Intelligent Design Enables Architectural Evolution

Ali Ghodsi
KTH / UC Berkeley

Teemu Koponen
Nicira Networks

Barath Raghavan
ICSI

Scott Shenker
ICSI / UC Berkeley

Ankit Singla
UIUC

James Wilcox
Williams College

ABSTRACT

What does it take for an Internet architecture to be evolvable? Despite our ongoing frustration with today's rigid IP-based architecture and the research community's extensive research on clean-slate designs, it remains unclear how to best design for architectural evolvability. We argue here that evolvability is far from mysterious. In fact, we claim that only a few "intelligent" design changes are needed to support evolvability. While these changes are definitely nonincremental (i.e., cannot be deployed in an incremental fashion starting with today's architecture), they follow directly from the well-known engineering principles of indirection, modularity, and extensibility.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Economics, Management

Keywords

Internet Architecture, Evolution, Innovation, Diversity

1 Introduction

The current Internet is *architecturally rigid*, in that there are core aspects (such as IP) that are particularly hard to change. This rigidity has prevented us from addressing several important problems in the current Internet. Blocked from making practical progress, some members of the research community sought to make intellectual progress by focusing on clean slate designs. The clean-slate approach enables the research community to investigate how to address these problems architecturally, even if the changes themselves cannot be incrementally deployed.

These clean slate efforts have increased our intellectual understanding of Internet architecture — particularly in the areas of security, reliability, route selection, and mobility/multihoming — and there are now many clean-slate proposals that offer significantly better functionality than our current Internet. However, in recent years these clean slate efforts moved beyond adding functionality and have begun addressing the reason we had to turn to clean slate design in the first place: architectural rigidity. That is, the research community is now devoting significant attention to the design of architectures that can support architectural evolution, so that future architectural changes could be made incrementally rather than through a series of clean-slate designs. For instance, two of the recent NSF FIA proposals (Nebula [3] and XIA [1]) have identified evolvability as an important aspect of their design. And even more recently, we introduced a new "framework" approach to evolution called FII [12].

However, this emerging literature on evolvability is relatively immature and certainly incomplete. We, as a community, are only beginning to understand what makes an architecture evolvable. In this paper we hope to advance the dialog on this topic by reviewing current barriers to evolution and then observing that relatively simple, but "intelligent" (and definitely nonincremental), changes would vastly improve the Internet's evolvability. These changes are similar to those outlined in [12] for FII. While the two designs are not identical (in particular, our thoughts about interdomain routing and other services, as well as our security story, have changed significantly since [12]), the more fundamental difference between [12] and our discussion here is that in [12] the design was motivated by high-level architectural arguments about architectural "anchors" and other general principles; here we arrive at similar design decisions by applying straightforward and well-known principles from systems design: layers of indirection, system modularity, and interface extensibility. We are encouraged (and somewhat relieved!) that we have arrived at similar conclusions about evolvability from both a top-down architectural analysis and a bottom-up systems design approach.

We begin this paper (Section 2) by discussing what we mean by architectural evolvability, which is *ongoing and pervasive architectural change* rather than episodic and/or narrow changes. We then (Section 3) discuss several examples of architectural evolution, outlining why it is hard in today's architecture and how we could alter the current architecture (in rather straightforward ways) to accommodate each particular form of architectural evolution. These examples lead us to a design that meets our definition of an evolvable Internet; we call this design OPAE (for Ongoing and Pervasive Architectural Evolution).[1] We identify the basic design principles underlying this design in Section 4 and contrast this with other approaches to architectural evolvability in Section 5, paying particular attention to the XIA proposal because articulating the differences between OPAE and XIA helps illuminate the underlying assumptions and ambitions of both.

2 Comments on Architectural Evolution

Before addressing architectural evolution, we first address its opposite: architectural rigidity. For all the talk in our introduction, and in many other papers (perhaps the most extreme articulation is in [4]), about architectural rigidity, it is important to state the obvious: the Internet is wonderfully dynamic, with change constantly occurring in many areas. For instance, we easily deploy new services at the application level (*e.g.*, Akamai, Facebook); with somewhat less ease, we introduce new host protocols (*e.g.*, HTTP, SIP, and BitTorrent); and after a lengthy standards process we introduce new technologies below the IP layer (from wireless to optical to new L2 technologies and protocols). No one could claim that the Internet is ossified in these respects.

However, there are core aspects of the Internet architecture that are indeed hard to change in any fundamental way. Changing the basic nature of IP, TCP, DNS, or the Sockets API (rather than merely extending them, as in DNSSEC) is quite difficult, and this rigidity has made it harder to address many of the Internet's longstanding problems (such as security and reliability). But even for these core protocols, one could imagine managing a transition given sufficient time and effort; the (perhaps!) forthcoming IPv4 to IPv6 transition is an example of this, where it has taken massive governmental exhortation and commercial investment to have gotten this far. But such transitions will be infrequent, because of the extreme degree of effort involved.

When we use the term architectural evolvability, we are referring to the ability to make fundamental changes in the entire architecture, not just the aspects that are easy to change today. That is, we want evolution to pervade the architecture, not be narrowly confined to its edges. Moreover, we want these changes to be relatively easy and frequent rather than extremely difficult and rare. *In short, we want an Internet where architectural innovation is ongoing and pervasive.*

There are many actors involved in adopting an architectural change, ranging from domain operators (*e.g.*, ISPs), router/switch vendors, OS vendors, application writers (*e.g.*, Microsoft, Adobe), and application service providers (*e.g.*, Facebook, CDNs, and search). Application service providers will perhaps be the most nimble of these actors, deploying new systems frequently and independently, so we assume that they are not a barrier to evolution. We further assume that all major OS vendors and most (but not all) application writers will adopt new protocols relatively quickly (*e.g.*, dual-stack support has been available in OSes for many years), but we also assume that there will always be legacy systems and applications that have not been upgraded and will continue to use older versions. Note that when we say "adopt" we mean that they will *add* support for these newer protocols, but will retain the ability to use the older protocols (and will probably keep these older protocols as the default for quite a while).

The same willingness (with somewhat less nimbleness) to adopt additional software features applies to router/switch vendors. Hardware changes are slower in coming (but not impossible: hardware support for IPv6 is widespread), and we will discuss the role of hardware in architectural evolution in Section 5. Lastly, the domain operators are probably the most conservative actors in this list, as the penalties for malfunctions are higher than the rewards for new functionality.

Finally, any change that requires adoption by all domains faces a daunting deployment barrier. We also assume that requiring widespread deployment of middleboxes that can translate between architectures is a significant barrier to architectural change. Thus, in what follows we assume that supporting ongoing and pervasive evolution must not require universal adoption or an extensive transition infrastructure.

3 Overcoming Barriers to Evolution

In this section we consider various kinds of architectural changes that are hard to effect today[2]; this list is not meant to be exhaustive, but comprehensive enough to illustrate most aspects of architectural evolution. Moreover, this is a list of changes to individual components (such as naming or routing); most architectural proposals involve changes to a set of these components, but for clarity we focus on these components individually. For each of these categories

[1]Opae is Hawaiian for shrimp, which is fitting because OPAE is a minimal design, standardizing very few aspects of the Internet architecture. As noted before, this is almost the same as the FII design in [12]; we use a different name here to avoid confusion because the two designs differ in their treatment of Interdomain routing and security.

[2]That is, we ignore deployment of new applications, new application-level services, new host protocols, and new L2 (and lower) technologies and protocols. Note that deployment of new designs inside a domain may require support from router/switch vendors and OS vendors. What it does not require is universal agreement among domains.

of changes, we first describe what prevents incremental adoption of these changes now and then discuss how the overall architecture might be modified in order to enable such evolutionary changes in the future. This will lead us to the OPAE design.

We hasten to note that *all* of these necessary modifications are known in the literature (at least in some form)[3], many of them are obvious, and some are even partially implemented today. We do not contend that the solution we propose here is novel; quite the opposite, *our claim is that almost no novelty is needed to support architectural evolution.*

But just because they are not novel does not mean they are easy to deploy. OPAE is most definitely a radical departure, in that it would require changes in several crucial places in the current architecture, but there is nothing conceptually interesting in these changes; they are both clean-slate *and* boring.

We divide these changes into four categories: easy, hard, complicated, and security.

3.1 Easy Architectural Changes

We start with the easiest case, changing the network API (hereafter, netAPI) offered by the host OS to applications. The netAPI is hard to change because it is embedded in applications; any nonbackwards-compatible change would immediately render all legacy applications unusable. This can be avoided by merely providing a layer of indirection — essentially a netAPI identifier — so that all calls to the netAPI first specify which version or flavor of the netAPI they want. This is no different, in spirit, from IP's use of a version number at the beginning of the packet header; netAPI calls start with a netAPI identifier with the following bits obeying the syntax of that particular netAPI (which could be publish/subscribe, Sockets, a streaming interface, or anything else). In fact, some netAPIs already support more limited forms of this indirection (*e.g.*, protocol families). It is easy for OS vendors to support multiple netAPIs, and the set of netAPIs on one host need not be identical with those on another, so little coordination is needed to deploy a new netAPI with this layer of indirection.[4]

Changing the naming system offers a similar challenge, and yields to a similar solution. Currently DNS names are embedded in applications, so any fundamental naming change would require the modification of many applications. This could be avoided by having applications (as some do now) treat names as semantic free bags-of-bits and only handle them via naming operations implemented in the network stack (such as `gethostbyname`). However, the stack must also change to adopt to these new names. To do so, one can take an approach similar to the netAPI, with all names taking a standard form of a namespace identifier followed by the bits representing the name. When a new namespace is introduced OS vendors can start including network stack support for these names (which can be recognized by the namespace identifier); the stack can support multiple namespaces simultaneously, so as with the netAPI one can add additional namespaces without revoking old ones.

However, such names must be resolved, so that requires the introduction of new name resolution mechanisms, which can be used to support a new namespace, or augment (or replace) the name resolution mechanism for an existing namespace. This requires OS stacks to support new resolution protocols, but also involves some coordination with domains, in that the domain must provide a way for a host's stack to reach the various name resolution systems. This could be in the form of a resource directory, or be bootstrapped by having hosts use an old name resolution system to reach a well-known service (*e.g.*, Google), and this well-known service could have a directory of the name resolvers for this new name resolution system.

3.2 Hard Architectural Changes

Perhaps the hardest architectural change is to modify IP. Currently, L3 provides both a universal exchange format and a universal addressing scheme. As we have seen with IPv6, keeping this dual use for L3 but replacing IP with a new version requires (among many other changes) changing applications (so they can deal with these new addresses), achieving universal agreement on the new protocol, and deploying an overlay infrastructure (so that IPv6 networks can talk directly with other IPv6 networks). These are significant barriers, so we must find another approach.

First, and most obviously, we require that all netAPIs pass names, not addresses, so applications are shielded from changes at L3. Second, we separate intradomain addressing from interdomain addressing. Interdomain addresses could just be domain identifiers, or could be at a smaller granularity, but the important factor is that they do not identify a particular host, only a region within which a host resides. A full destination address consists of an interdomain part and an intradomain part, so the intradomain part need not be understood by any domain other than the destination domain (see [2]). This is nothing more than making intradomain addressing look like L2 (which today need not be coordinated across domains).

Third, we note that once there is an interdomain addressing structure that is independent of a universal exchange format, interdomain addresses can be embedded in many forms of packet delivery. This frees two domains to peer with a variety of technologies and layers (such as optical, MPLS, or Ethernet); different domains can choose to peer in different ways, and a single domain might employ many different peering technologies simultaneously. Thus, this separation of interdomain and intradomain addressing, by enabling general forms of peering, not only makes changing the L3 protocol easy, it does away with the concept of a universal networking

[3]We do not cite each of these precursors because there are so many of them, but we want to make clear that these are not *our* ideas.

[4]Note that two communicating hosts can use different netAPIs, as long as the implementing protocols they use are compatible.

layer altogether. What will hold the Internet together is not a universal exchange format but the universal form of interdomain addressing and, as we discuss below, a way to route based on these interdomain addresses.

3.3 Complicated Architectural Changes

The most complicated aspect of an architecture to change is interdomain routing. It not only requires global agreement and adoption among the domains (similar to changing IP) but it also involves the extra complication of a globally distributed algorithm (whereas IP can be executed locally) with sophisticated semantics (BGP policies are far more complicated than basic IP forwarding). Achieving global agreement on an ongoing series of complicated changes to BGP is infeasible, so we need to find another way to evolve interdomain routing. We propose doing so on two timescales, each with its own mechanism: we use a tag to specify the routing protocol used, so that interdomain routing can evolve, and we make routing more extensible so that this form of evolution need not happen often.

Routing Extensibility. On short timescales, we advocate extremely extensible routing designs. In particular, routing systems should support arbitrary (and possibly external) route computations, a flexible policy model, innovations in the kinds of services offered along paths (such as QoS, middlebox services, monitoring), multipath routing, and accommodate various peering technologies (*i.e.*, not require a universal protocol is used to connect domains). This degree of extensibility would relieve pressure on developing entirely new interdomain routing systems by allowing much of the desired functionality to be achieved within the current design.

Another desirable feature of interdomain routing schemes is path visibility and choice. With traditional interdomain routing, hosts are only offered the default path; if that path is not acceptable, there is no recourse. With routing systems that allow hosts (or some route computation agent) to choose the path (as long as it is policy-compliant), there only needs to exist one acceptable path. Giving the hosts visibility into the properties of paths and choice among them significantly decreases the deployment hurdle for new path properties, changing the requirement from altering the default path to merely providing one acceptable path.

There are policy-compliant source routing designs that satisfy these extensibility and choice goals; we won't describe them here, but see [9,12] for a description of one such routing scheme.

Routing Evolution. On a longer timescale, we should enable partial deployment of new interdomain routing systems, running along side the current one. Such an incremental deployment can be achieved by having (i) packets carry a label that indicates which routing system computed the route, with the rest of the header understandable only by routers that participate in that routing system, and (ii) the routing systems expose which domains are reachable via that routing system (as BGP does today). In this way, whomever is choosing the route (end host, domain, external route computation agent, etc.) can inspect the various routing systems with their partial deployments and choose a route from the desired routing system. Deploying new routing systems with any reasonable degree of coverage is hard, so we expect this form of evolution to be relatively slow.

More General Services. So far, our discussion of interdomain routing has focused on designs that maintain the current Internet service model of sending and receiving packets to destinations (which themselves might be unicast, multicast, or anycast). However, once we insert this level of indirection (through the label in the packet), the service offered by a new "routing" system need not resemble traditional routing; for instance, such a mechanism might offer a pub/sub interface (as in DONA [11], CCN [10], and other Information-Centric designs). Thus, OPAE's general approach enables the incremental introduction of new interdomain service models.

3.4 Security

We, and many others, have argued elsewhere (see [2,12,15]) that, except for denial-of-service, *all* aspects of network security can be implemented at the end hosts. We adopt that view here as a fundamental architectural fact-of-life, and limit our discussion to dealing with denial of service. We require that every new service (whether routing for traditional destination based packet delivery, or new service models like pub/sub) provide an interface that ensures protection against denial of service. For traditional packet delivery, this interface could *enable delivery* (for capability-like approaches to DoS, such as in [16]) or *prevent delivery* (for filter-based approaches to DoS, such as in [5,12]). Thus, for a domain to support a new routing service (or new service model), it must support both the delivery *and* DoS interfaces.

4 Design Principles

We have described the changes needed to enable ongoing evolution of the network API, naming and name resolution, the internetworking layer, interdomain routing, new service models, and security (DoS). This is in addition to the kinds of evolution that are relatively easy today: applications, application-level services, host protocols, and L2 (and lower) technologies and protocols. This qualifies, we think, as *pervasive* evolution.[5] However, our discussion of architectural barriers and their potential solutions was somewhat *ad hoc*. To provide a more coherent picture, we now review the OPAE's design in terms of three basic design principles.

Layer of Indirection for Flexibility. For three aspects of the architecture which are currently hard to change, we merely inserted a simple layer of indirection that can be seen as little more than a version number which precedes (and

[5]See [12] for a discussion of how to evolve congestion control, which we did not have space to describe.

dictates the structure of) all the content to follow. For the netAPI, new API syntax and/or semantics can be introduced by merely defining a new "version". When implemented by the relevant OS vendors, updated applications can use this new netAPI, while old applications can continue to invoke older versions without disruption.

Names adopt a similar structure, a namespace identifier followed by the content of the name, so new naming systems can be introduced by defining a new namespace which must be supported by the stacks of the relevant OS vendors and have a deployed name resolution infrastructure. New namespaces need not be understood by applications, nor by any network-level entities, just the OS networking stacks and the resolution infrastructure.

Interdomain routing adopts a layer of indirection in a somewhat more complicated way. When the host networking stack sends a packet (a similar story holds for connection oriented services like optical), the packet must contain a "routing" identifier which indicates which interdomain routing system is responsible for routing this packet. That routing system will direct the packet to routers which understand the rest of the header.

Modularity to Minimize Scope Changes. Key to easing evolution is enforcing a tight modularity so that changes only have a limited scope of impact. For instance, the separation of intradomain and interdomain addressing (which was, in fact, part of the original IP addressing scheme!) allows domains to change their addressing schemes independently of each other. Using clean netAPIs (which pass names, not addresses) enables changes in domain and interdomain addresses without changing applications; and having applications rely on network stacks to process names means that applications need not understand the namespaces they are dealing with.

Extensibility to Accommodate New Functionality. Rather than rely on heavyweight replacement of interdomain routing in order to achieve new functionality, we advocated building an extreme degree of extensibility into the interdomain routing system itself. That is, while evolution is possible, extending extensible protocols is far easier and should be the adoption path of choice when possible.

We did not give a complete description of OPAE; there are many smaller issues to deal with (such as bootstrapping and negotiation); see [12] for more details. However, the remarkable aspect of OPAE is that it involves no novel design or deep architectural insight. If the Internet were any other computer system, we would have considered these design choices to be obvious (and long overdue).

As mentioned earlier, the OPAE design is similar to the FII design in [12], the main difference being the far more general approach to interdomain routing and security in OPAE. More fundamentally, the line of reasoning pursued here is quite different from that in [12]. The discussion in [12] is very much top-down, pontificating about grand architectural themes; here, our focus was bottom-up, looking at individual barriers to evolution and engineering ways to overcome them. The fact that both the top-down and bottom-up approaches arrived at very similar results is an encouraging sign.

5 Comparison with Other Approaches

This is by no means the first paper to discuss architectural evolution. Overlays are often mentioned as one way to achieve architectural evolution (*e.g.*, see [13]). However, while overlays make it easy to deploy a new architecture on top of an old one, they do *nothing* to make it easier for two architectures to interact. That is, one can deploy new architectures — say, AIP — using an overlay, but this only allows AIP hosts to talk to other AIP hosts; the overlay does not enable an AIP host to exchange packets with an IPv4 host. Thus, while overlays are useful for experimental deployments (particularly virtualized overlays like GENI [8]) and to achieve a wholesale replacement of one architecture by another (a grindingly slow process), they do not enable pervasive and ongoing change.

Active networks [14] enables certain forms of architectural evolution, particularly on the datapath, but it does not address architectural aspects such as naming and interdomain routing. However, a flexible datapath is extremely important for enabling architectural evolution without requiring new hardware. Whether active networks, or the more feasible but less flexible Software-Defined Networking, or some other approach is adopted, a flexible datapath is necessary for making architectural evolution (of the datapath) economically feasible.

Nebula [3] offers a great deal of extensibility in network paths and services, which is an important dimension of evolvability. However, the core of the architecture (*i.e.*, the datapath) is universal within and across domains; it is not clear how independently domains can evolve internally.

Plutarch [6] represents an entirely different approach to evolvability, stitching together architectural contexts, which are sets of network elements that share the same architecture in terms of naming, addressing, packet formats and transport protocols (IPNL [7] had a similar approach for mapping between addressing realms, but assumed that the realms were otherwise similar). This inter-context stitching is done by interstitial functions, which map between the different architectures. To some extent Plutarch focuses more on supporting static heterogeneity, while our goal is dynamic evolution. Needing to design and deploy interstitial functions between all pairs of architectures, while reasonable for a fairly static collection of architectures, would not qualify as supporting ongoing change.

The XIA proposal [1] is the most recent entry into the evolutionary sweepstakes. It is hard to briefly summarize such a comprehensive proposal, but we hope to capture the flavor of its design here because it provides an instructive contrast to OPAE. XIA enables the introduction of new services through the definition of *principals*. To cope with partially deployed services, XIA relies on a directed acyclic graph in the packet header that allows the packet to "fall back" to other services that will (when composed) provide the same service. For instance, a DAG could have paths for CCN [10] and a source route over IP addresses, with edges permitting intermediate routers to pick either means of reaching the data.

Thus, XIA's main approach to partial deployment (which is a key step in enabling evolution) is to allow transitions between architectures at network elements that understand both (*i.e.*, an element that understands CCN and source-routed IP could take a request sent on one architecture and pass it to the other). The emphasis on building and using bridges between architectures according to a DAG-like structure applies on all levels, from low-level routes (take this next-hop or that one) to high-level services (use CCN [10] or use DONA [11]).

In contrast, OPAE attempts to make low-level packet delivery seamless (*i.e.*, domains using different architecture can directly exchange packets). This allows domains to deploy different architectures without any disruption in service. Once low-level packet delivery is established, all higher-level services (such as a pub/sub service) are globally reachable. OPAE makes no attempt to bridge or otherwise smooth over differences in high-level services (*i.e.*, it does not require any network element to understand both). Instead, OPAE relies on the Internet's record of success in evolving high-level services through parallel deployment and concentrates on making sure they are globally reachable (because parallel deployment only works as an evolutionary strategy if they are globally reachable).

6 Discussion

As noted earlier, our proposal for an evolvable architecture, OPAE, follows three basic design principles that are familiar to any systems designer: layers of indirection for flexibility, modularity to limit the impact of changes, and extensibility to reduce the pressure on architectural change. The only surprising thing about OPAE is that there is nothing surprising in it. If we have made architectural evolution boring, then we have succeeded in our quest. Our central point is that for the Internet, intelligent design and evolution are not only compatible, they are synonymous.

But lurking beneath the topic of evolution is a deeper contradiction. What does it mean to design something that can change? What, exactly, are you defining, when what you are defining is not static? We believe that to have any serious discussion of architectural evolution, one must first identify the component of the architecture that will *not* change. In [12] this was called the architectural *framework* and we adopt that terminology here. The framework is the oasis of constancy that enables the rest of the architecture to evolve. Without this solid architectural ground on which to stand, one cannot hope to make systematic statements about architectural evolvability; *i.e.*, statements about the limits and prospects for architectural evolution only make sense once one has defined what will not change. We believe that one of the deepest questions facing the research community is: what is the minimal set of design components that must be held fixed so as to foster ongoing and pervasive architectural evolution?

Our investigation of alternative approaches to evolution revealed that there are at least two different schools of thought. The XIA design (and Plutarch before that, at a more primitive level) attempt to build and leverage bridges between architectures (network elements that understand more than one architecture). OPAE makes a clean distinction between low-level bit transport, which it seeks to make seamless, and higher-level services (such as content retrieval) which it assumes will evolve using the more traditional method of parallel deployment. It is far too early to tell which is the better choice, but at least now we have realized that there is more than one way to evolve.

7 References

[1] A. Anand, F. Dogar, D. Han, B. Li, H. Lim, M. Machadoy, W. Wu, A. Akella, D. Andersen, J. Byersy, S. Seshan, and P. Steenkiste. XIA: An Architecture for an Evolvable and Trustworthy Internet. Technical Report CMU-CS-11-100, CMU, February 2011.

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

[3] T. Anderson et al. NEBULA - A Future Internet That Supports Trustworthy Cloud Computing. `http://nebula.cis.upenn.edu/NEBULA-WP.pdf`.

[4] T. E. Anderson, L. L. Peterson, S. Shenker, and J. S. Turner. Overcoming the Internet Impasse through Virtualization. *IEEE Computer*, 38(4):34–41, 2005.

[5] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and Delay Accountability for the Internet. In *Proc. IEEE ICNP*, 2007.

[6] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. Plutarch: An Argument for Network Pluralism. In *Proc. of SIGCOMM FDNA*, 2003.

[7] P. Francis and R. Gummadi. IPNL: a NAT-extended Internet Architecture. In *Proc. of SIGCOMM*, 2001.

[8] GENI: Global Environment for Network Innovation. `http://www.geni.net/`.

[9] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet Routing. In *Proc. of SIGCOMM*, 2009.

[10] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking Named Content. In *Proc. of CoNEXT*, 2009.

[11] 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 *Proc. of SIGCOMM*, 2007.

[12] 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. *ACM SIGCOMM Computer Communications Review*, 41(3), 2011.

[13] S. Ratnasamy, S. Shenker, and S. McCanne. Towards an Evolvable Internet Architecture. In *Proc. of SIGCOMM*, 2005.

[14] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1), January 1997.

[15] D. Wendlandt, I. Avramopoulos, D. Andersen, and J. Rexford. Don't Secure Routing Protocols, Secure Data Delivery. In *Proc. HotNets*, 2006.

[16] X. Yang, D. Wetherall, and T. Anderson. A DoS-Limiting Network Architecture. In *Proc. SIGCOMM*, 2005.