Architecting for Innovation

Teemu Koponen[a], Scott Shenker[b], Hari Balakrishnan[c], Nick Feamster[d], Igor Ganichev[e], Ali Ghodsi[f], P. Brighten Godfrey[g], Nick McKeown[h], Guru Parulkar[i], Barath Raghavan[j], Jennifer Rexford[k], Somaya Arianfar[l], and Dmitriy Kuptsov[m]

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 It has NOT been peer reviewed.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

We argue that the biggest problem with the current Internet architecture is not a particular functional deficiency, but its inability to accommodate innovation. To address this problem we propose a minimal architectural "framework" in which comprehensive architectures can reside. The proposed Framework for Internet Innovation (FII) — which is derived from the simple observation that network interfaces should be extensible and abstract — allows for a diversity of architectures to coexist, communicate, and evolve. We demonstrate FII's ability to accommodate diversity and evolution with a detailed examination of how information flows through the architecture and with a skeleton implementation of the relevant interfaces.

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

Over the past few decades the Internet has unleashed an unprecedented wave of innovation. Below the IP layer, an amazing array of new networking technologies — from wireless to optical — has greatly expanded the Internet's capacity and reach, while above the IP layer a string of unforeseen applications — from the web to social networks — has transformed our lives. Throughout this period, however, the basic Internet architecture (*i.e.*, IP, TCP, DNS, and later BGP) has remained relatively unchanged. This architectural stability was crucial in fostering the development of new applications and networking technologies by giving the former a stable base upon which to build and giving the latter a fixed set of requirements to support.

However, in recent years this architectural stability has become a liability, as there are areas of increasing importance where the original Internet architecture falls short. Most notably, the current Internet does not provide the level of security or availability required from such a critical infrastructure nor does it incorporate adequate mechanisms for privacy, mobility, accountability, middleboxes, and data-oriented functionality. Incremental changes to various Internet protocols have not fixed these and other architectural deficiencies, so in recent years there has been a focus on developing "clean slate" redesigns of the Internet architecture that rectify one or more of these aforementioned problems through non-incremental changes. Clean slate redesigns typically propose architectures that would be as static as our current one, presumably lasting relatively unchanged for a generation or more.[1] Therefore, these clean-slate designs must not only meet current needs but also any future requirements that might arise in the next few decades. Predicting these requirements is a daunting challenge, one that our community has not fared well at in the past.

We too are proposing a clean-slate redesign, but one that largely avoids having to predict the future. Our approach is motivated by the observation that the literature already contains many proposals that would improve the Internet architecture; for instance, we know how to design an Internet that would be more secure (*e.g.*, [4, 41, 43]), more data-oriented (*e.g.*, [20, 23, 24]), and provide better support for mobility and middleboxes (*e.g.*, [7, 30]). Unfortunately, these architectural changes have not been incorporated into the Internet architecture because they face insurmountable deployment barriers, barriers which (as we discuss later) are largely due to the current Internet's lack of architectural modularity. Thus, the persistence of the Internet's architectural deficiencies is not because they are intellectually intractable, but because the current architecture puts them beyond the reach of incrementally deployable changes.

In this paper we attempt to solve this problem of architectural rigidity — the inability to solve problems through incrementally deployable solutions. More specifically, our goal is to design an Internet that supports architectural *evolution* (changes over time) and *diversity* (variations over space, so not all portions of the Internet must deploy exactly the same architecture) without massive

[a]Nicira Networks.
[b]ICSI / UC Berkeley.
[c]MIT.
[d]Georgia Tech.
[e]UC Berkeley.
[f]KTH / UC Berkeley.
[g]UIUC.
[h]Stanford.
[i]Stanford.
[j]ICSI.
[k]Princeton.
[l]Aalto.
[m]HIIT.

[1]See Dovrolis [33] for similar comments.

disruption in the infrastructure.[2] Enabling this kind of architectural innovation would allow us to address many current architectural problems with known solutions, and address future requirements as they arise with newly developed designs. Thus, the ability to incrementally deploy would relieve us from the burden of predicting precisely what these future requirements might be.

In addition, even for requirements we are already aware of, the ability to accommodate innovation allows us to adopt better solutions over time, rather than having to develop a near-perfect solution that we think will last for a generation or more. This need for a design to last for a generation turns the best into the enemy of the good; when we can adopt new solutions as they arise, we need only ask if the new solution is significantly better than our current one, not whether it is almost perfect.

To support architectural innovation through incremental changes, we propose that the core fixed design — the portion that the entire Internet agrees upon and that is not expected to change much over the next generation — not be a comprehensive architecture, but only a minimal architectural *framework*. One can think of frameworks and architectures as being akin to microkernels and kernels, with the former being a simple and minimal design that enables greater flexibility than the latter. To accomplish even the most basic networking tasks, a framework must be augmented with additional architectural components (just as a microkernel needs additional user-space modules to function). For example, a framework might not specify a particular naming or addressing scheme, so these additional design decisions must be made for each individual deployment.

This change in perspective from architecture to framework alters the nature of our intellectual quest. We are not trying to devise a comprehensive set of network components (*i.e.*, particular approaches to routing, addressing, naming, etc.) that provide a checklist of necessary characteristics (*e.g.*, security, reliability, scalability); instead, we are trying to provide a framework within which such comprehensive sets of components: *(i)* can operate as fully functional architectures, satisfying their checklist of properties, and *(ii)* need not be statically and uniformly deployed, but instead can coexist, communicate, and evolve over time. Moreover, we want to maximize the freedom to innovate with such components, while still providing an adequate framework of support. The less the framework specifies, the more freedom there is to innovate, so our goal is to design as little as possible while still achieving our goal. The core intellectual question is then: *which architectural features must be part of this fixed framework, and which architectural components can be allowed to evolve over time and differ over space?*

We propose an answer to this question in the form of a framework we call the *Framework for Internet Innovation* (FII). FII only defines three core interfaces (or primitives): the interface for communicating between domains (*i.e.*, interdomain routing), the interface between applications and the network (*i.e.*, the network API), and an interface hosts can use to protect themselves against denial-of-service attacks.[3] These are the only major design aspects that we expect to remain fixed for a generation or more; we expect all other important design components (such as naming, intradomain addressing, QoS, congestion control, etc.) to evolve as the need arises.

We now briefly expand on these primitives — we describe them more fully in Sections 4-6 — but we first note that there is nothing novel in the design of these basic interfaces; the interdomain interface leverages a recent interdomain routing proposal [21], the denial-of-service interface is based on several previous works [4,27,36], and the network API is merely the obvious way to build an arbitrarily extensible interface. Thus, our contribution lies not in the design of these primitives but in recognizing that these are the only interfaces that must remain fixed.

Interdomain routing: As we argue later, FII requires an interdomain routing solution that: supports a degree of policy flexibility and autonomy at least as great as BGP, is completely independent of intradomain designs (so different domains — *i.e.*, ASes — can adopt different internal architectures without coordination), and provides extensible functionality in areas such as QoS, congestion control, and route computation. We are only aware of one such routing solution (but are open to suggestions for others): pathlet routing [21], which is a scalable policy-compliant source routing design. Pathlets are policy-compliant segments of interdomain paths advertised by domains, and users (or hosts or domains on their behalf) select a set of pathlets to create an end-to-end path. FII specifies the syntax for describing these pathlets, but not the semantics of these pathlets (*i.e.*, FII does not constrain the quality-of-service or congestion control mechanisms used along the path) or how pathlets are disseminated or used in route computations; these aspects of interdomain routing can evolve over time and vary across space.

Network API (netAPI): Rather than embracing one particular set of netAPI semantics, FII supports a diversity of interface *schemas* (each of which defines a set of interface calls and their semantics) and only specifies the way in which the application specifies to the network stack which of these interface schemas the application wishes to use. As we discuss later, these interfaces should refer to other entities by name (rather than address), so naming plays an important role in the netAPI. Here, again, FII does not choose a particular naming scheme but instead allows for multiple namespaces and only requires that each name specify, in a standard way, to which namespace it belongs. This approach allows new network APIs and new naming systems to be introduced without changing existing applications, while enabling new applications to make use of these new features. This system extensibility allows FII to support a wide variety of functionality, such as content-centric networking, disruption- (and delay-) tolerant networking, and mobility by introducing new netAPI schemas and namespaces (along with the necessary support in the stack and domain).

Denial-of-Service: Network security requires the availability, integrity, authenticity, provenance, and confidentiality of network communications. All but availability can be addressed by cryptographic techniques implemented on the end-hosts, so these do not need to be mandated as part of FII. In addition, FII leverages the multipath nature of pathlet routing to improve availability. However, we find that dealing with interdomain denial-of-service requires an additional design element in FII. For this, leveraging earlier work in [4, 36] (see also [27] for a similar approach), we use a simple primitive called a "shut-up-message" (SUM) that allows the victim host to tell an attacking host in another domain to stop sending packets to it.

[2]As we observe more precisely in Section 7, the ability to accommodate diversity enables architectural evolution, because diversity allows evolution to occur without lockstep adoption.

[3]As with any well-defined interfaces, FII's core interfaces are independent of their implementation; thus, while the syntax and semantics of the core interfaces (or primitives) will remain fixed, their implementations can change over time.

The combination of these three major interfaces allows architectural innovations without changing interdomain routing or existing applications, thereby removing the two factors that (as we argue in the next section) impose severe constraints on architectural innovation in the current architecture. In addition, these interfaces enable FII to retain the ability to combat interdomain DoS no matter what other architectural innovations are adopted.

While FII does not remove all restrictions on architectural evolution, it significantly increases the space of designs that could be incrementally deployed. However, as we note later, FII's ability to support innovation should be measured in terms of the *functionality* it allows to be incrementally deployed, not whether a particular design can be incrementally deployed; many designs will, as currently defined, be inconsistent with FII, but their functionality can be incrementally deployed with an alternative design.

In the next section we motivate these design decisions by introducing the notion of "architectural anchors", portions of the architecture that make incremental change difficult. In Section 3 we describe an example of how architectures operate within FII and provide some general clarifications. We then discuss the design of the three core FII primitives in Sections 4-6. In Sections 7 and 8 we evaluate the degree to which FII supports architectural innovation through a detailed analysis and a skeleton implementation of the relevant interfaces[4]. We end in Section 9 with some concluding comments. But before moving on to the rest of the paper, we first state our assumptions about the structure of the future Internet, mention our relationship to related work, and briefly clarify our contribution.

Assumptions: The domain structure of the current Internet did not arise for performance-driven or feature-driven reasons but is instead a manifestation of infrastructure ownership and the need for autonomous administrative control. Because these requirements are driven by human rather than technological factors, we do not expect them to change any time soon. We therefore assume that the Internet will remain organized around autonomous domains, with similar requirements for routing-policy autonomy, and that a global interdomain routing system will continue to serve as the "glue" that ties these domains together. In addition, we assume the existence of an IANA-like body that assigns numbers to various protocols, so they can be referenced in a uniform manner.

Related work: In the body of this paper we cite several sources of inspiration, but we emphasize that our intellectual debt is far broader and deeper than we can itemize. In particular, because architecture is, by nature, a synthetic exercise, our ideas have been influenced by the community's entire literature about and experience with Internet architecture.[5]

Contribution: We should make clear from the outset that we are not proposing any novel algorithms or protocols, and the design principles we articulate are standard lore for distributed systems. To the contrary, the novelty of our proposal is not in the primitives we include, but in what we do not include. Our contribution is simply the observation that *carefully selecting and crafting only a few fixed components allows the rest of the architecture to evolve over time and differ over space*.

[4]By skeleton, we mean that we faithfully implemented the interfaces, but used oversimplified mechanisms behind the interface.

[5]Of particular note are the insightful architectural papers by Clark [13–16, 34] and Crowcroft [17, 37].

2 Motivating the Framework

Enabling architectural innovation is our primary goal in designing FII; however, we must also ensure that FII allows architectures to provide network security. We discuss how these two requirements motivate the three core FII primitives, and then describe some additional (but minor) FII primitives.

2.1 Removing Barriers to Innovation

We start by considering what prevents architectural innovation in the current Internet. Change most naturally occurs when various parties (domains, vendors, users, etc.) can independently modify different parts of the system without large-scale coordination. The Internet architecture enables this kind of independent innovation in applications and network technologies, but not within the architecture itself. To the contrary, many desirable design changes: *(i)* require widespread agreement (among vendors and/or domains), *(ii)* radiate throughout the architecture (*i.e.*, require changes in many protocols), and/or *(iii)* require changing substantial portions of the physical infrastructure. Consider the simple example of what should be a minor modification, increasing address sizes. This requires a global standard for the new addressing format; changes in network stacks, applications, and interdomain routing; and (in many cases) new router hardware. This high degree of coupling between architectural components greatly hinders innovation. To avoid this coupling, *architectural modularity* should be the guiding principle of any Internet redesign.

Modularity is a basic tenet of system design, calling for carefully conceived abstractions implemented by well-designed interfaces. One might think that the layered Internet architecture would be the epitome of modularity. But architectural modularity requires more than layering: it requires that interfaces be both *extensible* and *abstract*. By "extensible" we mean that new functionality can be added to a particular component, and utilized by other components that are aware of this change, without rendering unmodified components obsolete. By "abstract" we mean that the interface deals with the appropriate level of abstraction, avoiding implementation details. For instance, interfaces should not pass network addresses or particular byte layouts, but instead should pass names and structured data.

Making interfaces extensible and abstract is an obvious requirement in modern systems design, but it was not applied to the core networking interfaces. As a result, the current Internet architecture is hard to modify without significant disruption. While desirable everywhere, architectural modularity is absolutely necessary when interfacing with components of the architecture that are inherently hard to change (*i.e.*, that inherently resist innovation). We call these inherently rigid components *architectural anchors*. In order for these anchors to not tie down the components they interact with, the interfaces to these anchors must be extensible and abstract. Thus, enabling innovation requires identifying, and then designing appropriate interfaces for, these architectural anchors. We believe there are two architectural anchors: interdomain routing and applications.

Interdomain Routing: We have explicitly assumed that the Internet will continue to be organized around domains, with interdomain routing serving as a universal "glue" that makes end-to-end connectivity possible. Changing interdomain routing would therefore require global agreement among the domains. This is a very high barrier to innovation, so FII must enable architectural innovation without requiring globally agreed upon changes to

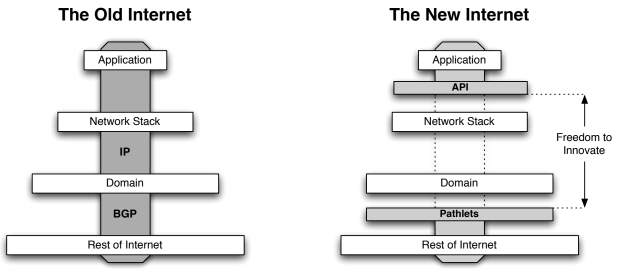

Figure 1: *Consider the set of interfaces from an application, to the network stack, to the domain, to the rest of the Internet. The application-stack interface is the netAPI, and the domain-to-rest-of-Internet interface is interdomain routing. In the current architecture (left), the central protocol, IP, is embedded in both applications and interdomain routing, so the entire architecture is tied to both anchors. In FII (right), the network stack and the domain are shielded from the two anchors by abstract and extensible interfaces, so there is complete freedom to innovate in stacks and domains.*

interdomain routing. As discussed above, this can occur only if interdomain routing serves as an abstract and extensible interface to other domains. Our interdomain routing solution qualifies as abstract because it refers only to domain identifiers and path properties, and is completely independent of domain internals; it qualifies as extensible because (as we describe in Section 4) it supports innovation in route computation, path QoS, and many other aspects of interdomain routing.

Applications: Applications are the most innovative part of the Internet; new and surprising applications are constantly being introduced, arising from a wide variety of sources. Being so large in number and diverse in provenance, it is not reasonable to require that every application be modified (and every end user update her copy of the application to include that modification) each time the architecture is changed. Thus, FII must enable architectural innovation without changing existing applications: this requires an abstract and extensible interface between applications and the network. Our netAPI qualifies as extensible because it allows new interface semantics to be introduced without rendering old ones obsolete, and it qualifies as abstract because these interfaces are defined in terms of names and structured data. See Section 5 for more details.

We have reviewed all the other architectural components, and could not identify any that are extremely difficult to change. In particular, below we argue why three natural candidates are indeed *not* architectural anchors:

- *IP or any other universal packet protocol:* IP is not an anchor because there is no reason why architectures built within the FII framework require a universal packet protocol. Interdomain addressing will be taken care of directly by pathlet routing (as we discuss later), freeing domains to use any peering technology they want (*e.g.*, IP, Ethernet, MPLS, etc.) because they no longer need IP's global addressing to guide packets along interdomain paths.

- *Host network stacks:* While deployed applications may remain unchanged for long periods of time, recent experience with major OS vendors suggest that widely deployed operating systems incorporate new networking features in a timely fashion. Therefore users who choose to avail themselves of new networking functionality will not be impeded (for long) by their host's networking stack.

- *Hardware infrastructure:* Because modern forwarding paths typically involve IP-specific chips, the current hardware infrastructure would be an anchor slowing down architectural change. However, we believe that in the future this can be avoided by using a newly emerging approach called Software-Defined Networks (SDN), which enables forwarding hardware to support a wide variety of architectures, and support multiple of them simultaneously.[6] However, current implementations of SDN do not support arbitrary forwarding behaviors; until the SDN forwarding models are more general, the hardware infrastructure will limit the forwarding behaviors that can be incrementally deployed.

Thus, we believe applications and interdomain routing are the two main architectural anchors, and the two nonsecurity primitives in FII are designed to keep them from weighing down the rest of the architecture. Figure 1 gives a pictorial comparison to today's architecture. In the current architecture, both anchors are tied to IP, since IP is embedded in both applications and BGP. By tying the central communication standard (IP) to both anchors, the current Internet architecture is, in some sense, *maximally rigid*. In contrast, in FII there is almost complete freedom to innovate with network stacks and domain technologies because they are shielded from the two anchors by abstract and extensible interfaces.

2.2 Security

FII's role in security is subtle: it must enable security but need not implement it. That is, FII does not have to, by itself, solve any particular security problem (aside from securing its own mechanisms), but one must be able to use the primitives supplied by FII, and the freedom it allows in other components, to build the mechanisms needed to create a more secure Internet. Here we do not attempt to describe precisely what these mechanisms might be, but rather ask: *what additional primitives need to be included in FII in order to allow the necessary security mechanisms to be built?*

Security is a broad topic, and it is important to define the scope of our task (our discussion is modeled somewhat on [10]). One can succinctly state the goal of network security, narrowly construed, as follows:

> *In the face of attacks, the network should ensure that each participant in a communication can: reach another willing participant (*availability*), determine if they are communicating with the party they intended to communicate with (*identity*), and verify the* provenance *and* authenticity *of any data received.*

Standard cryptographic techniques are sufficient to satisfy the identity, provenance, and authenticity requirements [40, 41]. Thus, the only remaining threat is to availability. Note that decoupling authenticity and provenance from the path (see [41]) enables the

[6]By software-defined networking, we mean the approach exemplified by OpenFlow and NOX. There are other realizations of the SDN paradigm in progress, some of them destined for the commercial market, and there is nothing in what follows that holds specifically for OpenFlow and NOX. The SDN approach does not support arbitrary packet processing, such as in deep packet inspection, but in the short term this could be handled in software (see [18]) until hardware support is available.

widespread use of opportunistic methods of delivery (through caches, alternate paths, etc.), and this in turn makes it easier to protect availability.

There are two forms of attacks against availability: *(i)* compromising individual routers (and other network elements) which can cause failures and byzantine behavior and *(ii)* denial-of-service attacks. We consider each case in turn.

Compromised routers: General reliability mechanisms (multipath, etc.) can protect against router failures, so we focus on byzantine behavior. In pathlet routing (which we review in Section 4), a byzantine domain (*i.e.*, a domain which has a byzantine router speaking on its behalf, or a domain that is being run maliciously) can lie only about its own delivery behavior (claiming that it will deliver packets that it won't), but the cryptographic protections won't allow it to advertise *false* pathlets; that is, it cannot advertise pathlets that include other domains who are not willing to participate in the pathlet. And this form of dataplane attack (claiming to deliver packets it won't) is of limited utility because once users notice that their packets are not being delivered by a domain (either through explicit in-network monitoring, or just end-to-end observation) they can choose routes that avoid that domain. Intradomain routing mechanisms can use similar approaches for byzantine resilience, or rely on in-network monitoring to identify components that are malfunctioning. Domains are free to choose whatever intradomain mechanisms they choose, so these solutions can improve over time. Thus, we see no reason to include a security primitive in FII to deal with this form of attack.

Denial-of-service attacks: Each domain can implement their own mechanisms to cope with intradomain DDoS attacks (such as [44]). However, FII must provide a primitive to deal with interdomain DDoS attacks. There have been many proposals for defending against DDoS that involve detailed specification of the data path (*e.g.*, [43]), and this specification would have to become part of the FII framework. To preserve interdomain datapath flexibility, we advocate a quite different approach that was first discussed in [36] and elaborated upon in [4] (see also [27]). Here, the victim can send a "shut-up message" (SUM) to the attacker telling it to not send any more packets to the victim. We describe the SUM design, and discuss the tension between accountability and anonymity, in Section 6.

2.3 Additional Primitives

There are three other minor primitives that must also be built into FII, mainly to cope with the architectural heterogeneity that FII enables.

Meta-negotiation: When two hosts communicate, they must use compatible higher-level protocols (*e.g.*, transport). With hosts supporting a diverse set of such protocols, they must negotiate which mutually compatible protocol to use (or determine that no such protocol exists). There are various negotiation protocols (see, for example, [19]), and we could have chosen one to include in FII. Instead, however, we choose to standardize a "meta-negotiation" protocol in which hosts decide which negotiation protocol to use. This meta-negotiation protocol can be quite simple (*e.g.*, A sends B the list of negotiation protocols it supports, in some standard format, and B chooses from that list), whereas the negotiation protocols themselves can be quite complex (again, see [19]). We thus believe the wiser choice is to let these negotiation protocols evolve and only standardize the meta-negotiation protocol.

However, we expect that in the common case the supported protocols will be returned in the name lookup (the process that translates a name into a location), allowing the sender to identify mutually supported protocols without any negotiation. Meta-negotiation, and the ensuing negotiation, are merely fallbacks if such hints are not available.

Bootstrap interface: When a host first arrives in a domain, it must learn about the environment and determine where its local resources are (a service typically delivered by DHCP today). FII requires that each intradomain networking technology support a method of bootstrap (such as anycast for a resource directory which contains information about the location of name resolvers and other local resources); the network stack on the host associated with that technology will implement the necessary host actions. FII standardizes a bootstrap interface that can be called by a user-process; the host's stack, having detected the local networking technology (*i.e.*, what kind of network is it plugged into), will execute the appropriate bootstrap actions. The bootstrap interface returns at least the following information: the domain identifier; an intradomain address (IDA) that can be used to reach the host from anywhere within the domain; the name of the default trusted third party (defined in Section 6) and addresses for reaching the first-hop router (or the equivalent in whatever domain technology is being used), a resource directory, and a route computation agent.[7] If the host is multihomed, then it will invoke the bootstrap interface for each network.

Interface query: The netAPI supports queries about which API schemas are supported, so applications can decide which to invoke (backward compatible applications will be able to use both new and old schemas).

3 Example and Clarifications

We will describe the details of the three basic primitives in Sections 4-6, and then describe our "evaluation" of FII in Sections 7 and 8. But before going into these detailed descriptions, we want to first give a more global description of how all the pieces fit together via an end-to-end example and then clarify a few aspects of our work.

3.1 End-to-End Example

To provide more information about how FII would work in practice, we now describe an end-to-end example. But first, we introduce some terminology: a *route computation agent* (RCA) is a virtual entity — existing either at the end host, as a service in a domain (perhaps at one of the domain's routers), or as a third-party service — that provides the host an appropriate end-to-end path to a given destination; an address (or full address) means a domain identifier plus the IDA; a forwarding directive means the set of pathlets constituting a path plus the full address of the destination.

For the example, consider the case of an application on host A in domain X trying to send a file to host B in domain Y, where domain X supports a "next-generation Ethernet" (NGE) and domain Y supports "newfangled IP" (NIP), while the pathlets between them use end-to-end optical. The application calls the netAPI, selecting the appropriate interface schema for file transfer, and passes the file and the name of the host B to the netAPI. The network stack then resolves B's name by first determining the namespace to which the name belongs and then sending the name to the appropriate name

[7]The bootstrap interface specification also includes interfaces to the resource directory and route computation agent. We discuss route computation agents in the next section. Also, when we use the term address here, it could also, in general, be a forwarding directive as defined in the next section.

resolver (whose location it determined from the resource directory). The stack receives a response from the resolver containing B's full address (domain plus IDA). Note that the IDA need not be understood by A, or X, or even B. It only needs to be understood by the domain Y.

If A and B are in the same domain, A's network stack would recognize that fact and would use NGE to reach B using the IDA. However, since we are assuming that B is in a different domain, then A's network stack would retrieve an appropriate set of pathlets from the RCA (whose location was determined by bootstrap, and could be on the host itself, on the first-hop router, or elsewhere). A would construct packets appropriate for the first pathlet (*i.e.*, in the format specified by the pathlet, which we are assuming takes in packets and carries the data via end-to-end optical), encapsulate them in an NGE packet, and then send them to the first-hop router which then forwards them on to the border router. At this point, the border router transfers the payloads into optical signals (with the appropriate framing). Once the optical signal reaches Y, Y's border router transfers the payloads into NIP packets and uses the IDA as the destination address, causing the packets to be delivered to B.

What are the compatibility requirements in this example? The hosts A and B need to use compatible application, session, and transport protocols in order to communicate (also, if a router-based congestion control algorithm is used along the path, the transport protocols must be compatible with that). However, there are *no* other compatibility requirements. The two domains are running completely different internal architectures (NGE and NIP), with different addressing and routing schemes. While we make this argument more explicit in Section 7, this example indicates that domains can adopt different architectures without causing any disruption in service.

This is both similar to today, and very different. Today, domains can adopt different L2 technologies without any coordination, much like in the above example. What is different is that when domains exchange packets with other domains, they must do so using IP because that embodies the only globally understood addressing scheme. Here, pathlet routing provides interdomain communication, so domains need not all implement IP.

3.2 Clarifications

Does FII make architectural innovation easy? No! While FII does not require universal agreement on architectural components, some coordination is still needed to deploy a new architecture (*e.g.*, OS vendors must update their networking stacks to accommodate new technologies); we discuss this further in Section 7. However, we think it fair to say that FII takes architectural innovation from "essentially impossible" to "possible, but nontrivial". Note, however, that deploying FII itself will be quite difficult; once installed, FII will ease innovation, but (as with any clean-slate design) there will be significant barriers to its initial deployment.

Does FII place any limitations on the architectures the Internet can adopt? Yes. FII can only support architectures that: (*i*) use pathlets for interdomain routing, (*ii*) use FII's netAPI syntax (which, as discussed later, places essentially no limitations on the schema semantics), and (*iii*) supports SUM. This would preclude almost all architectures in the literature, since they have not been designed with these constraints in mind. The question, then, is not whether FII supports these particular architectures as currently defined, but whether the functionality these architectures provide can be supported by a FII-based architecture. We think, in most cases, the answer to this is yes: based on our own previous work on designs that support mobility, wireless, data-orientation, end-to-end optical, and disruption/delay-tolerant networking, we believe all of these functionalities and more can easily be implemented within the FII framework.

Why not overlays or GENI-like virtualized testbeds? Overlays allow one architecture to be deployed on top of another architecture. While overlays are an essential technique in networking [5, 6, 38], they do not allow an architecture in wide use to change without disrupting existing applications or other components in the architecture. Similarly, virtualized testbed infrastructures like GENI (and several others) enable many different architectural designs to be deployed simultaneously on a single physical infrastructure, but GENI does not describe how to evolve a currently used architecture without widespread disruption. For example, neither overlays nor GENI allow a domain using the traditional IP architecture to start using, say, AIP [4] without significantly disrupting applications and substantially reworking interdomain routing.[8]

Why not Active Networks? The active network paradigm provides a flexible (and therefore evolvable) data path, but does not solve any other aspect of architectural innovation, such as addressing, naming, APIs, or interdomain routing. A flexible forwarding infrastructure is indeed important for innovation, but here rather than adopt active networks we suggest a far more feasible (but significantly less flexible) approach, Software-Defined Networking, that is rapidly gaining acceptance.

What about other related work? The previous work that is perhaps the closest to what we propose here is Plutarch [17], and the earlier MetaNet whitepaper [42]: these papers bravely articulated the need for architectural heterogeneity in the face of the prevailing monotheist IP model. However, they grappled with a somewhat different problem: allowing hosts, each bound to one or more static architectures ("contexts" in the Plutarch terminology), to communicate with each other. Innovation, within this model, occurs by introducing a completely separate architecture and providing "interstitial functions" to translate between this new architecture and the existing architectures. Moreover, the routing between domains is left undefined, as are defenses against DDoS. In contrast, our approach allows those architectures themselves to evolve, enables hosts to operate within any of them, is built around a sophisticated interdomain routing system, and provides protection against DDoS.[9]

Is this the end of the story? No. We view this as merely a starting point in a much longer research program on architectural evolution. We, and hopefully others, will be looking for superior alternatives for each of the three basic primitives, and at the same time questioning our basic line of reasoning to see if additional, or different, primitives are needed to foster architectural innovation. Further, we need to better understand what kinds of functionality cannot be implemented in a FII-based architecture.

4 Routing

4.1 Overview of Pathlets

As described in Section 2, FII must have an interdomain routing solution that is abstract (independent of domain internals) and

[8]The approach in [32] is essentially an automated way to deploy overlays and, as such, does not address the kind of architectural evolution we are discussing in this paper.

[9]The two approaches are complementary, in that one might use Plutarch's pairwise interstitial functions to "glue" some resource-constrained networking technologies (such as sensornets) to the broader Internet.

extensible (can accommodate future features in areas such as route computation and QoS), while still retaining the BGP's policy flexibility and autonomy. We are aware of only one design satisfying these requirements: pathlet routing [21, 22, 39].[10] In this approach, each domain advertises a set of path segments (called *pathlets*) over which they are willing to carry traffic. Pathlets are specified in terms of the "virtual nodes" (vnodes) they traverse, where for now think of a vnode as synonymous with a domain (and all domains have self-certifying identifiers). All domains involved in a particular pathlet must be willing to support it (which they signify cryptographically), so pathlets are inherently policy compliant and unforgeable.

Pathlets are then broadly disseminated in some fashion. One approach would be to use a gossiping-style algorithm along the physical topology (as described in [21]), which then allows domains to withdraw routes when they fail (*i.e.*, the pathlet withdrawal will reach all sites that received the original pathlet announcement). Another approach would be to have central repositories (hosted by sites such as Google) where pathlets are registered and withdrawn as needed.

Route computation consists of selecting, from all advertised pathlets, a sequence of pairwise compatible pathlets that provide an end-to-end path; this selection can be done using any criteria. To send data, the selected pathlets are either listed in the packet header or, for technologies that require path-setup, communicated on the control plane. As in any source-routing protocol, this approach trivially supports user-controlled multipath routing (*i.e.*, users can always select more than one path and send packets along each). Pathlets scale well (see [21]), can provide intrinsic monitoring, and are secure (in the sense that they are unforgeable). In addition, the approach can be extended to support interdomain anycast and multicast (similar to the approach in [31]).

While each packet-based pathlet can specify its own packet format, in general these packet headers will have the following organization: pathlet information (the series of pathlets to be followed), path-visible information (header information needed for, say, router-based congestion control, along with the accountability field we describe in Section 6), destination IDA, and payload.

4.2 Extensibility

This interdomain routing design is clearly independent of domain internals, so it satisfies the abstraction requirement. We now argue that it is also sufficiently extensible; in addition to supporting fully general route computation, it allows interdomain routing to evolve in several important ways.

First, these pathlet descriptions can be augmented with extensible metadata, allowing domains to advertise novel services (e.g., QoS or middleboxes) or performance information (e.g., bandwidth or loss rates). Second, these pathlets need not interconnect at the IP level; one could have pathlets that interconnect with optical, L2, or any newly developed technology interface. As long as two adjoining pathlets have compatible transmission media at the relevant pathlet endpoints, they can interconnect. Third, FII does not specify how the pathlet information is disseminated. FII must have, initially, some system that ensures that the pathlets can be scalably disseminated (such as gossiping along the physical topology as described in [21]),

[10]We are interested in suggestions for other interdomain routing approaches that satisfy these requirements. Our contribution is not the design of interdomain routing (so we are open to using other interdomain designs within FII), but in noting how an abstract and extensible interdomain routing system would enable Internet evolvability.

Schema ID	Primitives	Data Structure
Sockets	open, connect, accept, read, write, close	Bytes
PubSub	publish, subscribe	Publications
RPC	send_request, receive_request	Function call and response
Multimedia	play	A/V frames

Table 1: Some examples of different interfaces, each with its own set of primitives and data structures.

but new approaches can be developed to augment and/or replace the original method.

Fourth, pathlet routing as defined above supports a wide range of policies (a strict superset of BGP), but even more general policies (such as in [35]) could be implemented using mechanisms like that found in [30] (applied to the control plane, because FII should not specify a particular realization of the data plane), so that a spectrum of policies could be adopted, ranging from the standard customer-provider-peer policies to more idiosyncratic per-user or per-application policies, with widespread dissemination for the former and explicit set-up for the latter. Fifth, the granularity of pathlets is not specified. Transit service is represented in a virtual topology whose nodes are vnodes and whose links are pathlets. A vnode might represent an entire AS, a geographical region, a physical router, or a slice of a router; similarly, a pathlet might represent a physical link, a full end-to-end path, a path augmented with some service (such as virus scanning), or (the most typical case envisioned here) a path across several ASes with the pathlet length just long enough to enforce the relevant domain policies. Sixth, packet formats need not be globally specified; each pathlet can specify its ingress and egress formats (the same applies to signaling interfaces for non-packet technologies).

5 NetAPI

Enabling architectural innovation without changing legacy applications requires that the netAPI (or, rather, the set of netAPIs offered by different operating systems) must be both extensible and abstract. However, current netAPIs are not abstract because (among other reasons) they pass addresses (a network-level concept) to applications, unnecessarily coupling applications to the underlying network architecture. In addition, current netAPIs are not adequately extensible, being tied to Sockets-like semantics and not supporting, for instance, a publication-subscribe network interface.[11] We believe that a fully extensible netAPI should support arbitrary interface semantics by letting applications specify an identifier for an interface schema and then issue calls associated with that interface schema. See Table 1 for a depiction of our approach, where schemas as varied as Sockets, Pub-Sub, and RPC could all be invoked through the same netAPI. Applications would be programmed against one of these schemas, and would invoke it using its identifier. This approach defines almost nothing but the level of indirection necessary to achieve arbitrary extensibility (*i.e.*, by using a schema ID as the initial construct in the netAPI). When new schema were introduced (with new identifiers), operating systems would continue to support

[11]The selection of protocol families in the Sockets API allows extensible selection of protocols, but the basic semantics of the Sockets API are not extensible in typical implementations. However, there is already movement towards more extensible interfaces. We don't view our advocacy of extensibility as novel, we merely note that it is important for architectural innovation.

old schemas. Old applications would continue to use the schema they were designed for, while new or updated applications could use the new schema. For this to work, network stacks would need to be updated to support these new schema, but there need not be uniformity among all hosts before adoption occurs; applications written for Linux can use any new schema in Linux without waiting for Apple or Microsoft to follow suit.

While applications interact with the rest of their world through the netAPI, they refer (within this netAPI) to entities in the world with *names*. As such, the extensibility of the netAPI demands that the names themselves be extensible, so that different naming schemes (namespaces) can be introduced over time. Specifically, applications must not be tied to particular name formats and should instead treat names as opaque, semantic-free bags of bits, relying on the network stack for resolution and other name-based functions. To this end, names should have a well-known syntax that separates the namespace from the name itself: for example, names could be of the form $\langle namespace : name\rangle$, where $namespace$ is an identifier of the namespace. Name resolution (within the stack) would start with a "meta-resolution" step where the network stack identifies the appropriate namespace and then the stack would call the name resolver appropriate for that namespace. This would allow a wide variety of naming designs to flourish [2, 7–9].[12] In passing, we note that name resolution of a host would typically return a full address (and, if the host were multihomed, multiple such full addresses belonging to the same or different domains), along with metadata describing which end-to-end protocols the host supports (which would ease the negotiation process).

Thus, for both naming and the netAPI, FII merely inserts a trivial level of indirection and then allows complete generality in both interface and naming. We note, however, that different operating systems can choose to either expose the netAPI directly to applications (as the Sockets interface is today), or place the netAPI deeper in the stack and hide network actions from the application itself (such as in Plan 9 [29], where the application merely interacts with a file system, and the stack determines if the object requested is local or remote and invokes the netAPI in the latter case). Also, a namespace can be served by several different name resolution protocols. In the resource directory, a name resolver for a namespace will list which name resolution protocols it supports, and the host stack will invoke only those resolvers that support a compatible resolution protocol.

6 Dealing with DoS

6.1 General Approach

FII deals with DoS by allowing the victim to tell an attacking machine to stop sending packets to it via what we call a "shut-up message" (SUM) [4, 27, 36].[13] Similar to capability-based designs (*e.g.*, [28, 43, 44]), this gives the receiver the power to decide what traffic it is willing to receive but, in contrast to capabilities, our approach does not alter the basic default-on nature of the Internet nor does it require cryptographic operations on the datapath. This primitive belongs in FII, rather than allowing each domain to adopt its own DoS defense, because DoS attacks can cross domain boundaries.[14] Domains are still free to use whatever internal mechanisms they want to deal with intradomain DoS, but the inclusion of SUM in FII ensures that there is a way to prevent cross-domain attacks.

SUM is only implementable if we assume that for every host there is a secure networking control-point somewhere in the network that can: *(i)* prevent the source from spoofing (see below) and *(ii)* enforce the SUM message by preventing the source from sending packets to a particular victim. For these simple functions, technologies for secure control-points currently exist: middleboxes in a POP or a hardware NIC could easily provide this functionality. Also, the SUM approach can be used to protect links, domains, or other more general targets of attacks (rather than just hosts) by allowing the SUM to specify more generally which paths are not allowed (*i.e.*, any that pass through a particular domain) rather than merely a destination.

Note that before our security discussion, there was no need for FII to require that signaling mechanisms (packet headers for packet-based transport, signaling protocols for circuit-based transport) carry any information about the sender. However, FII needs a mechanism that enables the SUM to reach the sender. The natural way to do this would be to include the source address in the packet header (which the control-point could ensure was valid), but, for privacy reasons, we do not want to reveal the identity of the sender to anyone who can see the packet header.

There already exist approaches to achieving accountability while preserving privacy in the literature that we could adopt in FII [3, 11]. However, since FII is a general framework, we do not mandate a mechanism, but instead specify an interface that the accountability mechanism must support. Our interface uses two fields in a packet header (or similar information in a signaling protocol) that describes (*i*) who to contact (an entity we will call a trusted third party, TTP) to shut up the source of that packet, and (*ii*) an accountability field that allows the TTP to identify the source of the packet but hides the identity from everyone else. Revealing only the TTP, not the individual source, in the packet header provides some degree of privacy. Of course, the TTP does know the identity, but they have been contracted (by the sender) to keep that information private and only forward SUMs as needed. There need not be any globally trusted third party, but all trusted third parties must be globally reachable and assigned an identifier. We assume domains provide hosts with a default TTP, although hosts may choose to use another TTP (which must be approved by the domain).

The challenge is in designing the accountability field that provides the TTP with the appropriate information while not imposing significant per-packet computational costs along the path. Below, we sketch the design of one such protocol, but many others are presumably possible.

[12]While DNS provides separate TLDs, with independent naming control, they are all resolved using the same infrastructure. Here, namespaces can use completely different name resolution mechanisms.

[13]See [4, 27] for a description of how this can be done safely and scalably; with the appropriate cryptographic techniques, one can ensure that sources can identify valid SUMs, and SUMs can be used to deal with large-scale attacks. In addition, the SUM semantics of "stop sending packets to me" represents a statement of receiver rights, not an indictment of the sending machine (in stark contrast to taking a machine off the net entirely), so we find it attractive from a policy viewpoint.

[14]There are other architectural aspects that cross domain boundaries, such as congestion control, but these all involve cooperating hosts that can negotiate mutually supported protocols. With DoS, we are dealing with malicious hosts that will use any such flexibility to preserve their ability to attack. Thus, FII must include SUM as a required primitive.

6.2 One Possible Accountability Protocol

The entities involved include a source host S, a destination host D, the source host's first-hop router R_S, and the trusted third party G. Via a pair of key exchanges S, R_S, and G establish a shared symmetric key k_{srg} (and G learns the address of R_S) and R_S and G establish a shared symmetric key k_{rg}.[15] G also assigns R_S a set of identifiers $ID_{r1}, \ldots, ID_{rn}$ (each a bag of bits) that it can use to identify R_S; these IDs may be generated cryptographically as well to avoid state. S and R_S should use long-term well known state about G (like G's public key) to ensure the key exchanges are authenticated, thereby helping to prevent a MITM attack.

When S sends a packet with payload P via R_S, it must include a tag $T = \text{MAC}_{k_{srg}}(P||\text{epoch}\,\#)$ where MAC is a message authentication code such as HMAC [25] or PMAC [12]. The epoch # is incremented on a timescale specified by G (and may vary depending on S, to allow for delay-tolerant networks, etc.). The purpose of T is to prove that S indeed vouches for this packet.

When R_S forwards the packet along, it must generate an encrypted source address value $SAD = E_{k_{rg},T}(\text{``}S\text{''})$. Here E is a tweakable block cipher [26] keyed on k_{rg} with a tweak T (the tag of the packet); "S" is some domain-specific identifier that can be used by R_S to identify S, such as its IDA. "S" must also be known to G. The router includes in the packet T, SAD, G's identifier, the epoch, and ID_{ri} (chosen at random from $ID_{r1}, \ldots, ID_{rn}$). If we didn't tweak the block cipher call on T, all packets from S would be stamped with the same SAD, linking them; if T didn't include an epoch #, the same packet sent during two different time periods would have the same SAD. By choosing the ID at random, the router helps prevent intermediaries from determining that two packets originated at the same router.

If D wants to shut up S, it sends a request, along with a packet from S, to G. G can immediately determine the responsible router/domain and source host — and can verify the accountability field of the request from D itself by asking D's TTP — and request a shut-up. How the shut-up is handled is an administrative issue. A more detailed discussion of the benefits and tradeoffs of this protocol is forthcoming.

7 Evaluation: Analysis

Any meaningful evaluation of FII must assess its ability to support architectural evolution and diversity. One cannot judge this using typical performance metrics, so instead we seek more qualitative methods. We evaluate FII in two steps: in this section we use information flow diagrams to determine the changes required when adopting new architectures; in the next section we describe a skeleton implementation of the architectural interfaces that verifies that FII enables packets to flow between different architectures. But first we note that FII's ability to support evolution flows almost directly from its ability to support diversity; that is, one domain can change its internal architecture without coordinating with another domain precisely because in FII each domain is oblivious to the architecture another domain is using. Thus, in what follows we focus on diversity, because that leads to evolvability.

7.1 Information Flows

An architecture involves many interacting components, and without understanding the interfaces between these components one cannot assess the impact of architectural diversity. One technique for analyzing these interfaces is to trace how information flows through the architecture (*e.g.*, how does the client get the destination address, how does a host know its own address); this will reveal where architectural incompatibilities might arise when domain architectures differ.

[15]There exist numerous authenticated key exchange protocols that these parties could engage in; we do not tie ourselves to a specific one.

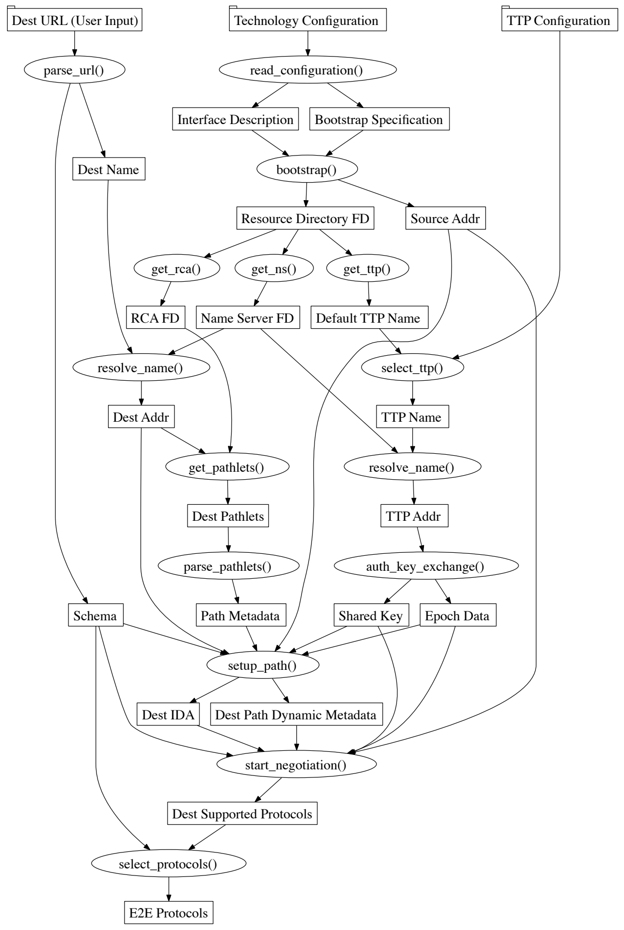

Figure 2: Information flow in FII for a client host. Rectangles represent pieces of information and ovals represent functions that combine information to yield new information.

Figure 2 shows a portion of the information flow in FII, focusing mainly on how a client interacts with the network from bootstrap until a URL is fetched. Each box depicts a piece of information, and each oval depicts a function. The initial information provided through configuration or user input is at the top of the diagram. For illustration, we describe a small portion of the diagram starting at Technology Configuration, which depicts the process of bootstrapping. The technology configuration (*i.e.*, what kind of network the host is attached to) is parsed by a function that extracts an interface description (*e.g.*, a MAC address from a NIC) and a bootstrap primitive (*e.g.*, a DHCP broadcast address) associated with the network technology. These are used by a bootstrap function (in the stack) that gets a forwarding directive for a resource directory. The forwarding directive is used by three different functions that produce information about the RCA, the default TTP, and various name resolvers. This information is used by the left portion of the diagram to perform the actual data request.

This is only a small portion of the diagram, and in turn this diagram is merely a small part of the much larger flow of information, but we hope the preceding illustrates the nature of these diagrams. We pored over this and similar information flow diagrams, determining where incompatibilities arise when architectures differ, and then used this knowledge to determine what changes would be needed to implement architectural changes within FII. Below we briefly describe two such changes.

First, consider a domain that changes its technology from IP to AIP. This would change the technology configuration, which in turn would change the interface description (*e.g.*, a new IDA) and bootstrap specification (*e.g.*, AIP may have a different bootstrap procedure). However, once bootstrap has been called and the host's address and the resource directory forwarding directive are obtained, the existence of this new technology is hidden from the rest of the information flow (except for the value of the IDA, which is just seen as an opaque bag-of-bits by the stack).[16]

Second, consider the case where the destination host upgrades its OS and this version of the OS supports a new transport protocol. This would become visible to the source when (toward the bottom of the graph) the negotiation process reveals which protocols the destination supports. As long as the destination host still supports the old transport protocol, the negotiation process will either choose the new protocol (if supported by both) or the old one.

Our point, here, is that neither of these changes penetrated deeply into the information flow; the change in source domain technology merely changed the bootstrap procedure, and the destination's change in transport protocol merely influenced the actions of the negotiation protocol (but did not require any change in that protocol).

Contrast this with today's architecture, where a domain can't change from IP to AIP and still communicate with other domains (both because they can't exchange packets and because interdomain routing wouldn't supply paths to this domain). However, today a domain can change its L2 technology without coordination with other domains. The difference is that L2 protocols never provided internetworking, and thus local L2 changes were hidden from other domains. In FII, internetworking is provided by the domain-based interdomain routing solution, so all FII-based intradomain solutions are more like today's L2 than today's L3.

7.2 Various Architectural Changes

For completeness we now walk through a much larger list of possible architectural changes and briefly describe, for each such category, what modifications are required.[17] Note that deploying a clean-slate architecture would typically require more than one category of change (*e.g.*, adopting DONA [24] encompasses, at minimum, changes to name resolution, namespace, path properties, end-to-end protocols, and domain technology).

Naming Changes: Introducing a new name resolution protocol requires (*i*) modifying existing or supplying new name resolvers to support the new protocol, (*ii*) adding OS stack support for the protocol, and (*iii*) changing entries in the resource directories to indicate support for the protocol. Domains and OS vendors can take these actions independently, and hosts can take advantage of this new protocol when both the OS and local name resolvers have been modified.

[16]Note that in the case where the destination's domain changes its technology from IP to AIP, the change in the destination IDA (obtained from the name resolver, and seen by the source as a bag-of-bits) is the *only* sign of an architectural change!

[17]We omit, however, changes to the structure of interdomain routing, as this would involve changing FII itself.

Introducing a new namespace typically requires a new name resolution protocol, so all the changes required above pertain here as well. Content providers must alias objects with names in the new namespace. This latter step is presumably the biggest barrier to adopting a new namespace.

Domain and Path Changes: For a domain to change its internal architecture, it must change its networks (which may or may not require new hardware), and the IDA entries (in the naming resolution system) for hosts within the domain must be changed. If this technology is new, the OS stacks must be upgraded to support it.

Any set of domains supporting a particular pathlet can introduce a new path property (such as a novel QoS). At the very least, this requires the participating domains to take whatever measures are required to deliver this new property (which might require new router algorithms, or new network management algorithms). For the other actors involved, there are two cases. If the property is merely descriptive (*i.e.*, it tells the RCA information that may inform its decision about whether to use this pathlet, but otherwise the communication over the pathlet is unchanged), then the only modifications are that the RCA needs to understand the metadata associated with this new property. If the property requires host involvement (*i.e.*, a new congestion control algorithm, or a new packet header format), the OS stacks must be upgraded to include support for these functions.

Some pathlets might require a host to "set up" the flow before sending data (as in ATM). A change in such a setup protocol requires changes in the OS stacks and in the domain infrastructure to support it.

Any number of domains, or even other entities, can decide to distribute their pathlets (or the pathlets of others) in any way they choose. This requires the cooperation of the participating domains and modifying RCAs so they can obtain pathlet information through these new mechanisms.

End-to-end related changes: Introducing a new negotiation protocol requires OS stack support. However, this support need not be immediate, since the meta-negotiation protocol will allow hosts to discover when they both support a new negotiation protocol; otherwise, they can use an old one.

Similar to above, adding an end-to-end protocol (such as a new transport protocol) requires OS stack support. However, this support need not be immediate since the negotiation protocol will allow hosts to discover when they both support the new protocol; otherwise, they can use an old one.

Changes in network-based congestion control (and other changes requiring both host and path involvement) require both OS stacks and path elements to support the new feature. However, note that two hosts can use such a feature as long as there is *some* path between them that supports it (*i.e.*, they can choose an appropriate path, rather than waiting for all providers to adopt this change). In some cases the originating and destination domains must also support this feature.

For a TTP to change the protocol with which it exchanges keying and other accountability information with hosts and domains, the host stack and first-hop router must be upgraded to support the changes (if this support is not already present).

The message here is that in none of the above categories did the changes require global agreement, or lockstep adoption. Typically new designs could be introduced alongside old ones, and then the old ones slowly be phased out; thus, as mentioned earlier, the key

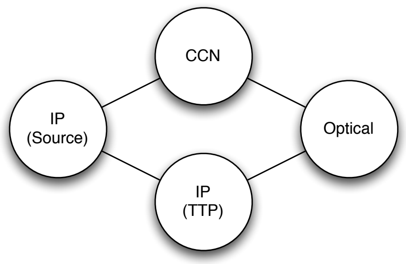

Figure 3: The diversity experiment domain topology. We arbitrarily placed the TTP in the second IP domain.

to FII's ability to evolve is precisely its ability to accommodate diversity and allow both the new and old designs to coexist. In summary, this analysis suggests that changing architectures in FII, while not trivial, is significantly easier than it is today.

8 Evaluation: Skeleton Implementation

Here we report on a skeleton implementation of FII's interfaces and on a few architectures that we implemented within FII (traditional IP, CCN [23], and an all-optical design).[18] Our implementation focuses on the core FII interfaces, the interfaces needed to distribute information through the architecture, and the main features of implemented architectures. However, we do not implement the detailed mechanisms *behind* these interfaces. For example, we don't implement recursive DNS queries, just the name resolution request and reply between a host and a server. One of the architectures we implemented within FII to test its support for diversity was CCN, and for this we implemented interfaces for the main features of name registration and name-based routing with oversimplified mechanisms behind the interfaces. Since the purpose of our implementation is to evaluate FII's ability to support innovation, not to evaluate the performance or scalability of any particular architecture, we feel this degree of implementation is sufficient.

Implementation Details: The information flow in our working implementation is very similar to that depicted in Figure 2 (and in the other information flow diagrams) thus providing some validation of the diagrams used in our previous architectural analysis. To capture information flow more explicitly, we implemented FII entities (hosts, routers, RCAs, etc.) in separate processes communicating via protocols defined using Google protobufs [1]. We chose to use protobufs for all of our messages to focus on the content and structure of headers rather than their byte-level formatting. Communication between processes happens over a topology whose links are implemented as TCP connections.

For simplicity, and because intradomain routing is hidden from FII, we connect all entities in a domain to a single router, which also has a number of interdomain (peering) links to routers in other domains. In all of our experiments, we create a single trusted third party. We use Diffie-Hellman key exchange between a host, its router, and the trusted third party to establish the keys needed; we use AES as the block cipher and HMAC as the MAC algorithm. Finally, our implementation is composed of 2k+ non-whitespace lines of C++ and Python and uses standard networking, asynchronous I/O, graph theory, and cryptography libraries.

[18]Because none of these architectures were designed with FII in mind, we necessarily altered their designs to fit them into FII.

Basic experiment: We begin with a basic experiment — an end-to-end ping — between two IP domains. Each domain has the following entities: a host, a bootstrap server/resource directory, a router, a name server, and an RCA. There is also a single trusted third party located in the first domain.

Host1 sends a ping to Host2 by performing the following steps: (1) an ARP-like message exchange with the router, (2) a DHCP-like bootstrap to discover the RCA and the name server from the bootstrap server, (3) a key exchange with the first-hop router and the TTP, (4) name resolution to learn the domain and the intradomain address (IDA) of Host2, (5) obtain a path to Host2 from the RCA, (6) send a ping packet to Host2 with the appropriate accountability tag. Host2 performs a similar sequence of steps to reply.

We can also look at this basic experiment from the perspective of a packet; this helps to explain how entities along the path interact with the abstractions used in FII. After Host1 sends a packet, its first-hop router examines the accountability tag, adds accountability information such as the encrypted IDA of Host1, and forwards the packet using its intradomain routing system to the first vnode (the entryway to the first pathlet). When the first vnode receives the packet, it forwards the packet as described in [21] until it reaches the next vnode. This process continues until all the pathlets have been traversed, at which point the packet will be in its destination's domain. The router on which the last vnode resides uses its intradomain routing system to forward the packet using Host2's IDA. Note that transit domains never look at the destination IDA and thus are oblivious to its architecture. Similarly, the source and destination domains don't know about the architectures of transit domains, all they care about is that the pathlet delivers the packet (and that the incoming and outgoing interfaces of the first and last pathlets are compatible with their respective domains, which is one of the criteria for path selection).

Advanced features: To better understand some of the key features that enable diversity in FII, we implemented mechanisms such as (recursive) path setup. Some domains may require that all communication over certain portions of their networks first obtain a capability or include some other special headers. Others may require dynamic path setup to establish path state before communication can proceed.

In our implementation, to perform path setup, a host reads pathlet metadata indicating the entities — middleboxes or routers — with whom they must communicate regarding path state. By sending a special path setup request to these parties, the host causes the path state to be set up and receives the appropriate header and IDA information to place in its end-to-end packets. The abstract nature of FII's path setup interface allows for a host to set up paths even using remote technologies it has no knowledge of, thereby enabling technology diversity across space, which we turn to next.

Diversity: To validate FII's ability to support diversity, we set up an experiment with four domains, two using IP, one using CCN, and another using a hypothetical all-optical architecture (as shown in Figure 3). We sent data transfer requests from the source host (in the first IP domain) to hosts in each of the three other domains. Despite the disparate architectures these domains use, interdomain communication proceeds unimpeded; Figure 4 depicts summaries of packet traces from all four domains, stretching over the five stages of progress: bootstrap, key exchange, request to Optical, request to IP, request to CCN.

In the first phase, all entities perform ordinary bootstrap opera-

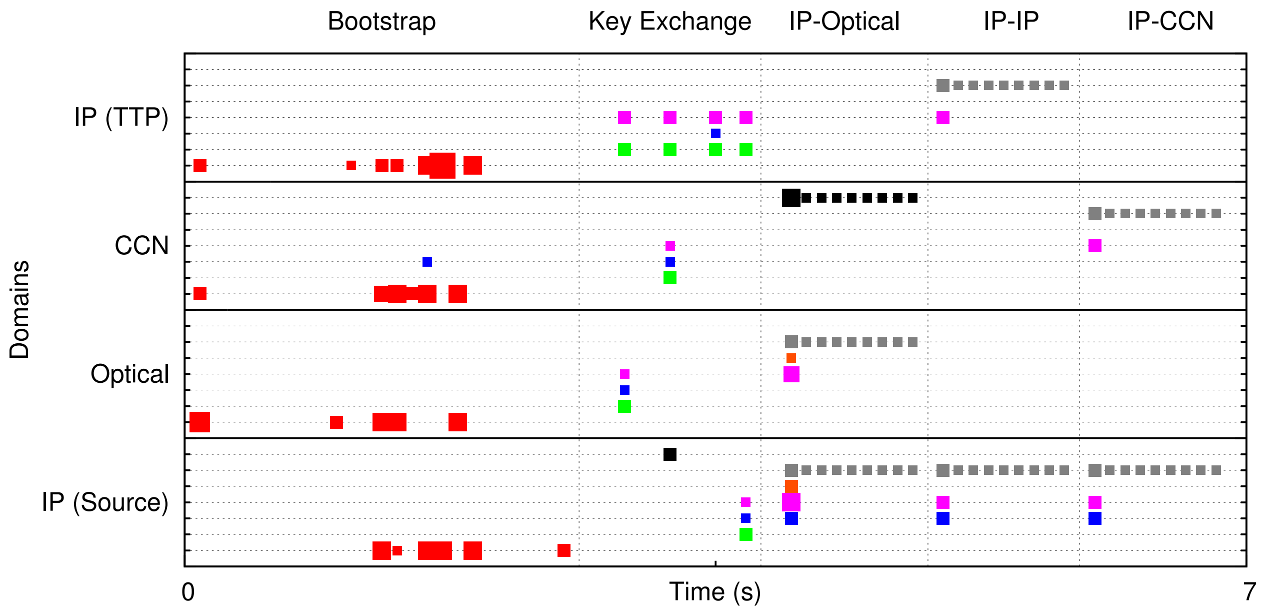

Figure 4: Packet traces for the four domain experiment. For each of the domains (listed along the y-axis), we categorize packets into seven types, each of which is a row (from bottom to top) for each domain: 1) bootstrap, 2) key exchange, 3) naming, 4) pathlet data, 5) path setup, 6) end-to-end data transfer, and 7) interdomain transit. A dot appears if packet(s) were observed of that type, in that domain, at that time. Dot size depicts packet count. We note the communication phases above the graph.

tions, except CCN, which also registers a name. In the key exchange phase, all end hosts and their routers perform key exchange with the TTP. Since the CCN domain and the TTP's domain are not directly connected, they use the other IP domain as transit in this phase. The next three phases involve data retrieval. The first of these, from the source host to a host in the all-optical domain, requires path setup and subsequently more pathlet requests than other data retrieval requests. Figure 4 shows that a CCN domain can serve as transit for communication between IP and Optical domains, and that IP and Optical can even talk, neither of which would be possible without the right abstraction.

All of the many details above are irrelevant, except to provide evidence that (a) we built a reasonably faithful skeleton implementation of FII and three resident architectures, and (b) we were able to exchange packets between these architectures without any special measures (*i.e.*, the interoperability was intrinsic in FII). This ability to accommodate diversity is the bottom line; everything else is just commentary.

9 Conclusions

In the past decade the research community has devoted significant effort to "clean-slate" redesigns of the Internet architecture. The typical goal in such designs is to improve Internet functionality along a number of dimensions (security, reliability, data-orientation, etc.), and the resulting literature has taught us much about how to build a better Internet. However, since the deployment of such clean slate designs is so difficult, the resulting designs must not only meet current needs (which is hard enough) but also anticipate future needs (which is much harder, and impossible to know if we have gotten it right).

To avoid this need for anticipating the unknowable, in this paper we propose a different goal for clean slate design efforts: building a more evolvable Internet, one that supports architectural innovation. We described a microkernel-like approach to Internet architecture, where fixing a minimal design allows the rest of the architecture to evolve much more easily. We do not claim that our particular design, FII, is right in all its details; in fact, we would be shocked if subsequent discussions with the community left this design unchanged. Instead, we merely offer it here to initiate a broader discussion on how one achieves this goal of architectural innovation.

Does our emphasis on evolvability imply that the previous emphasis on functionality was misguided? Not at all. We believe that going forward, the community's clean-slate design efforts should have a dual focus: we need to understand how best to support architectural innovation with a minimal framework (such as we have proposed here), and we also need to understand how to best support various functionalities with architectures that fit within such frameworks. Both research agendas are essential for the future of the Internet.

10 References

[1] Google protocol buffers. `http://code.google.com/p/protobuf/`.

[2] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proc. SOSP*, 1999.

[3] M. Afanasyev, T. Kohno, J. Ma, N. Murphy, S. Savage, A. C. Snoeren, and G. M. Voelker. Privacy-preserving network forensics. *Communications of the ACM*, June 2011.

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

[5] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. SOSP*, 2001.

[6] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. N. Rao. Improving web availability for clients with MONET. In *Proc. NSDI*, 2005.

[7] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the Internet. In *Proc. SIGCOMM*, 2004.

[8] H. Balakrishnan, S. Shenker, and M. Walfish. Semantic-free referencing in linked distributed systems. In *Proc. IPTPS*, 2003.

[9] M. Balazinska, H. Balakrishnan, and D. R. Karger. INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Proc. Pervasive*, 2002.

[10] S. M. Bellovin, D. D. Clark, A. Perrig, and D. Song (Eds). Report of NSF workshop on a clean-slate design for the next-generation secure internet. *GENI Design Document 05-05*, 2005.

[11] A. Bender, N. Spring, D. Levin, and B. Bhattacharjee. Accountability as a service. In *Proc. SRUTI*, 2007.

[12] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *Proc. EUROCRYPT*, 2002.

[13] M. S. Blumenthal and D. D. Clark. Rethinking the Design of the Internet: The End-to-End Arguments vs. the Brave New World. *Transactions on Internet Technology*, 2001.

[14] D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proc. SIGCOMM*, 1998.

[15] D. Clark. Toward the design of a Future Internet. *Manuscript*, October 2009.

[16] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's Internet. *IEEE/ACM Transactions on Networking*, June 2005.

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

[18] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proc. SOSP*, 2009.

[19] B. Ford and J. Iyengar. Efficient Cross-Layer Negotiation. In *Proc. HotNets*, 2009.

[20] M. J. Freedman, M. Arye, P. Gopalan, S. Y. Ko, E. Nordstrom, J. Rexford, and D. Shue. Service-centric networking with SCAFFOLD. Technical Report TR-885-10, Princeton Computer Science Department, September 2010.

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

[22] P. B. Godfrey, S. Shenker, and I. Stoica. Pathlet Routing. In *Proc. HotNets*, 2008.

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

[24] 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. SIGCOMM*, 2007.

[25] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104, IETF, February 1997.

[26] M. Liskov, R. L. Rivest, and D. Wagner. Tweakable block ciphers. In *Proc. CRYPTO*, 2002.

[27] X. Liu, X. Yang, and Y. Lu. To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets. In *Proc. SIGCOMM*, 2008.

[28] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. In *Proc. SIGCOMM*, 2007.

[29] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from bell labs. *Computing systems*, 8(3):221–254, 1995.

[30] L. Popa, I. Stoica, and S. Ratnasamy. Rule-based forwarding (RBF): Improving the Internet's flexibility and security. In *Proc. HotNets*, 2009.

[31] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP multicast. In *Proc. SIGCOMM*, 2006.

[32] S. Ratnasamy, S. Shenker, and S. McCanne. Towards an evolvable internet architecture. In *Proc. SIGCOMM*, 2005.

[33] J. Rexford and C. Dovrolis. Future internet architecture: clean-slate versus evolutionary research. *Communications of the ACM*, September 2010.

[34] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[35] A. Seehra, J. Naous, M. Walfish, D. Mazieres, A. Nicolosi, and S. Shenker. A Policy Framework for the Future Internet. In *Proc. HotNets*, 2009.

[36] M. Shaw. Leveraging Good Intentions to Reduce Unwanted Network Traffic. In *Proc. SRUTI*, 2006.

[37] J. Su, J. Scott, P. Hui, E. Upton, M. H. Lim, C. Diot, J. Crowcroft, A. Goel, and E. de Lara. Haggle: Clean-slate Networking for Mobile Devices. Technical Report UCAM-CL-TR-680, University of Cambridge, Computer Laboratory, January 2007.

[38] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz. OverQoS: An overlay based architecture for enhancing Internet QoS. In *Proc. NSDI*, 2004.

[39] V. Valancius, N. Feamster, R. Johari, and V. V. Vazirani. MINT: A Market for INternet Transit. In *Proc. CoNEXT*, 2008.

[40] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *Proc. NSDI*, 2004.

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

[42] J. Wroclawski. The metanet: White paper. In *Workshop on Research Directions for the Next Generation Internet*, May 1997.

[43] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *Proc. of IEEE Symposium on Security and Privacy*, 2004.

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