XIA: An Architecture for an Evolvable and Trustworthy Internet

Ashok Anand[†] Fahad Dogar Dongsu Han Boyan Li Hyeontaek Lim
Michel Machado[*] Wenfei Wu[†] Aditya Akella[†] David G. Andersen John W. Byers[*]
Srinivasan Seshan Peter Steenkiste

ABSTRACT

Motivated by limitations in today's host-based IP network architecture, recent studies have proposed clean-slate network architectures centered around alternative first-class principals, such as content, services, or users. However, much like the host-centric IP design, elevating one principal type above others hinders communication between other principals and inhibits the network's capability to evolve. Our work presents the eXpressive Internet Architecture (XIA), an architecture with native support for multiple principals and the ability to evolve its functionality to accommodate new, as yet unforeseen, principals over time. XIA also provides intrinsic security: communicating entities validate that their underlying intent was satisfied correctly without relying on external databases or configuration.

In this paper, we focus on core architectural issues in the XIA data plane. We outline key design requirements relating to native support for multiple principals and intrinsic security. We then use case studies to demonstrate how the XIA design facilitates evolvability and flexibility.

Categories and Subject Descriptors

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

General Terms

Design

Keywords

Internet Architecture, Evolution, Multiple Communication Styles, Intrinsic Security

Carnegie Mellon University, Pittsburgh, PA, USA.
[*]Boston University, Boston, MA, USA.
[†]University of Wisconsin-Madison, Madison, WI, USA.

Hotnets '11, November 14–15, 2011, Cambridge, MA, USA.

1. INTRODUCTION

The "narrow waist" design of the Internet has been tremendously successful, helping to create a flourishing ecosystem of applications and protocols above the waist, and supporting diverse media, physical layers, and access technologies below. However, the Internet, almost by design, does not facilitate a clean, incremental path for the adoption of new capabilities. This shortcoming is clearly illustrated by the 15+ year deployment history of IPv6 and the difficulty of deploying primitives needed to secure the Internet.

We argue that these problems are fundamentally tied to the difficulty of changing the "contract" between hosts and routers: it is nearly impossible to modify in any deep way the information shared between end-points and routers that informs routers how to act upon packets. We make the case for a new Internet architecture, called the eXpressive Internet Architecture or XIA, to address these problems from the ground up. XIA maintains several key features of the current Internet such as a narrow waist that networks must support and default-on communication, but it differs from today's Internet in three key areas.

First, while the Internet architecture and protocols are optimized for host-to-host communication, XIA supports multiple communication styles. XIA is centered on the abstraction of a *principal*: a named originator or recipient of a packet, such as a host, a service, or a piece of content. Each *type* of principal (e.g., host, service or content) is associated with a different contract with the network and, therefore, enables a different communication style. This provides *expressiveness* in the sense that applications can more precisely specify their intent by choosing the appropriate principal types for communication. It also allows the network to more aggressively optimize communication operations for a particular communication style through, e.g., caching and replication.

Second, whereas the current Internet's narrow waist is fixed, the set of principal types supported in XIA can be extended over time. This provides *evolvability* in two ways. First, new application paradigms and usage models can be supported more effectively by adding native support for the appropriate principals. However, ubiquitous support for a new principal type is unlikely to occur overnight. As a result, the XIA architecture provides support for incremental deployment of new principal types, allowing applications to benefit from

even partial deployment of router support for the principal. Second, advances in computing and storage technology can be leveraged to enhance network support for a given principal (e.g., allow incremental deployment of principal-specific optimizations) in a clean, transparent fashion.

Taken together, these two aspects enable XIA to provide many, if not all, of the goals of alternative architectures such as, content-centric [12, 25] networks that better support various forms of content retrieval, and service-centric [10, 21] networks that provide powerful primitives such as service-level anycast. The XIA architecture enables each of these important communication styles—and those that will emerge in the future—to be supported natively to the degree that it makes sense to do so at a given point in time.

Finally, XIA guarantees principal-specific *intrinsic security* properties with each communication. This allows an entity to validate that it is communicating with the correct counterpart without needing access to external databases, information, or configuration. Intrinsic security is central to reliable sharing of information between hosts and routers and to ensuring correct fulfillment of the contract between them. Also, it can be used to bootstrap higher level security mechanisms.

In the rest of the paper, we present the design of the XIA architecture, which addresses the above three requirements: expressiveness, evolvability, and intrinsic security. In addition to the details of our key design features, we also present case studies of interesting usage scenarios enabled by XIA.

2. XIA DESIGN

Below, we outline the three key pillars that XIA uses to meet the requirements outlined in the previous section. We then use a simple Web browsing example to illustrate how these design pillars play out in practice.

2.1 Three Pillars of XIA

1. Principals. In contrast with an IP-based Internet, XIA enables a richer "contract" between applications and the network through the use of *principal types*. Applications can use different principal types to directly express their intent to use specific functionality to the network. Each principal type defines its own contract, instructing routers to process the packet in a principal type-specific way, e.g., by employing appropriate per-hop behavior such as unicast for host-to-host communication and anycast for services. It also gives the network significant flexibility for in-network optimizations to satisfy that intent by allowing network elements to observe and act upon it directly.

XIA supports an open-ended set of principal types, from the familiar ("hosts"), to those popular in current research ("content" or "services"), to those that we have yet to understand; we illustrate some possibilities in §3. New applications or protocols may choose to define a new principal type at will, and use this new principal type for the destination or source of their packets at any time, *even before the network has been modified to natively support the new function*. This allows incremental deployment of native network support without further change to the network endpoints, as we will explore through examples in §5.

2. Fallback. A key goal in XIA is to make it possible to seamlessly introduce new functionality while avoiding the "bootstrapping problem". Experience has shown that deploying new functionality in a way that requires applications to use new data formats, has host stacks handle it appropriately, and asks all network routers to act upon this new information to forward packets, *all at the same time*, is untenable. In XIA, we require that the architecture have a clean, built-in mechanism for enabling new functions to be deployed in a piecewise fashion, e.g., starting from the applications and hosts, then, if popular enough, providing gradual network support. The key challenge is how a legacy router in the middle of the network should handle a new principal type that it does not recognize. To address this, we introduce the important architectural notion of a *fallback*. Fallbacks allow communicating parties to specify alternative action(s) if routers cannot operate upon the primary intent. We provide details in §4.

3. Intrinsically secure identifiers. One of the main reasons why IP is notoriously hard to secure is that security was not a first-order design consideration. A key goal in XIA is to build security into the core architecture as much as possible, without impacting expressiveness. In particular, we require that the semantics of a communication's security correspond to the specific functionalities being exercised.

To meet these requirements, we require source and destination identifiers in XIA to be *intrinsically secure*, i.e., cryptographically derived from the associated communicating entities in a principal type-specific fashion. This allows communicating entities to more accurately ascertain the security and integrity of their transfers. For example, the key requirement in host-based communication is to authenticate the respective hosts, whereas in content retrieval it is to ensure integrity and validity of the data obtained. As result, host identifiers in XIA are a hash of host's public key, as in AIP [2], while content identifiers are a hash of the content. This also helps ensure fulfillment of the contract between end-points and routers; for example, routers can attest that they delivered specific bytes to the intended recipients. We do not require the architecture to specify how these identifiers are obtained, but it should support many options for doing so (e.g., root of trust, social relations, etc.), and these options should have well-defined semantics and interoperable interfaces.

2.2 Illustrative Example

We highlight the role of the above design pillars by showing how a simple webpage retrieval scenario can be supported in XIA. Note that this application involves communication with multiple principal types. The process of fetching a web page begins as a user interaction with a *service*, such as the Web server for `http://site.com`. This service identifies or creates the content that should be provided to the user, and

the browser must then obtain the *content*. We provide the detailed steps of the web browsing interaction below:

(1) The browser sends a request to the `site.com` service by sending a packet with the (intrinsically secure) identifier of `site.com` as the destination. One advantage of using a service identifier, rather than a host identifier, is that the network may be able to locate a nearby server if the service is replicated on multiple hosts, in the spirit of anycast.

(2) The `http://site.com` service returns a response addressed to the requesting host's secure host ID with a list of intrinsically secure content IDs for the chunks of data that make up the web page. Responses are signed, allowing the browser to verify that the response was generated by the intended service using the service's public key.

(3) The browser retrieves the content using a sequence of request packets that use the content identifiers as the destination, expressing the primary intent of the application (namely, to reach any source of the identified content). The network could route the requests to the nearest source of valid content, and upon retrieval, the browser can verify the correctness of the content by verifying that the hash of the content matches the secure content identifier.

(4) Of course, not all content will be cacheable, and not all routers are expected to support the content principal type, or to maintain routes to arbitrary content IDs. To account for this, each content request also includes a "fallback" option. In the case of our web retrieval example, the fallback option is the service identifier of `site.com`, which acts as the origin server for the requested content. This is an example of how XIA supports evolution: an application can still get some benefits even if not all routers and hosts support the new principal type. In a similar fashion, service access requests also include a fallback, for example to a host.

Note that the above description leaves out many details, but it does illustrate the value of an architecture that supports multiple principal types, fallbacks and intrinsic security. We discuss a number of additional case studies in §5. In what follows, we describe the foundations of the XIA design in more detail.

3. EXPRESSIVENESS WITH MULTIPLE PRINCIPAL TYPES

In the XIA architecture, the network provides concurrent support for different communication styles through the use of *principal types*. Multiple principal types enable applications or end-hosts to express the nature of their diverse intent. Note that even a single application may use different principal types for different tasks. For example, the Web browser in §2.2 uses content principals when it is fetching static content.

To define its unique "contract" with the network, each *type* of principal must define:

1. The semantics of communicating with a principal of that type.
2. A unique XIA identifier (XID) type and a method for generating XIDs and mapping these XIDs to intrinsic security properties of any communication.
3. Any principal type-specific per-hop processing and routing of packets that must be coordinated or made consistent in a distributed fashion.

The semantics (definition 1) help define the communication intent associated with a particular principal type (i.e., define what it means to send or receive a packet from an XID of that type). Examples of these intents might include fetching particular content or communicating with a particular host. In general, it makes sense to consider adding a new principal type when a new communication style or mode is not expressed efficiently with the existing types. For example, neither the host nor content principals may be well-suited to expressing a desire to communicate with all nodes in a geographic area. Later research could, therefore, define a "GeoCast" principal type [17] to meet this (hypothetical) need.

The second definition is used to generate intrinsically secure addresses that the network operates upon. We define intrinsic security as *the capability to verify type-specific security properties without relying on external information*. XIDs are therefore typically derived from their associated principal in some cryptographic type-specific manner, e.g., using a hash of the principal's public key or a hash of the content. This enables verification of each operation that a principal type supports. For example, when using a content XID, an application can verify that it received the right content, or when using a host XID, an application can authenticate that the host it is conversing with.

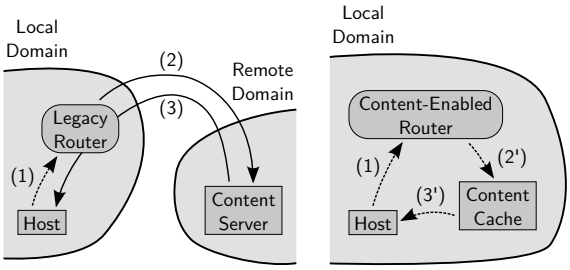
The third definition sets forth correctness requirements for in-network optimization for handling specific principal types. Many in-network optimizations can be handled locally at each router. In such cases, each router can handle packets as it desires, as long as it meets the semantics associated with the principal type. Examples of this type of processing are in-network content caching and support for anycast. Other type-specific support requires more coordination. For example, a group communication principal type would require that router implementations (for routers within a domain) agree upon routing or tree-construction protocols.

When a new principal type is introduced we can't expect complete and immediate network support. We explore how the new functions is gradually introduced into the network.

4. SUPPORTING EVOLUTION

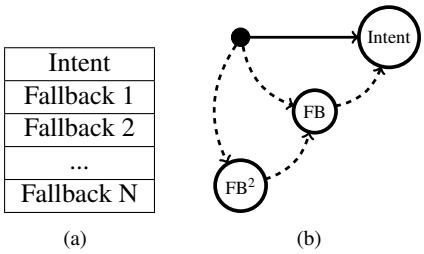
The architecture must pave a seamless path for an incremental deployment of new functionalities. Otherwise, the network cannot readily evolve, and the expressiveness would be limited to the legacy principal types.

Our approach to supporting incremental deployment of new principal types is through the notion of a *fallback*. Applications expressing an intent that may not be universally recognized are requested to specify alternative, *fallback* ways to achieve the intent. When a network element does not under-



(a) Partially deployed (host only) (b) Fully deployed (with in-network support)

Figure 1: An example of incremental deployment of content-centric networking. Solid arrows and dotted arrows respectively represent per-hop processing of host and content principals.



(a) (b)

Figure 2: The two XIA addressing options show how fallbacks may be encoded. Option (a) specifies an intent and fallbacks in an ordered list. Option (b) uses a directed acyclic graph to represent an intent, a fallback, and a fallback to a fallback.

stand how to satisfy an expressed intent, it fails over to inspect the fallback options, at least one of which legacy systems are capable of handling. For example, when retrieving content (Figure 1 (a)) using a content identifier, a host identifier can be used as a fallback. Routers that receive this packet will first try to operate on the primary intent (the content identifier), but would fall back to forwarding based on the host identifier either if they do not support the content principal type or if they do not know a route to the requested content identifier.

When the local domain natively supports the content principal type, it can potentially provide in-network optimization for content delivery. In Figure 1 (b), by understanding the content request at the router level (step $2'$), it can serve the request directly from a local in-network cache (step $3'$).

In general, only the end-hosts involved in a communication using a given principal need to support that principal type for correctness, and consequently, incremental deployment is straight-forward. This property is beneficial from several perspectives: 1) introduction of a new principal type is easy – one does not have to rely on a global deployment of principal-specific components; 2) each network operator can decide for itself, based on economic, technological, or other considerations, which set of *principal types* it wants to support natively and which set of *principals* of a given type to support without affecting the inter-operability of the network; and 3) endpoints and operators can flexibly adjust these decisions without any external coordination. By requiring that all routers support host-based communication, one's intent can always be satisfied by an end-host fallback who understands how to process the intent. Thus, fallbacks ensure that the network can service an intent without disruption, regardless of the status of network support for a principal type.

We now turn to practical considerations: how XIA end-hosts and applications obtain fallback information, and how fallbacks are encoded in XIA packets.

Obtaining fallback information: While obtaining fallback information in XIA can be seen as an additional burden for end-hosts or applications, in practice, it can be done in conjunction with name resolution or other mechanisms through which applications resolve the XID. For example, a name server can provide fallback hosts for a service. These fallbacks are provided by the service provider who had initially registered the service name. Similarly, Web servers who provide content identifiers can also provide a fallback host or a service from which the content can be retrieved. This process does not involve any further bookkeeping than what is done today in host-based communication. We, therefore, believe that the cost of obtaining and maintaining fallback information will be minimal in most cases, but further facilitating this process is open to future research.

Encoding fallback: We list a couple of options for encode multiple identifiers into XIA packet headers to support fallback. One way of representing fallback in a packet header is to have multiple destination address fields and encode fallbacks as an alternative destination (Figure 2 (a)). Another option is to encode a directed acyclic graph which represents the original intent and fallback identifiers as alternative paths. The latter generalizes the approach in Slick Packets [18], and is a more flexible representation. For example, one can express fallbacks to a fallback as in Figure 2 (b).

5. CASE STUDIES

Previously, we have shown how popular usage models of today's Internet can be supported using service and content principal types, and we discussed how to overcome the challenge of evolving the set of principal types. In this section, we present two additional case studies that highlight how XIA can support more diverse use cases.

5.1 Supporting Mobility and Disruptions

One implication of the best effort service provided by today's Internet is that *both* end-points have to be connected to the Internet *simultaneously* for effective end-to-end communication to take place. This is limiting for applications as disruptions are quite common in typical mobile scenarios. Unlike the Internet, architectures such as DTNs [9] handle disruptions, but they are not optimized for the case where both end-points are connected. Because of the complicated per-hop behavior

such as custody transfer, DTN results in sub-optimal performance under good connectivity. Ideally, we would like to seamlessly switch between a "good performance" mode to a DTN-like mode, depending on the connectivity of end-points.

In XIA, we can achieve this using a combination of host and service principal types. We can imagine a DTN "service" that stores data temporarily when the mobile host is disconnected from the network and subsequently delivers the data when the mobile host connects again [8]. Mobile users can subscribe to such a service. Using it in a good connectivity scenario will typically result in sub-optimal performance, since it may not be in the direct path of communication. Moreover, the DTN service may charge the user based on the traffic it stores/forwards on behalf of the user. Therefore, the mobile host will want to use this service only when it is disconnected.

XIA naturally supports such a solution – one that uses host based communication under a normal scenario and service based communication under periods of disconnectivity – through the notion of intent and fallback. Packets destined for the mobile host will have the identifier of the mobile host as the intent and the identifier of the DTN service as a fallback. If a network element cannot reach the host, then it will use the fallback and send the packet to the DTN service.[1] Once the mobile host is connected again, it can use any mechanism (push, pull) to retrieve the packet from the DTN service.

The benefit is quite obvious when both end-points are mobile (e.g., a direct chat session between two mobile users). In today's Internet, the chat message will only be delivered if both end-points are connected to the network *at the same time*. However, XIA will seamlessly switch from an interactive chat session to an email-like communication if one of the end-points gets disconnected.

There are advantages even when only one end-point is mobile and the other end-point is connected—a typical scenario for mobile devices. In such a scenario, the use of a generic DTN service eliminates the need for servers to bear the responsibility of supporting disconnections by *delegating* the data transfer responsibility to the DTN service, thereby allowing servers to potentially serve more users.

5.2 Seamless Multicast

Multicast offers an effective mechanism to reduce bandwidth for single-to-many or many-to-many communications. However, while native multicast has been deployed in local networks and across single domains, wide-area multicast has not seen broad adoption due to various concerns including scalability, deployment overhead, and network management issues. On the other hand, overlay-based solutions such as end-system multicast [11] have been broadly deployed in WAN environments by carefully circumventing these issues. Ultimately, pushing overhead such as membership management, overlay construction and maintenance, and multicast forwarding out of the network core, and onto end systems, amounts to clever realization of multicast functionality via a complex amalgam of pairwise *host-based* communications.

XIA enables multicast applications to easily leverage different implementations and partial deployments of multicast that may coexist, including native multicast. The first key is that in XIA, the notion of multicast intent (for simplicity, we focus on point-to-multipoint intent) can be expressed as a new principal type with multicast sessions as the principals. This expressiveness enables us to get the benefits of limited deployments of multicast, while having the ability to switch to service-based multicast wherever native multicast is not supported. This can be achieved by specifying the multicast address as a primary intent and the service identifier of a multicast service as the fallback. When native multicast is present in a domain, those participants within the domain will be served by native multicast. On the other hand, when a packet with multicast intent reaches a network that does not support this principal type, the packet is routed to the handler for the multicast service, which ensures the delivery of packets to all of its recipients, and neighboring domains. XIA also enables us to easily construct a hybrid multicast system of service-based multicast and islands of networks with native multicast support. As before, the applications specify the multicast address as a primary intent and the identifier of the multicast service as the fallback address. The multicast service distributes packets through a tunnel to each of its subtrees, some of which may be directed to a rendezvous point of a multicast enabled network.

6. OUR WORK IN CONTEXT

Substantial prior works [6, 10, 12, 13] have examined the benefits of network architectures tailored for individual principals; in general, they are complementary to our work and have motivated the design of our architecture that supports multiple principal types.

Extensibility through naming and indirection: Prior research has examined solutions that extend network functionality purely through naming or overlay-based indirection. Layered Naming Architecture (LNA) [5] resolves service and data identifiers to end-point identifiers (hosts) and end-point identifiers to IP addresses. Like XIA, this architecture improves support for mobility, anycast, and multicast, but only at the cost of additional indirection. Similarly, *i3* [23] uses an overlay infrastructure that mediates on sender-receiver communication to provide enhanced flexibility.

Extensibility through programmability has been pursued through efforts, such as active networks [24]. The biggest drawback to the extreme flexibility of such approaches is resource isolation and security. In contrast, XIA does not make it easier to program new functionality into existing routers, but creates an architecture in which new functionality can be incrementally deployed.

Architectures that evolve well: FII [14] shares our goals of supporting evolution and diversity, but their design focuses

[1] Typically, the fallback will be used once the packet is inside the access network to which the mobile host was last connected.

primarily on improved interfaces for inter-domain routing and applications, whereas XIA targets innovation and evolution of data plane functionality within or across domains. FII allows a domain to adopt any architecture, but does not specify how to do so incrementally, while XIA's fallback mechanisms allows incremental adoption of new principal types by design. Ratnasamy et al. propose modifications to IP to enhance its evolvability [20]; compared to XIA, this work seems more easily deployable, but does not admit the flexibility offered by XIA's support for multiple principal types. Others have argued that we should give up and accept that IP and HTTP atop it are here to stay, and simply build the next networks atop them [19]. While we politely disagree, we cannot argue the vast inertia of today's Internet, but hope that our work proves useful regardless in the design of future networks.

Substantial work has examined creating *virtualizable networks* in which links can be partitioned to allow many competing Internet protocols to run concurrently [3, 4, 22, 26]. Clark, in particular, presents a compelling argument for the need to enable competition at an architectural level [7], which we internalized in our support for multiple principals. We believe that there are substantial benefits to ensuring that all applications can communicate with all other applications using the same "Internet", but virtualizable networks offer the potential for stronger isolation properties and to support deeper-reaching architectural changes.

Borrowed foundations: *Self-certifying identifiers* were explored in various other systems [15, 16]. AIP [2] used self-certifying network and host identifiers to simplify network-level security mechanisms. Similarly, DONA [13] and SCAFFOLD [10] have used self-certifying content and service identifiers. This prior work demonstrated the substantial power of these intrinsically secure identifiers, which XIA in turn generalizes to an architectural requirement.

7. CONCLUSION

The design of the XIA architecture started from the premise that today's Internet is limited by the specificity and inflexibility of the implied contract underlying IP, notably host-based addressing, open-ended trust assumptions, and little provision for security. The XIA architecture is founded on the notion of intrinsically secure principals that allow applications to formally and expressively state their intent, and verify that their intent has been satisfied correctly. The key requirement of evolvability is realized in XIA by virtue of the ability to introduce new principal types over time, and to do so in an incrementally deployable manner using the notion of a fallback. Secure content retrieval, disruption-tolerant communication, and multicast provided representative examples of the value that the XIA architecture can provide.

ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation under awards CNS-1040757, CNS-1040800, and CNS-1040801. Ashok Anand is supported by a Google PhD Fellowship. Dongsu Han and Hyeontaek Lim are supported in part by the Korea Foundation for Advanced Studies.

REFERENCES

[1] Pittsburgh, PA, Aug. 2002.

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

[3] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet impasse through virtualization. *IEEE Computer*, 38, Apr. 2005.

[4] M. B. Anwer and N. Feamster. Building a Fast, Virtualized Data Plane with Programmable Hardware. In *Proc. ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, Aug. 2009.

[5] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the Internet. In *Proc. ACM SIGCOMM*, pages 343–352, Aug. 2004.

[6] D. R. Cheriton and M. Gritter. TRIAD: A new next-generation Internet architecture. Technical report, Jan. 2000.

[7] D. Clark, J. Wroclawski, K. Sollins, and B. Braden. Tussle in cyberspace: Defining tomorrow's Internet. In *Proc. ACM SIGCOMM* sig [1], pages 347–256.

[8] F. R. Dogar and P. Steenkiste. M2: Using Visible Middleboxes to Serve Proactive Mobile-Hosts. In *Proc. ACM MobiArch*, 2008.

[9] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proc. ACM SIGCOMM*, pages 27–34, Aug. 2003.

[10] 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 University, Sept. 2010.

[11] Y. hua Chu, S. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *Selected Areas in Communications, IEEE Journal on*, 20(8), Oct. 2002.

[12] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proc. CoNEXT*, Dec. 2009.

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

[14] T. Koponen, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. B. Godfrey, N. McKeownh, G. Parulkari, B. Raghavan, J. Rexford, S. Arianfar, and D. Kuptso. Architecting for Innovation. *ACM CCR*, 2011.

[15] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proc. ACM SOSP*, Dec. 1999.

[16] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. Interent Engineering Task Force, RFC 4423, May 2006.

[17] J. C. Navas and T. Imielinski. GeoCast–geographic addressing and routing. In *Proc. ACM MOBICOM*, pages 66–76, Sept. 1997.

[18] G. T. K. Nguyen, R. Agarwal, J. Liu, M. Caesar, B. Godfrey, and S. Shenker. Slick packets. In *Proc. SIGMETRICS*, 2011.

[19] L. Popa, A. Ghodsi, and I. Stoica. HTTP as the narrow waist of the future Internet. In *Proc. ACM Hotnets-IX*, Oct. 2010.

[20] S. Ratnasamy, S. Shenker, and S. McCanne. Towards an evolvable Internet architecture. In *Proc. ACM SIGCOMM*, Aug. 2005.

[21] U. Saif and J. Mazzola Paluska. Service-oriented network sockets. In *Proc. ACM MobiSys*, May 2003.

[22] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *Proc. 9th USENIX OSDI*, Oct. 2010.

[23] I. Stoica, D. Adkins, S. Zhaung, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. ACM SIGCOMM* sig [1], pages 73–86.

[24] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. *ACM CCR*, 26(2):5–18, Apr. 1996.

[25] D. Trossen, M. Sarela, and K. Sollins. Arguments for an information-centric internetworking architecture. *ACM CCR*, 40:26–33, Apr. 2010.

[26] G. Watson, N. McKeown, and M. Casado. NetFPGA: A tool for network research and education. In *Proc. 2nd workshop on Architectural Research using FPGA Platforms (WARFP)*, 2006.