provide a community-led solution for the kinds of software services that carriers have struggled to develop, standardize, and deploy. Moreover, their basic Internet offerings would immediately become functionally equivalent to those of the private networks because the public services would be common to both. Lastly, when it comes to deployment of EI, traditional carriers have a huge installed base of central offices that could easily house service nodes. Thus, carriers would get the benefits of EI's better Internet service without having to be the technical leaders in its development.

7 Why is this time different?

There have been many clean-slate proposals for radically transforming the Internet architecture, yet its architecture remains largely unchanged. Why do we believe our approach will have better success? We think there are four factors.

First, the set of Internet providers has changed. The classic large-scale carriers are struggling to adapt to the increased role of software in modern networking. At the same time, a new generation of cloud and content providers have arisen who run their own global private networks and are experts in developing and managing software at scale. Thus, we now have a set of incumbents technically prepared to lead the (r)evolution.

Second, we are not proposing another one-size-fits-all architecture to replace the current Internet but instead are making the current Internet extensible, through minimal architectural changes, so it can support a growing variety of public services, with an initial focus on providing immediate consumer value through better application performance, security, and privacy. EI also enables longer-term architectural evolution; while not the initial driver for adoption, this could be the EI's most important legacy.

Third, software processing is now sufficient to enable the rapid deployment of new services at service nodes. This allows the Internet to evolve at the speed of open-source software, not at the speed of standards-driven protocols or hardware upgrades. Moreover, this software processing is only implemented in service nodes, so the vast bulk of the Internet infrastructure remains completely unchanged by the transition to EI.

Fourth, and most importantly, this approach is not advocating a radical change in what the actual Internet does, only a radical change in what the public Internet officially provides. For almost as long as the Internet has been commercial, the functionality of the basic architecture has been augmented with additional privately-defined functionality such as on-path caches. We think it is time to bring these developments within the architecture, so they can be supported by the public Internet in an open, extensible, and universally available manner. As such, our proposal is the opposite of an academic pipedream; it is the incorporation of current commercial practices into the paradigm of the public Internet, and doing so in a way that is aligned with the interests of users and some major stakeholders in the Internet ecosystem.

8 Why are we writing this position paper?

The Internet community has become inured to the public Internet's architectural stagnation and now greets talk of transformation with well-earned skepticism if not outright scorn. However, for the above reasons, we think this time may be different. In our view, the confluence of industry trends, technical developments, and customer needs have produced a rare opportunity to transition to a fundamentally extensible Internet architecture. This is not a revolution overthrowing the Internet's current architecture but an incrementally deployable evolution that allows the public Internet to incorporate a broader scope of functionality. Success will require a community effort, and we are writing this note with the aim of enlisting others to help make this vision a reality.

9 Acknowledgments

The authors gratefully acknowledge support from DARPA contract HR001120C0107 and NSF grants 1836872, 18311698, and 1817115. The opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *ACM SIGCOMM*, 2008.

[2] T. Arnold, J. He, W. Jiang, M. Calder, I. Cunha, V. Giotsas, and E. Katz-Bassett. Cloud Provider Connectivity in the Flat Internet. IMC, 2020.

[3] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, Oct. 2016.

[4] D. Clark. *Designing an Internet*. The MIT Press, 2018.

[5] Edge computing. https://www.cloudflare.com/learning/serverless/glossary/what-is-edge-computing/.

[6] EdgeNet testbed. https://edge-net.org.

[7] FABRIC Testbed. https://fabric-testbed.net.

[8] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, page 27–34, New York, NY, USA, 2003. Association for Computing Machinery.

[9] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Intelligent design enables architectural evolution. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011.

[10] S. Guha, P. Francis, and N. Taft. Shutup: End-to-end containment of unwanted traffic. Technical Report http://hdl.handle.net/1813/11101, January 2008.

[11] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste. XIA: Efficient support for evolvable internetworking. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 309–322, Berkeley, CA, USA, 2012. USENIX Association.

[12] Y. Harchol, D. Bergmann, N. Feamster, E. Friedman, A. Krishnamurthy, A. Panda, S. Ratnasamy, M. Schapira, and S. Shenker. *A Public Option for the Core*. In *ACM SIGCOMM*, 2020.

[13] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and beyond) Network Architecture. In *ACM SIGCOMM*, 2007.

[14] T. Koponen, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. B. Godfrey, N. McKeown, G. Parulkar, B. Raghavan, J. Rexford, S. Arianfar, and D. Kuptsov. Architecting for innovation. *SIGCOMM Comput. Commun. Rev.*, 41(3):24–36, July 2011.

[15] C. Labovitz. Pandemic impact on global internet traffic. https://storage.googleapis.com/site-media-prod/meetings/NANOG79/2208/20200601_Labovitz_Effects_Of_Covid-19_v1.pdf/. NANOG 2020.

[16] J. McCauley, Y. Harchol, A. Panda, B. Raghavan, and S. Shenker. *Enabling a Permanent Revolution in Internet Architecture*. In *ACM SIGCOMM*, 2019.

[17] Pronto Project. https://prontoproject.org/, 2020.

[18] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker. Software-defined Internet Architecture: Decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 43–48, New York, NY, USA, 2012. ACM.

[19] B. Raghavan, T. Koponen, A. Ghodsi, V. Brajkovic, and S. Shenker. Making the Internet More Evolvable. Technical report, International Computer Science Institute, September 2012.

[20] A. Saeed, V. Gupta, P. Goyal, M. Sharif, R. Pan, M. Ammar, E. Zegura, K. Jang, M. Alizadeh, A. Kabbani, and A. Vahdat. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 735–749, New York, NY, USA, 2020. Association for Computing Machinery.

[21] P. Schmitt, A. Edmundson, A. Mankin, and N. Feamster. Oblivious dns: Practical privacy for dns queries. In *Proceedings of the Applied Networking Research Workshop*, 2019.

[22] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proceedings of the 2012 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, SIGCOMM '12, pages 13–24, New York, NY, USA, 2012. ACM.

[23] J. Sherry, D. Kim, S. Mahalingam, A. Tang, S. Wang, and S. Ratnasamy. Netcalls: End Host Function Calls to Network Traffic Processing Services. UC Berkeley Technical Report No. UCB/EECS-2012-175. http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-175.html, 2012.

[24] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation - Volume 6*, OSDI'04, page 15, USA, 2004. USENIX Association.

[25] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named Data Networking. *ACM SIGCOMM Comput. Commun. Rev.*, 44(3), July 2014.