XIA: Efficient Support for Evolvable Internetworking

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

Abstract

Motivated by limitations in today's host-centric IP network, recent studies have proposed clean-slate network architectures centered around alternate 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. This paper 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. We describe key design requirements, and demonstrate how XIA's rich addressing and forwarding semantics facilitate flexibility and evolvability, while keeping core network functions simple and efficient. We describe case studies that demonstrate key functionality XIA enables.

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 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 at the waist. This shortcoming is clearly illustrated by the 15+ year deployment history of IPv6 and the difficulty of deploying primitives needed to secure the Internet. Serious barriers to evolvability arise when:

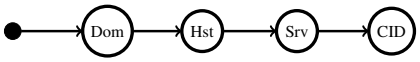
- Protocols, formats, and information must be agreed upon by a large number of independent actors in the architecture; and
- There is no built-in mechanism that supports (and embraces) incremental deployment of new functionality with minimal friction.

IP today faces both of these barriers. First, senders, receivers, and every router in between must agree on the format and meaning of the IP header. It is not possible, therefore, for a destination to switch to IPv6-based addressing and still remain reachable by unmodified senders who use IPv4. Second, today's paths to incremental deployment typically involve tunnels or overlays, which have the drawback that they *hide* the new functionality from the existing network. For example, enabling a single router in a legacy network to support some form of content-centric networking is fruitless if that traffic ends up being tunneled through the network using IPv4[1].

This paper presents a new Internet architecture, called the eXpressive Internet Architecture or XIA, that addresses these problems from the ground up. XIA maintains some features of the current Internet, such as a narrow waist that networks must support, and packet switching, but it differs from today's Internet in several areas.

The philosophy underlying the design of XIA is, simply, that we do not believe we can predict the usage models for the Internet of 50 years hence. The research community has presented compelling arguments for supporting many types of communication—content-centric networking [26, 44], service-based communication [20, 38], multicast [19], enhanced support for mobility [4, 40, 48], and so on. We believe that a new network architecture should facilitate *any or all* of these capabilities, and it must be possible to enable or disable "native" support for them as makes sense in that time and place.

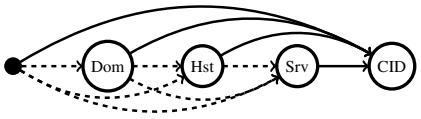
The key architectural element that XIA adds to improve evolvability is one we call expressing *intent*. XIA's addresses can simultaneously express both a "new" type of address (or addresses), and one or more backwards compatible pathways to reach that address. This notion is best explained by example: Consider the process of retrieving a particular piece of content (CID) using a network *that provides only host-to-host communication*, much like today's Internet. The source would send a packet destined to a destination network (Dom); the destination network would deliver it to a host; the host would deliver it to a process providing a service (Srv) such as HTTP; and the process would reply with the desired content, as such:



Carnegie Mellon University
[*]Department of Computer Science, Boston University
[†]University of Wisconsin-Madison

[1]Without resorting to deep packet inspection of various sorts, which itself is typically fragile.

XIA makes this path explicit in addressing, and allows flexibility in expressing it, e.g., "The source really just wants to retrieve this content, and it does not care whether it goes through Dom to get it." As a result, this process of content retrieval might be expressed in XIA by specifying the destination address as a *directed acyclic graph*, not a single address, like this:



By expressing the destination in this way, senders *give flexibility to the network to satisfy their intent*. Imagine a future network in which the destination domain supported routing directly to services [20] (instead of needing to route to a particular host). Using the address as expressed above, this hypothetical network would already have both the *information* it needs (the service ID) and *permission* to do so (the link from the source directly to the service ID). A router or other network element that does not know how to operate on, e.g., the service ID, would simply route the packet to the furthest-along node that it does know (the domain or the host in this example). Note that the sender can use the same address before and after support for service routing is introduced.

XIA terms the types of nodes in the address *principals*; examples of principals include hosts, autonomous domains (analogous to today's autonomous systems), services, content IDs, and so on. The set of principals in XIA is not fixed: hosts or applications can define new types of principals and begin using them *at any time*, without waiting for support from the network. Of course, if they want to get anything done, they must also express a way to get their work done in the current network. We believe that the ability to express not just "how to do it today", but also your underlying intent, is key to enabling future evolvability.

The second difference between XIA and today's Internet comes from a design philosophy that encourages creating principals that have *intrinsic security*: the ability for an entity to validate that it is communicating with the correct counterpart without needing access to external databases, information, or configuration. An example of an intrinsically secure address is using the hash of a public key for a host address [10]. With this mechanism, the host can prove that it sent a particular packet to any receiver who knows its host address. Intrinsic security is central to reliable sharing of information between principals and the network and to ensuring correct fulfillment of the contract between them. It can furthermore be used to bootstrap higher level security mechanisms.

We make the following contributions in this paper: We outline novel design ideas for evolution (§2), and systematically incorporate them into the eXpressive Internet Protocol (XIP) (§3). We describe optimizations for high-speed per-hop packet processing, which can achieve multi-10Gbps forwarding speed. Then, through concrete examples, we show how networks, hosts, and applications interact with each other and benefit from XIA's flexibility and evolvability (§4). Through prototype implementation and deployment, we show how applications can benefit, and demonstrate the practicality of XIP and the architecture under current and (expected) future Internet scales and technology (§5). We discuss related work (§6) and close with a conclusion and a list of new research question that XIA raises (§7).

2 Foundational Ideas of XIA

XIA is based upon three core ideas for designing an evolvable and secure Internet architecture:

1. Principal types. Applications can use one or more principal types to directly express their intent to access specific functionality. Each principal type defines its own "narrow waist", with an interface for applications and ways in which routers should process packets destined to a particular type of principal.

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 formalize. As new principal types are introduced, applications or protocols may start to use these new principal types 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 §4.2 and §4.3.

2. Flexible addressing. XIA aims to avoid the "bootstrapping problem": why develop applications or protocols that depend on network functionality that does not yet exist, and why develop network functionality when no applications can use it? XIA provides a built-in mechanism for enabling new functions to be deployed piecewise, e.g., starting from the applications and hosts, then, if popular enough, providing gradual network support. The key challenge is: how should a legacy router in the middle of the network handle a new principal type that it does not recognize? To address this, we introduce the 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 §3.2.

3. Intrinsically secure identifiers. IP is notoriously hard to secure, as network security was not a first-order consideration in its design. XIA aims to build security into the core architecture as much as possible, without impairing expressiveness. In particular, principals used in XIA source and destination addresses must 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, a publisher can attest that it delivered specific bytes to the intended recipients. While the implementation of intrinsic security is not a focus of this paper, we briefly describe the intrinsic security of our current principal types in §3.1, as well as the specification requirements for intrinsic security for new principal types.

3 XIP

XIA facilitates communication between a richer set of principals than many other architectures. We therefore split both the design and our discussion of communication within XIA into two components. First, the basic building block of per-hop communication is the core eXpressive Internet Protocol, or XIP. XIP is principal-independent, and defines an address format, packet header, and associated packet processing logic. A key element of XIP is a flexible format for specifying multiple paths to a destination principal, allowing for "fallback" or "backwards-compatible" paths that use, e.g., more traditional autonomous system and host-based communication.

The second component is the per-hop processing for each principal type. Principals are named with typed, unique eXpressive identifiers which we refer to as XIDs. In this paper, we focus on host, service, content, and administrative domain principals to provide an example of how XIA supports multiple principals. We refer to the above types as HIDs, SIDs, CIDs, and ADs, respectively. The list of principal types is extensible and more examples can be found in our prior work [8].

Our goal is for the set of mandatory and optional principals to evolve over time. We envision that an initial deployment of XIA would mandate support for ADs and HIDs, which provide global reachability for host-to-host communication—a core building block today. Support for other principal types would be optional and over time, future network architects could mandate or remove the mandate for these or other principals as the needs of the network change. Or, put more colloquially, we do not see the need for ADs and HIDs disappearing any time soon, but our own myopia should not tie the hands of future designers!

3.1 Principals

The specification of a principal *type* must define:

1. The semantics of communicating with a principal of that type.
2. A unique XID type, a method for allocating XIDs and a definition of the intrinsic security properties of any communication involving the type. These intrinsically secure XIDs should be globally unique, even if, for scalability, they are reached using hierarchical means, and they should be generated in a distributed and collision-resistant way.
3. Any principal-specific per-hop processing and routing of packets that must either be coordinated or kept consistent in a distributed fashion.

These three features together define the *principal-specific support* for a new principal type. The following paragraphs describe the administrative domain, host, service, and content principals in terms of these features.

Network and *host* principals represent autonomous routing domains and hosts that attach to the network. ADs provide hierarchy or scoping for other principals, that is, they primarily provide control over routing. Hosts have a single identifier that is constant regardless of the interface used or network that a host is attached to. ADs and HIDs are self-certifying: they are generated by hashing the public key of an autonomous domain or a host, unforgeably binding the key to the address. The format of ADs and HIDs and their intrinsic security properties are similar to those of the network and host identifiers used in AIP [10].

Services represent an application service running on one or more hosts within the network. Examples range from an SSH daemon running on a host, to a Web server, to Akamai's global content distribution service, to Google's search service. Each service will use its own application protocol, such as HTTP, for its interactions. An SID is the hash of the public key of a service. To interact with a service, an application transmits packets with the SID of the service as the destination address. Any entity communicating with an SID can verify that the service has the private key associated with the SID. This allows the communicating entity to verify the destination and bootstrap further encryption or authentication.

In today's Internet, the true endpoints of communication are typically application processes—other than, e.g., ICMP messages, very few packets are sent to an IP destination without specifying application port numbers at a higher layer. In XIA, this notion of processes as the true destination can be made explicit by specifying an SID associated with the application process (e.g., a socket) as the intent. An AD, HID pair can be used as the "legacy path" to ensure global reachability, in which case the AD forwards the packet to the host, and the host "forwards" it to the appropriate process (SID). In §4, we show that making the true process-level destination explicit facilitates transparent process migration, which is difficult in today's IP networks because the true destination is hidden as state in the receiving end-host.

Lastly, *the content principal* allows applications to express their intent to retrieve content without regard to its location. Sending a request packet to a CID initiates retrieval of the content from either a host, an in-network content cache, or other future source. CIDs are the cryptographic hash (e.g., SHA-1, RIPEMD-160) of the associ-

ated content. The self-certifying nature of this identifier allows any network element to verify that the content retrieved matches its content identifier.

3.2 XIP Addressing

Next, we introduce key concepts for XIP addresses that support the long-term evolution of principal types, the encoding mechanism for these addresses, and a representative set of addressing "styles" supported in XIP.

3.2.1 Core concepts in addressing

XIA provides native support for multiple principal types, allowing senders to express their intent by specifying a typed XID as part of the XIP destination address. However, XIA's design goal of evolvability implies that a principal type used as the intent of an XIP address may not be supported by all routers. Evolvability thus leads us to the architectural notion of **fallback**: intent that may not be globally understood *must* be expressed with an alternative backwards compatible route, such as a globally routable service or a host, that can satisfy the request corresponding to the intent. This fallback is expressed *within* an XIP address since it may be needed to reach the intended destination.

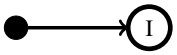
XIP addressing must also deal with the fact that not all XID types will be globally routable, for example, due to scalability issues. This problem is typically addressed through scoping based on network identifiers [10]. Since XIA supports multiple principal types, we generalize scoping by allowing the use of XID types other than ADs for scoping. For example, scaling global flat routing for CIDs may be prohibitively expensive [12, 42], and, thus, requests containing *only* a CID may not be routable. Allowing the application to refine its intent using hierarchical **scoping** using ADs, HIDs, or SIDs to help specify the CID's location can improve scalability and eliminate the need for XID-level global routing. We explore the effectiveness of using this more scalable approach in §5.3.

The drawback of scoping intent is that a narrow interpretation could limit the network's flexibility to satisfy the intent in the most efficient manner, e.g., by delivering content from the nearest cache holding a copy of the CID, rather than routing to a specific publisher. We can avoid this limitation by combining fallback and scoping, a concept we call (iterative) **refinement** of intent. When using refinement of intent, we give the XID at each scoping step the opportunity to satisfy the intent directly without having to traverse the remainder of the scoping hierarchy.

3.2.2 Addressing mechanisms

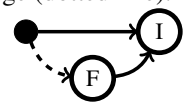
XIA's addressing scheme is a direct realization of these high-level concepts. To implement fallback, scoping, and iterative refinement, XIA uses a restricted directed acyclic graph (DAG) representation of XIDs to specify XIP addresses. A packet contains both the destination DAG and the source DAG to which a reply can be sent. Because of symmetry, we describe only the destination address.

Three basic building blocks are: intent, fallback, and scoping. XIP addresses must have a *single* intent, which can be of any XID type. The simplest XIP address has only a "dummy" source and the intent (I) as a sink:



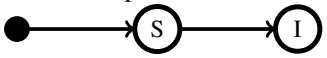
The dummy source (•) appears in all visualizations of XIP addresses to represent the conceptual source of the packet.

A fallback is represented using an additional XID (F) and a "fallback" edge (dotted line):



The fallback edge can be taken if a direct route to the intent is unavailable; we allow up to four fallbacks.

Scoping of intent is represented as:



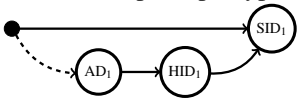
This structure means that the packet must be first routed to a scoping XID S, even if the intent is directly routable.

These building blocks are combined to form more generic DAG addresses that deliver rich semantics, implementing the high-level concepts in §3.2.1. To forward a packet, routers traverse edges in the address in order and forward using the next routable XID. Detailed behavior of packet processing is specified in §3.3.2.

3.2.3 Addressing style examples

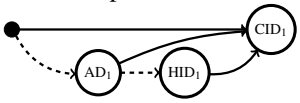
XIP's DAG addressing provides considerable flexibility. In this subsection, we present three (non-exhaustive) "styles" of how it might be used to achieve important architectural goals.

Supporting evolution: The destination address encodes a service XID as the intent, and an autonomous domain and a host are provided as a fallback path, in case routers do not understand the new principal type.



This scheme provides both fallback and scalable routing. A router outside of AD_1 that does not know how to route based on intent SID_1 directly will instead route to AD_1.

Iterative refinement: In this example, every node includes a direct edge to the intent, with fallback to domain and host-based routing. This allows iterative incremental refinement of the intent. If the CID_1 is unknown, the packet is then forwarded to AD_1. If AD_1 cannot route to the CID, it forwards the packet to HID_1.



An example of the flexibility afforded by this addressing is that an on-path content-caching router could directly reply to a CID query without forwarding the query to the content source. We term this *on-path interception*. Moreover, if technology advances to the point that content IDs became globally routable, the network and applications could benefit directly, without changes to applications.

Service binding and more: DAGs also enable application control in various contexts. In the case of legacy HTTP, while the initial packet may go to any host handling the web service, subsequent packets of the same "session" (e.g., HTTP keep-alive) *must* go to the same host. In XIA, we do so by having the initial packet destined for: $\bullet \rightarrow AD_1 \rightarrow SID_1$. A router inside AD_1 routes the request to a host that provides SID_1. The service replies with a source address bound to the host, $\bullet \rightarrow AD_1 \rightarrow HID_1 \rightarrow SID_1$, to which subsequent packets can be sent.

Other uses of DAGs are described in [9]. Some potential uses of DAG-based addresses, such as source routing, raise questions of policy and authorization that we do not explore in this paper. Here, we focus on supporting evolvability, refinement, and closely related uses such as binding and rebinding (§4).

3.3 XIP Header and Per-Hop Processing

This section describes the XIP packet format and per-hop processing that routers perform on packets. Later, in §5, we show that this design satisfies both the requirements for flexibility and efficient router processing.

3.3.1 Header format

Figure 1 shows the header format. Our header encodes a source and a destination DAG, and as a result our address is variable-length—`NumDst` and `NumSrc` indicate the size of the destination and source address. The header contains fields for version, next header, payload length, and hop limit. More details are described in [9].

Our header stores a pointer, `LastNode`, to the previously visited node in the destination address, for routers to know where to begin forwarding lookups. This makes per-hop processing more efficient by enabling routers to process a partial DAG instead of a full DAG in general.

DAGs are stored as adjacency lists. Each node in the adjacency list contains three fields: an XID Type; a 160-bit XID; and an array of the node indices that represent the node's outgoing edges in the DAG. The adjacency list format allows at most four outgoing edges per node (`Edge0...Edge3`). This choice balances: (a) the per-hop processing cost, overall header size, and simple router implementation; with (b) the desire to flexibly express many styles of addressing. However, we do not limit the degree of expressibility; one can express more outgoing edges by using a special node with a predefined `XIDType` to represent indirection.

Note that our choice of 160-bit XID adds large overhead, which could be unacceptable for bandwidth or power-limited devices. We believe, however, that common header compression techniques [30] can effectively reduce the header size substantially without much computational overhead.

3.3.2 Per-hop processing

Figure 2 depicts a simplified flow diagram for packet processing in an XIP router. The edges represent the flow of packets among processing components. Shaded elements are principal-type specific, whereas other elements are common to all principals. Our design isolates principal-type specific logic to make it easier to add support for new principals.

When a packet arrives, a router first performs source XID-specific processing based upon the XID type of the sink node of the source DAG. For example, a source DAG $\bullet \rightarrow AD_1 \rightarrow HID_1 \rightarrow CID_1$ would be passed to the CID processing module. By default, source-specific processing modules are defined as a no-op since source-specific processing is often unnecessary. In our prototype, we override this default only to define a processing module for the content principal type. A CID sink node in the source DAG represents content that is being forwarded to some destination. The prototype CID processing element opportunistically caches content to service future requests for the same CID.

The following stages of processing iteratively examine the outbound edges of the last-visited node of the DAG in priority order. We refer the node pointed by the edge in consideration as the next destination. To attempt to forward along an adjacency, the router examines the XID type of the next destination. If the router supports that principal type, it invokes a principal-specific component based on the type, and if it can forward the packet using the adjacency, it does so. If the router does not support the principal type or does not have an appropriate forwarding rule, it moves on to the next edge. This enables principal-specific customized forwarding ranging from simple route lookups to packet replication or diversion. If no outgoing edge of the last-visited node can be used for forwarding, the destination is considered unreachable and an error is generated.

3.3.3 Optimizations for high performance

The per-hop processing of XIA is more complex than that of IP, which raises obvious concerns about the performance of routers, especially in scenarios using more complex DAGs for addressing. In this section, we show that, despite those concerns, an XIP router can achieve comparable performance to IP routers by taking advantage of well-known optimizations, such as parallel processing

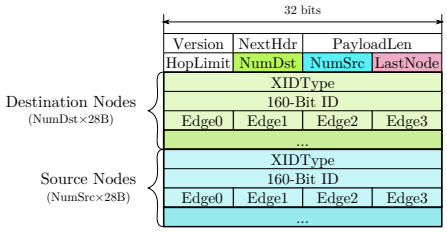


Figure 1: XIP packet header.

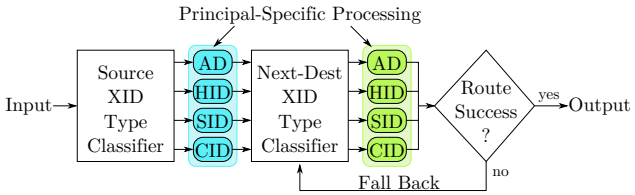


Figure 2: Simplified diagram of an XIP router.

and fast-path evaluation.

Packet-level parallelism: By processing multiple packets concurrently, *parallel packet processing* can speed up XIP forwarding. Fortunately, in XIP, AD and HID packet processing resembles IP processing in terms of data dependencies; the forwarding path contains no per-packet state changes at all. In addition, the AD and HID lookup tables are the only shared, global data structure, and like IP forwarding tables, their update rate is relatively infrequent (once a second or so). This makes it relatively straightforward to process packets destined for ADs and HIDs in parallel. While SID and CID packet processing may have common data structures shared by pipelines, any data update can be deferred for less synchronization overhead as the processing can be opportunistic and can always fall back into AD and HID packet processing. This makes CID and SID packet processing parallelizable.

Packet-parallel processing may result in out-of-order packet delivery, which disrupts existing congestion control mechanisms in TCP/IP networks [31]. One solution is to preserve intra-flow packet ordering by serializing packets from the same flow and executing them in the same pipeline processor, or alternatively by using a reordering buffer [22, 46]. An alternative solution is to design to ensure that congestion control and reliability techniques deployed in XIP networks are more tolerant of reordering [13].

Intra-packet parallelism: As discussed earlier, a DAG may encode multiple next-hop candidates as a forwarding destination. Since the evaluation of each candidate can be done in parallel, this address structure also enables *intra-packet parallel processing*. While the different next-hops can be evaluated in parallel, the results of these lookups must be combined and only the highest priority next-hop candidate with a successful lookup should be used. Note that this synchronization stage is likely to be expensive in software implementations and this type of parallelism may be most appropriate in specialized hardware implementations.

Fast-path evaluation: Finally, the XIP design can use *fast-path* processing to speed commonly observed addresses—either as an optimization to reduce average power consumption, or to construct low cost routers that do not require the robust, worst-case performance of backbone routers. For example, our prototype leverages a look-aside cache that keeps a collision-resistant fingerprint of the destination DAG address[2] and the forwarding result (the output port and the new last-node value). When a packet's destination address matches an entry in the cache, the router simply takes the cached result and skips all other destination lookup processing. Otherwise, the router pushes the packet into the original slow-path processing path. Since the processing cost of this fast path does not depend on the address used, this performance optimization may also help conceal the impact of packet processing time variability caused by differences in DAG complexity.

In §5.1, we show that the combination of these optimizations enables XIP routers to perform almost as well as IP routers. In addition, we show that hardware implementations would be able to further close the gap between XIP and IP performance.

4 XIA Addresses in Action

We elaborate how the abstractions introduced in previous sections can be put to work to create an XIP network. The following subsections explain how addresses are created and obtained, and show how XIA's architectural components can work together to support rich applications.

4.1 Bootstrapping Addresses

We assume the existence of autonomous domains and a global inter-domain routing protocol for ADs, e.g., as discussed in [10]. We walk through how HIDs, SIDs, and CIDs join a network, and how communication occurs.

Attaching hosts to a network: Each host has a public/private key pair. As a first step, each host listens for a periodic advertisement that its AD sends out. This message contains the public key of the AD, plus possibly "well-known" services that the AD provides such as a name resolution service. Using this information, the host then sends an association packet to the AD, which will be forwarded by the AD routers to a service that can proceed

[2]The use of collision-resistant hash eliminates the need to *memcmp* potentially lengthy DAGs. The fingerprint is a collision-resistant hash on a portion of the XIP address, which consists of the last-visited node and a few next-hop nodes, effectively representing forwarding possibilities of the flow of the packets. Such an operation is shown to scale up to 222 Gb/s [41] in hardware. We assume this is implemented in the NIC. Modern network interfaces already implement hash-based flow distribution for IPv4 and IPv6 for receiver-side scaling and virtualization.

with authentication based on the respective public keys.

Advertising services and content: We have designed an XIA socket API which is described in detail in our technical report [9]. We describe here how hosts can advertise services or content using this API.

To advertise a service, a process running on a host first calls `bind()` to bind itself to a public key of the service. This binding inserts the SID (the hash of the service's public key) into the host's routing table so that the service is reachable through the host. Likewise, `putContent()` stores the CID (the hash of the content) in the host's routing table. Since at this point services and content are only locally routable, request packets will have to be scoped using the host's HID, e.g., $\bullet \rightarrow AD \rightarrow HID \rightarrow CID$, to reach the intent.

For a service or a content to be reachable more broadly, the routing information must be propagated. For example, an AD can support direct routing to services and content within its domain using an intra-domain routing protocol that propagates the SIDs and CIDs supported by each host or in-network cache to the routers. Global route propagation can be handled by an inter-domain routing protocol, subject to the AD's policies and business relationships. We leave the exact mechanism, protocol, and policy for principal-specific routing (e.g., content or service routing) as future research.

Obtaining XIP addresses: Now we look into how two parties (hosts, services or content) can obtain source and destination XIP addresses to communicate. As in today's Internet, obtaining addresses is the application's responsibility. Here, we provide a few example scenarios of how XIP addresses can be created.

Source address specifies the return address to the specific instance of the principal (i.e., a bound address). Therefore, when a principal generates a packet, the source address is generally of the form $\bullet \rightarrow AD \rightarrow HID \rightarrow XID$. The AD-prefix is given by the AD when a host attaches to the network; the HID is known by the host; and the XID is provided by the application, allowing the socket layer to create the source address. XIDs will often be ephemeral SIDs. In this case, the socket layer can automatically create an SID when `connect()` is issued to an SID socket without calling `bind()`. This is similar to the use of ephemeral ports in TCP/IP.

Destination address can be obtained in many alternative ways. One way is to use a name resolution service to resolve XIP addresses from human readable names. For example, a lookup of "Google search" in the name resolution service can return a DAG that includes the intent SID along with one or more fallback ADs that host (advertised) instances of the service (similar to today's DNS SRV records). Alternatively, a Web service can embed URLs in its pages that include an intent CID for an image or document, along with the original source

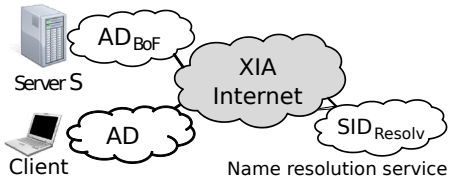


Figure 3: Bank of the Future example scenario.

($\bullet \rightarrow AD \rightarrow HID$) or content-distribution SIDs as fallbacks. This information can be in the form of a "ready-to-use" DAG, or as separate fields that can be assembled into a destination address (e.g., iterative refinement style) by the client based on local preferences. For example, the client could choose to receive content via a specific CDN based on the network interface it is using.

Note that we intentionally placed the burden of specifying fallbacks to the application layer. This is because a fallback is an authoritative location of an intent that the underlying network may not know about. Name resolution systems and other application-layer systems are more suitable to provide such information in a globally consistent manner. On the other hand, network optimizations can be applied locally in a much more dynamic fashion. Networks may choose to locally optimize for intent by locally replicating the object of intent and dynamically routing the intent.

A final point is that client ADs may have policies for what addresses are allowed. For example, it may want to specify that all packets entering or leaving the AD go through a firewall. This can be achieved by inserting an $SID_{Firewall}$ in the address, e.g., $\bullet \rightarrow AD \rightarrow SID_{Firewall} \rightarrow HID \rightarrow SID$ for a source address.

4.2 Simple Application Scenarios

In this section and the next, we use the example of online banking to walk through the lifecycle of an XIA application and its interaction with a client. Our goal is to illustrate how XIA's support for multiple principals and its addressing format give significant flexibility and control to the application.

In Figure 3, Bank of the Future (BoF) provides a secure on-line banking service hosted at a large data center on the XIA Internet. The service runs on many BoF servers and it has a public key that hashes to SID_{BoF}. We assume that all components in BoF's network natively support service and content principals. We focus on a banking interaction between a particular client host HID_C, and a particular BoF process P_S running on server HID_S.

Publishing the service: When process P_S starts on the server, it binds an SID socket to SID_{BoF} by calling `bind()` with its public/private key pair. This SID binding adds SID_{BoF} to the server's (HID_S) routing table, and the route to SID_{BoF} is advertised in the BoF network AD_{BoF}. The service also publishes the association between a human readable service name (e.g., "Bank of

the Future Online") and $\bullet \rightarrow AD_{BoF} \rightarrow SID_{BoF}$ through a global name resolution service (SID_{Resolv}).

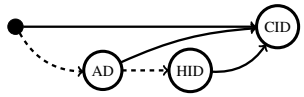
Connection initiation and binding: When a client wants to connect to the service, it first contacts the name resolution service SID_{Resolv} to obtain the service address. It then initiates a connection by sending a packet destined to $\bullet \rightarrow AD_{BoF} \rightarrow SID_{BoF}$ using the socket API. The source address is $\bullet \rightarrow AD_C \rightarrow HID_C \rightarrow SID_C$, where AD_C is the AD of the client, and SID_C is the ephemeral SID. This packet is routed to AD_{BoF} and then to an instance of SID_{BoF}. After the initial exchange, both processes will establish a session, which includes, for example, establishing a symmetric key derived from their public/private key pairs. Because of this session state, the client needs to continue to communicate with the same server, not just any server that supports SID_{BoF}. To ensure this, the client changes the destination address to $\bullet \rightarrow AD_{BoF} \rightarrow HID_S \rightarrow SID_{BoF}$, where HID_S is the server that it is currently talking to.

Content transfer: The client can now download content from the on-line banking service. For convenience, we assume that the content being transferred is a Web page. Let us consider both static (faq.html) and dynamic content (statement.html), both of which may contain static images. For *static (cacheable) content*, the SID_{BoF} service will provide the client with the CID_{faq} of the static Web page faq.html along with the CIDs of the images contained in it. The client can then issue parallel requests for those content identifiers, e.g., using $\bullet \rightarrow AD_{BoF} \rightarrow CID_{faq}$ as the destination address for the Web page. The request for *dynamic (non-cachable) content*, e.g., a list of recent bank transactions, is directly sent to SID_{BoF}.

4.3 Support for Richer Scenarios

Using the example above, we now show how XIA's addressing format can support more challenging scenarios such as evolution towards content networking, process migration, and client mobility. §5.2 provides an evaluation of these example scenarios.

Network evolution for content support: The previous section described how the client can specify static content using scoped intent. Switching to the iterative refinement style (§3):



means that routing through the specified *AD* or *HID* is optional, and opens the door for *any set of XIA routers to satisfy the client's request for static content*.

The DAG address format supports incremental deployment of content support in an XIA Internet. As a first step, BoF can deploy support for CIDs internally in its network. Even if no ISPs support CIDs, the above address will allow the delivery of the above request (using the *AD*) to the BoF network, where the intent CID can be served.

As the next step, some ISPs may incrementally deploy *on-path* caches. The above address allows them to opportunistically serve content, which may allow them to cut costs by reducing their payments to upstream service providers [6]. Over time, as support for content-centric networking expands, ISPs may make bilateral agreements to enable access to each other's (cached) content. XIA can help leverage such *off-path* caches as well; of course, it would require ISPs to exchange information about cached content and update router forwarding tables appropriately.

Process migration: XIA's addressing can also support seamless process (service) migration through re-binding. Suppose that the server process P_S migrates to another machine, namely HID_T, as part of load balancing or due to a failure; we assume that appropriate OS support for process migration is available. At the start of migration, the route to SID_{BoF} is removed from the old server and added to the new server's routing table. Before migration, the service communication was bound to $\bullet \rightarrow AD_{BoF} \rightarrow HID_S \rightarrow SID_{BoF}$. After migration, the OS notifies the ongoing connections of the new HID, and the binding is changed to $\bullet \rightarrow AD_{BoF} \rightarrow HID_T \rightarrow SID_{BoF}$ at the socket layer. Notification of the binding change propagates to the client via a packet containing the message authentication code (MAC) signed by SID_{BoF} that certifies the binding change. When the client accepts the binding change message, the client updates the socket's destination address. To minimize packet drops, in-flight packets from the client can be forwarded to the new server by putting a redirection entry to SID_{BoF} in the routing table entry of the old server.

Client mobility: The same re-binding mechanism can be used to support client mobility in a way that generalizes approaches such as TCP Migrate [40]. When a client moves and attaches to another AD, AD_{new}, the new source address of the client becomes: $\bullet \rightarrow AD_{new} \rightarrow HID_C \rightarrow SID_C$. When a rebind message arrives at the server, the server updates the binding of the client's address.

Although the locations of both the server and the client have changed in the previous two examples, the two SID end-points did not change. The intrinsic security property remains the same because it relies on the SID. Both the server and the client can verify that they are talking to the services whose public keys hash to SID_{BoF} and SID_C.

5 Evaluation

We evaluate the following three important aspects of XIA: *(i) Router processing:* Can we scale XIA's packet forwarding performance? In §5.1, we show that using techniques borrowed from fast IP forwarding, the speed of an XIA router can be made comparable to that of an IPv4 router. *(ii) Application design:* How does XIA benefit application design and performance? In §5.2, we show that use of

CPU	2x Intel Xeon L5640 2.26 GHz (12MB Cache, QPI 5.86 GT/s)
NIC	2x Intel Ethernet Server Adapter X520-T2
Motherboard	Intel Server Board S5520UR

Table 1: Router Hardware Specification

multiple principal types can simplify application design, and that applications can benefit from the type-specific in-network optimizations allowed by the architecture.
(iii) Scalable routing on XIA: How does forwarding and routing scale with the number of XIA identifiers in use? In §5.3, we show that XIA routing can scale well beyond support for today's network requirements to more extreme hypothetical scenarios.

5.1 Router Design and Performance

We first demonstrate that XIA's forwarding is fast enough for a practical deployment. We show that the packet processing speed of an XIA router is comparable to that of an IP router, and various techniques can be leveraged to further close the performance gap.

Implementation: To measure the packet processing speed, we set up a software router and a packet generator to exploit packet-level parallelism by leveraging their NIC's receiver-side-scaling (RSS) function to distribute IP and XIP packets to multiple CPU cores[3]. The implementation uses the Click modular router framework [28] for processing IP and XIP packets, and PacketShader's I/O Engine (NIC driver and library) [24] for sending and receiving packets to and from the NICs. Table 1 provides the specification of the machines.

Forwarding performance: We used a forwarding table of 351K entries based on the Route Views [35] RIB snapshot on Jan 1, 2011. IP uses this table directly; XIA pessimistically uses 351K entries for the AD forwarding table, associating each CIDR block with a distinct AD.

To measure XIA packet processing performance, we generate packets using five different DAGs for the destination address: `FB0`, `FB1`, `FB2`, `FB3`, and `VIA`. `FB0` is the baseline case where no fallback is used. `FB`i refers to a DAG which causes the XIA router to evaluate exactly i fallbacks and to then forward based on the $(i+1)$-th route lookup. To force this, we employ a DAG with i fallbacks: the intent identifier and the first $i-1$ fallback identifiers are not in the routing table, but the final fallback is. The last DAG, `VIA`, represents the case where an intermediate node in the DAG has been reached (e.g., arrived at the specified AD). In this case, the router must additionally update the last-visited node field in the packet header to point to the next node in the DAG before forwarding, unlike the other scenarios. Identifiers are generated based on a Pareto distribution over the set of possible destination ADs (shape parameter: 1.2) to mimic a realistic heavy-tailed distribution.

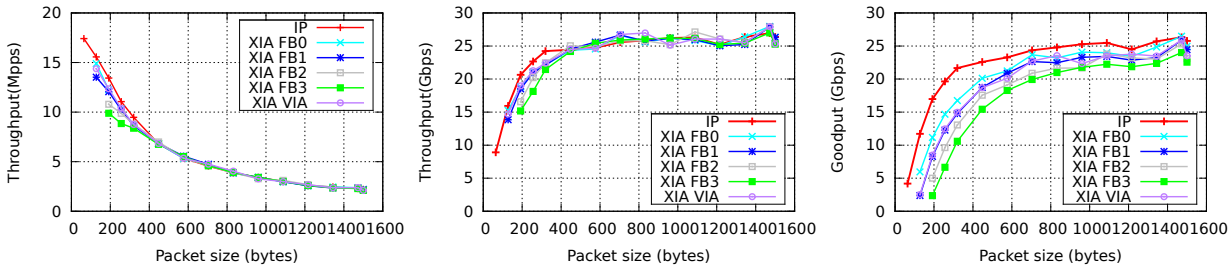
[3]To enable RSS on XIP, we prepend an IP header when generating the packet, but immediately strip the IP header after packet reception.

Figure 4 (a) and (b) respectively show the impact of varying packet size[4] on packet forwarding performance in packets and bits per second. Figure 4 (c) shows the actual goodput achieved excluding the header in each of the above experiments. The results are averaged over ten runs each lasting two minutes. Large packet forwarding is limited by I/O bandwidth, and therefore XIA and IP show little performance difference. For small packets of 192 bytes, XIA's `FB0` performance (in pps) is only 9% lower than `IP`. As more fallbacks are evaluated, performance degrades further but is still comparable to that of IPv4 (e.g., `FB3` is 26% slower than `IP`). However, the goodput is much lower due to XIA's large header size. We believe that in cases where the goodput is important, header compression techniques will be an effective solution.

Fast-path processing: We can further reduce the gap between IP forwarding and XIP forwarding using fast-path processing techniques outlined in §3.3.3. Our fast-path implementation uses a small per-thread table, which caches route lookup results. The key used for table lookups is a collision-resistant hash of the partial DAG consisting of the last visited node and all its outbound edges. Our choice of hash domain aligns with the fact that a given router operates only on a partial DAG. We assume that the NIC hardware performs this task upon packet reception and the driver reads the hash value along with the packet (§3.3.3). We emulated this behavior in our evaluation. We generate 351K unique partial DAGs, and each thread holds 1024 entries in the lookup table. In total, the table holds 14% of these partial DAGs. Figure 5 shows the result with and without this fast-path processing for packet size of 192 bytes. Without the fast-path, the performance degrades by 19% from `FB0` to `FB3`. However, with fast-path this difference is only 7%. With a marginal performance gain of IP fast-path, the gap between `FB3` and `IP` fast-path is reduced to 10%.

Intra-packet parallelism: Note that the fast-path optimizations do not improve worst-case performance, which is often critical for high-speed routers that must forward at line speed. High-speed IP routers often rely on specialized hardware to improve worst-case performance. Although we do not have such specialized hardware for XIP, we use micro-benchmarks to estimate the performance that might be possible. The micro-benchmark results in Figure 6 show that the route lookup time is dominant and increases as more fallbacks are looked up. Fallbacks within a packet can be processed in parallel

[4]We include a 14 byte MAC header in calculations of packet size and throughput. We only report the performance of packet sizes in multiples of 64 bytes because of limitations of our underlying hardware. When packet sizes do not align with the 64 byte boundary, DMA performance degrades significantly; we suspect this is triggering a known defect with our Intel hardware. This along with the additional IP and MAC header (34 bytes) and XIP's larger header size (minimum 64 bytes) resulted in a minimum packet size of 128 bytes for XIA.



(a) Throughput in Mpps (b) Throughput in Gbps (c) Goodput in Gbps

Figure 4: Packet forwarding performance of a software router. The forwarding table has 351 K entries.

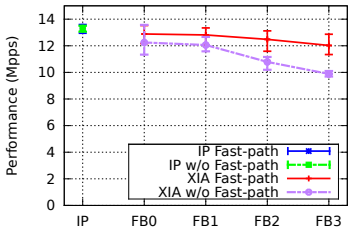


Figure 5: Fast-path processing smooths out total cost.

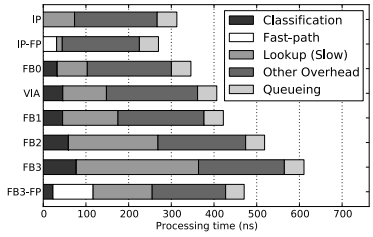


Figure 6: In-memory packet processing benchmark (no I/O).

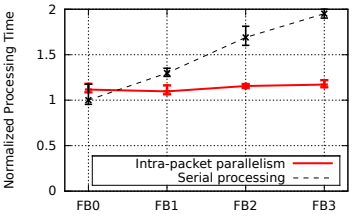


Figure 7: Parallel and serial route lookup cost.

using special-purpose hardware to further close the gap. Figure 7 shows a comparison between serial and four-way parallel lookup costs without the fast-path processing in our software router prototype, where we deliberately exclude the I/O and synchronization cost in the parallel lookup. The reason we exclude these costs is that while the overhead of intra-packet parallelism is high in our software implementation, we believe that such parallel processing overhead will be minimal in hardware router implementations or highly-parallel SIMD systems such as GPU-based software routers [24]. The figure shows that the performance gap between `FB0` and `FB3` can be eliminated with specialized parallel hardware processing.

In summary, our evaluation shows that DAG-based forwarding can be implemented efficiently enough to support high speed forwarding operations.

5.2 Application Design and Performance

We now evaluate XIP's support for applications by implementing and evaluating several application scenarios. While it is hard to quantitatively measure an architecture's support for applications, we demonstrate that XIA is able to retain many desirable features of the current Internet and subsume the benefits of other architectures [20, 26, 29]. Through this exercise, we show that XIP's flexible addressing and support for multiple principals simplifies application design, accommodates network evolution, and accelerates application performance.

Implementation: Using our XIA socket API [9], and our Click implementation of the XIP network stack, we implement in-network content support, service migration, and client mobility. To closely model realistic application behavior, we built a native XIA Web server that uses service and content principals, and a client-side HTTP-to-XIA proxy that translates HTTP requests and replies into communications based on XIP service and content principals. This proxy allows us to run an unmodified Web browser in an IP network. Implementing the proxy and server using our socket API required only 305 SLOC of Python code, and in-network content support took 742 SLOC of C++, suggesting that the principal-specific socket API and router design facilitates software design.

We now evaluate scenarios described in §4.2 and §4.3.

Content transfer and support for evolution: We created a wide-area testbed spanning two universities. The service side (CMU) operates the XIA Web server, and the client side (UWisc) performs Web browsing. The server serves dynamic and static Web pages, each of which consists of either a dynamic or static HTML file and a static image file (15 KB in each Web page). We use the addressing style described in §4.3.

Baseline content transfer: To highlight the application's use of multiple principals, we first show the baseline case where the content request takes the fallback path and the content is fetched from the origin server. Figure 8 (a) and (b) respectively show the steps and time taken to retrieve the dynamic and static Web page. When the document is dynamic (a), the server uses service-based communication to directly send the document; it also transmits a CID for the client to access the static image. In the static case (b), the service sends CIDs for both the document and the image, and the client fetches them

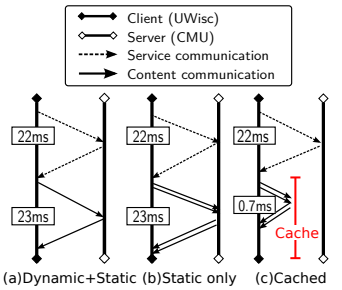


Figure 8: Content transfer.

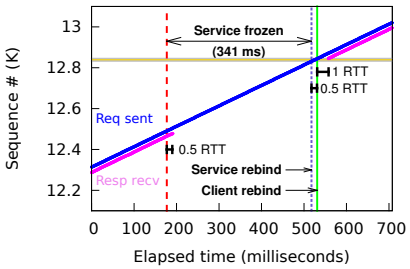


Figure 9: Service migration.

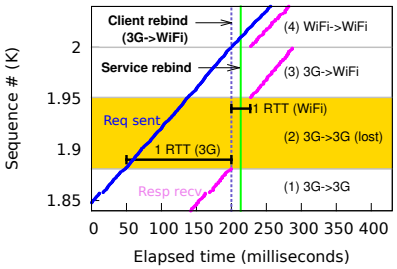


Figure 10: Client mobility.

using content communication. In both cases, it takes two round-trip times (45 ms) to retrieve the content.

Evolution: To highlight XIA's support for evolution, we consider two scenarios: 1) An in-network cache is added within the client's network without any changes to the endpoints, and 2) endpoints do not use XIA's fallback; clients first send the request to the primary intent, then redirect the intent to the original server after a timeout.

Figure 8(c) shows that content retrieval becomes faster (22.7 ms) just by adding an in-network cache without any changes (i.e., request packets in (b) and (c) are identical). This is enabled by iterative refinement-style addressing, which permits the intent to be satisfied at any step, while allowing it to fallback to the original server when it's not.

In the second scenario, the source does not use a fallback address and only uses a CID in its DAG address. The completion time becomes much worse (87 ms, not shown in the figure). The initial content request is dropped by the network because an intermediate router does not know how to route to the content. After a timeout, the application redirects the content request to the original server using the address: $\bullet \rightarrow AD_{BoF} \rightarrow HID_S \rightarrow CID$.

Service migration: As shown in §4.3, XIA supports seamless service migration with rebinding. To evaluate our service migration support, we run service SID_{BoF} on a virtual machine, which initially resides in a host machine (HID_S), and later migrates to another host (HID_T). We use KVM's live migration [3] to implement stateful migration of a running process; we move the VM along with the process running the service.

Figure 9 shows the timeline of service migration. The client makes continual requests to the service, who responds to these requests. Initially this session is bound to: $\bullet \rightarrow AD_{BoF} \rightarrow HID_S \rightarrow SID_C$. When live migration is initiated (not shown in the figure), the service continues to run on HID_S, but the underlying VM starts copying its state to the new host in the background. When most of the state is transfered to HID_T, SID_{BoF} (and the VM) is *frozen* to perform the final state transfer.

After the final state transfer, the VM and the service resume at HID_T. At this time, the service rebinds to HID_T (`Service rebind`). However, the client is still directing queries to HID_S, because it is not aware of the rebinding. The service then notifies the client of the new binding: $\bullet \rightarrow AD_{BoF} \rightarrow HID_T \rightarrow SID_C$. When the client receives this message, it rebinds to the new DAG after verification (`Client rebind`). The rebound client then starts sending subsequent requests to the new address. After one round-trip time, the client receives responses from the service. The communication is interrupted in between the rebinds. The duration of this interruption due to XIA address rebinding is minimal (the shaded region; about 1.5 RTT). The downtime due to VM migration (freezing) is much longer (341 ms) in our experiment. In a more sophisticated migration implementation, packet drops can be eliminated by buffering all packets received during the entire service interruption period at HID_S and redirecting them to the service at HID_T when it completes rebinding.

Client mobility: In this scenario, a client moves from a 3G network (RTT=150ms) to a WiFi network (RTT=25ms). The client is using a simple ping-like echo service on a server during the move. After connecting to the WiFi network, the client sends a cryptographically signed rebind message, notifying the server of the new binding: $\bullet \rightarrow AD_{new} \rightarrow HID_C \rightarrow SID_C$.

Figure 10 shows the timeline of events and the sequence numbers of packets from the echo service; blue dots indicate the time and the sequence number of the requests sent by the client, and purple dots those of the responses received by the client. Different regions (regions 1 to 4) indicate the network from which the request is sent from the client and to which the response is sent by the service. Only the packets in flight at `Client rebind` are lost (shaded area, region (2)) since they are sent to the 3G network, to which the client is no longer connected. However, after `Service rebind`, responses are sent to the WiFi network. Note that packet loss can be eliminated if the 3G network forwards these packets to the new location; this can be done by updating the routing entry for the client's HID on the 3G network. One round-trip time after `Client rebind`, the client begins receiving responses from the service. However, due to the difference in the round-trip times of the 3G and the WiFi network, packet reordering happens at the server-side, and the service responds to requests made from the 3G network prior to client rebind as well as from the WiFi network after client

	Forwarding Table	Public Key Store	Content Store
HIDs (100M)	6.25 GB	50 GB	-
SIDs (2 Billion)	125 GB	1 TB	-
CIDs (YouTube 2009 [16])	21 TB	-	168 PB
CIDs (WWW 2010 [18])	At least 227 TB	-	-

Table 2: Forwarding table size and public key store size of an AD with 100 million hosts.

rebind for a short period of time (regions (3) and (4)).

In summary, XIP's flexible addressing supports seamless network evolution and provides adequate application control, while keeping the application design simple.

5.3 Scalability

We now turn to a discussion of scalability. First, we demonstrate that XIA works as a "drop-in" replacement for IP, using AD and HID routing much as today's network uses CIDR blocks and ARP within subnets. We then examine scaling to plausible near-term scenarios, such as deploying on-path or near-path content caching, in which routers maintain large lists of content chunks stored in a nearby cache. We conclude by looking at long-term deployment ideas that would stretch the capabilities of near-term routers, such as flat routing within a large provider such as Comcast; and those requiring further advances to become plausible, such as global flat routing.

XIA at today's Internet scale. IP relies on hierarchy to scale: The "default-free" zone of BGP operates on only a few hundred thousand prefixes, not the 4 billion possible 32-bit IP addresses. XIA's AD principals can provide the same function, and we expect that at first, core routers would provide only inter-AD routing. Similar to AIP, the number of ADs should be roughly close to that of today's core BGP routing tables. §5.1 demonstrated that handling XIP forwarding with a routing table of this size is easy for a high-speed software router.

HID routing within a domain is likely to have a larger range, from a few thousand hosts, to several million. Many datacenters, for example, contain 100K or more hosts [33]. We expect that extremely large domains might split into a few smaller domains based upon geographic scope. XIP routing handles these sizes well: A 100K-entry forwarding table requires only 6.2MB in our (not memory optimized) implementation. Boosting the number of table entries from 10 K to 10 M decreases forwarding performance by only 8.3% (for Pareto-distributed flow sizes with shape=1.2) to 30.9% (for uniformly distributed flows); this slowdown is easily addressed by adding a small amount of additional lookup hardware.

While we believe naive flat forwarding will work in many networks, highly scalable systems, such as TRILL [5] and SEATTLE [27] can also be used.

Supporting tomorrow's scale. XIP provides considerable flexibility in achieving better scalability. First, XIP is not limited to a single level of hierarchy; a domain could add a second subdomain XID type beneath AD to improve its internal scalability. Second, other identifier types can be used to express hierarchy. For example, related content IDs can be grouped using a new principal (e.g., into an object ID). Doing so requires no cooperation from end-hosts: they must merely change the DAG address returned by naming to reflect the new hierarchy.

PortLand [33] suggests that a new layer-2 routing and forwarding protocol is needed to support millions of virtual machines (VMs) in data centers. In XIA, we can create a $HID_{Host} \rightarrow HID_{VM}$ hierarchy[5] that reduces the number of independently routable host identifiers and the forwarding table size by 1 to 2 orders of magnitude (we can also omit HID_{VM} by exposing all services in guest VMs to the host if the host's forwarding table is not overloaded, as in the service migration example of §5.2).

Hierarchy reduces forwarding table size at the cost of more bandwidth for headers; in XIA, this cost is modest: adding an extra XID requires 28 bytes/packet, but this addition might greatly simplify network components. Also, adding hierarchy in XIA does not hinder evolution. Using iterative refinement addressing, later networks could choose to ignore the hierarchy and route directly to the intended destination. This is the case even for hosts—were memory ever to become so cheap that global host-based routing was practical. More likely, however, this allows optimizations: A network might be willing to store a "redirect pointer" for a recently departed mobile host, similar to some optimizations proposed for Mobile IP. These pointers would operate on the host ID independent of hierarchy, but would be limited in number and duration.

On-path content caching and interception is a concrete example of opportunistic in-network optimization. A router forwards content (chunk) responses to a cache-engine, which caches the chunk. The router then intercepts the requests for chunks that are in the cache, and forwards these requests to the cache-engine, which serves the content. The forwarding table size can be small in this case, since it only contains information about the local cache (a 4 GB cache with an average object size of 7.8 KB [15] requires a forwarding table of size $<$32 MB.).

Hypothetical extremes. We now look at some extreme scenarios to better understand what kind of advances that are needed to make them feasible in XIA.

Some of the largest organizations have tens of millions of hosts. The largest cable operator has about 23 million customers [1], and Google is preparing to manage 10 million servers in the future [2]. YouTube (2009) has 1.8 billion videos [16] (large objects), and the World Wide Web (2010) has at least 60 billion pages (small objects). Table 2 shows the space needed to route on HIDs and SIDs

[5]For readers concerned about the performance penalty for having to go through a host, a simple filter can be inserted in modern NICs for direct access to VMs. Similar functionality for IPv4 matching is already implemented in many server-class NICs.

in an AD with 100 million hosts, and CIDs for YouTube and the Web[6]. Even though for large organizations, the HID and SID tables can fit in DRAM, its cost might be prohibitive if all devices had such a large table.

XIA does not make these extreme designs possible today; they may require non-flat identifiers or inexact routing [47], or techniques that have not yet been developed. Instead, XIP's flexible addressing makes it possible to take advantage of them if they are successfully developed in the future.

6 Related Work

Substantial prior work has examined the benefits of network architectures tailored for specific principal types. We view this work as largely complementary to ours, and we have drawn upon it in the design of individual principal types. The set of relevant architectural work is too large to cite fully, but includes proposals for content and service-centric networks, such as CCN [26], DONA [29], TRIAD [23], Serval [20], and many others.

Extensibility through indirection: One approach used in prior work to support multiple principal types devises solutions that leverage indirection, such as through name resolution or via overlays. For example, the Layered Naming Architecture [12] resolves service and data identifiers to end-point identifiers (hosts) and end-point identifiers to IP addresses. *i3* [42] uses an overlay infrastructure that mediates sender-receiver communication to provide enhanced flexibility. Like XIA, these architectures improve support for mobility, anycast, and multicast, but at the cost of additional indirection. Of course, an advantage of these approaches that leverage indirection over XIA is their ease of deployment atop today's Internet. DONA eliminates the cost of indirection by forwarding a packet in the process of name resolution. Like DONA, XIA separates the name (intent) from its locations. However, XIA differs in two key aspects: 1) XIA makes translation from name to location as part of packet forwarding and combines it with principal-specific processing. 2) DONA relies on the network for correct translation from a name to its location, but XIA relies on the backwards-compatible paths provided by the application.

Extensibility through programmability has been pursued through many efforts such as active networks [43], aiming to ease the difficulty of enhancing already-deployed networks. The biggest drawback to such approaches is resource isolation and security. In contrast, XIA does not make it easier to program new functionality into existing routers—although an active networks approach could potentially be applied in tandem.

Architectures that evolve well have been a more recent focus. OPAE [21] shares our goals of supporting evolution and diversity, but their design focuses primarily on improved interfaces for inter-domain routing and applications, whereas XIA targets innovation and evolution of data plane functionality within or across domains. OPAE 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. Role-based architecture [14] promotes a non-layered design where a role provides a modularized functionality—similar to XIA's principal-specific processing. However, it is unclear how it allows incremental deployment of new functionality like XIA does.

Ratnasamy *et al.* propose deployable modifications to IP to enhance its evolvability [37]; but does not admit the expressiveness afforded by XIA. Others have argued that we should concede that IP (and HTTP) are here to stay, and simply evolve networks atop them [36]. However, this is not a solution in the long run; EvoArch [7] points out that merely pushing the narrow waist from layer 3 to layer 5 would result in yet another "ossified" layer.

Finally, *virtualizable networks* admit evolution by allowing many competing Internet instances to run concurrently on shared hardware [11, 39, 45]. Clark *et al.* present a compelling argument for the need to enable competition at an architectural level [17], 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 a single Internet instance, but virtualizable networks offer the potential for stronger isolation properties and to support farther-reaching architectural changes than XIA (e.g., such as moving to a fully circuit-switched network). Substantial research remains in moving these architectures closer to fruition and in comparing their strengths.

Borrowed foundations: *Self-certifying identifiers* were used in many systems [25, 32]. AIP [10] used self-certifying identifiers for both network and host addresses, as XIP does, to simplify network-level security mechanisms. Several content-based networking proposals, such as DONA [29], use them to ensure the authenticity of content. Serval [20] similarly names services based upon the hash of a public key. These works demonstrate the substantial power of these intrinsically secure identifiers, which XIA generalizes to an architectural requirement.

Addressing schemes: The flexibility of DAG-style addressing has been used elsewhere, notably Slick Packets [34]. Our addressing scheme uses this concept in a new way, to provide support for network evolution.

[6] We estimate the number of SIDs by assuming that each host uses up to 20 ephemeral SIDs at a time on average. For large objects, the forwarding table is 0.0125% of the content size assuming a chunk size similar to BitTorrent. For the Web, the average object size is (7.8KB) [15]. We then double the size assuming a 50% load factor hash table for storage.

7 Conclusion

XIA builds upon the TCP/IP stack's proven ability to accommodate technology evolution at higher and lower network layers by incorporating evolvability directly into the narrow waist of the network. XIA supports expressiveness, evolution and trustworthy operation through the use of an open-ended set of principal types, each imbued with intrinsic security. The centerpiece of our design, XIP, is a network-layer substrate that enables network innovation, and has the potential to support and amalgamate diverse sets of ideas from other clean-slate designs.

Substantial research remains to address issues such as crafting transport protocols that take advantage of content caching; devising a congestion-control mechanism that accommodates all principals; adapting or engineering suitable intra- and inter-domain routing protocols for HIDs and ADs; and incorporating trustworthy protocols that leverage intrinsic security. We view the large scope of future work as an architectural strength, showing that XIA enables a wealth of future innovations in routing, security, transport, and application design, without unduly sacrificing performance in the pursuit of flexibility.

Acknowledgments

We thank Sangjin Han for sharing his experience on router performance evaluation. We are also grateful to our shepherd Jon Crowcroft and anonymous reviewers for their feedback. This research was supported in part by the National Science Foundation under awards CNS-1040757, CNS-1040800, and CNS-1040801.

References

[1] Comcast press room - corporate overview. `http://www.comcast.com/corporate/about/pressroom/corporateoverview/corporateoverview.html`, 2011.

[2] Google: one million servers and counting. `http://www.pandia.com/sew/481-gartner.html`, 2007.

[3] KVM: Kernel based virtual machine, 2012. `http://www.linux-kvm.org/page/Main_Page`.

[4] MobilityFirst Future Internet Architecture Project. `http://mobilityfirst.winlab.rutgers.edu/`, 2010.

[5] IETF transparent interconnection of lots of links (TRILL) working group, 2012. `http://datatracker.ietf.org/wg/trill/charter/`.

[6] P. Agyapong and M. Sirbu. Economic incentives in content-centric networking: Implications for protocol design and public policy. In *Proc. Research Conference on Communications, Information and Internet Policy*, Sept. 2011.

[7] S. Akhshabi and C. Dovrolis. The evolution of layered protocol stacks leads to an hourglass-shaped architecture. In *Proc. ACM SIGCOMM*, Aug. 2011.

[8] A. Anand, F. Dogar, D. Han, B. Li, H. Lim, M. Machado, W. Wu, A. Akella, D. Andersen, J. Byers, S. Seshan, and P. Steenkiste. XIA: An architecture for an evolvable and trustworthy Internet. In *Proc. ACM Hotnets-X*, Nov. 2011.

[9] A. Anand, F. Dogar, D. Han, B. Li, H. Lim, M. Machado, W. Wu, A. Akella, D. Andersen, J. Byers, S. Seshan, and P. Steenkiste. XIA: An architecture for an evolvable and trustworthy Internet. Technical Report CMU-CS-11-100, Carnegie Mellon University, Feb. 2011.

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

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

[12] 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.

[13] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM SIGCOMM CCR*, 32:20–30, Jan. 2002.

[14] R. Braden, T. Faber, and M. Handley. From protocol stack to protocol heap: role-based architecture. *ACM SIGCOMM CCR*, 33, Jan. 2003.

[15] J. Charzinski. Traffic properties, client side cachability and CDN usage of popular web sites. In *Proc. MMB&DFT*, pages 136–150, 2010.

[16] G. Chatzopoulou, C. Sheng, and M. Faloutsos. A first step towards understanding popularity in Youtube. In *Proc. INFOCOM IEEE Conference on Computer Communications Workshops*, 2010.

[17] D. Clark, J. Wroclawski, K. Sollins, and B. Braden. Tussle in cyberspace: Defining tomorrow's Internet. In *Proc. ACM SIGCOMM*, Aug. 2002.

[18] M. de Kunder. The size of the World Wide Web. `http://www.worldwidewebsize.com/`, Jan. 2011.

[19] S. E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, Dec. 1991.

[20] M. J. Freedman, M. Arye, P. Gopalan, S. Y. Ko, E. Nordstrom, J. Rexford, and D. Shue. Serval: An end-host stack for service-centric networking. In *Proc. USENIX NSDI*, Apr. 2012.

[21] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Intelligent design enables architectural evolution. In *Proc. ACM Workshop on Hot Topics in Networks*, 2011.

[22] S. Govind, R. Govindarajan, and J. Kuri. Packet reordering in network processors. In *Proc. IEEE IPDPS 2007*, Mar. 2007.

[23] M. Gritter and D. R. Cheriton. TRIAD: A new next-generation Internet architecture. `http://www-dsg.stanford.edu/triad/`, July 2000.

[24] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated software router. In *Proc. ACM SIGCOMM*, Aug. 2010.

[25] HIP. Host Identity Protocol (HIP) Architecture. Interent Engineering Task Force, RFC 4423, May 2006.

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

[27] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A scalable ethernet architecture for large enterprises. In *Proc. ACM SIGCOMM*, Aug. 2008.

[28] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM ToCS*, 18(3):263–297, Aug. 2000.

[29] 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.

[30] J. Lilley, J. Yang, H. Balakrishnan, and S. Seshan. A unified header compression framework for low-bandwidth links. In *Proc. ACM Mobicom*, Aug. 2000.

[31] R. Ludwig and R. H. Katz. The Eifel algorithm: making TCP robust against spurious retransmissions. *ACM SIGCOMM CCR*, 30:30–36, Jan. 2000.

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

[33] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer2 data center network fabric. In *Proc. ACM SIGCOMM*, Aug. 2009.

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

[35] U. of Oregon. RouteViews. `http://www.routeviews.org/`, 2012.

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

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

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

[39] 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.

[40] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proc. ACM Mobicom*, pages 155–166, Aug. 2000.

[41] M. Soliman and G. Abozaid. Performance evaluation of a high throughput crypto coprocessor using VHDL. In *Proc. ICCES*, Dec. 2010.

[42] I. Stoica, D. Adkins, S. Zhaung, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. ACM SIGCOMM*, pages 73–86, Aug. 2002.

[43] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *ACM SIGCOMM CCR*, 26(2):5–18, Apr. 1996.

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

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

[46] W. Wu, P. Demar, and M. Crawford. Sorting reordered packets with interrupt coalescing. *Computer Networks*, 53:2646–2662, Oct. 2009.

[47] M. Yu, A. Fabrikant, and J. Rexford. BUFFALO: bloom filter forwarding architecture for large organizations. In *Proc. ACM CoNEXT*, 2009.

[48] S. Q. Zhuang, K. Lai, I. Stoica, R. H. Katz, and S. Shenker. Host mobility using an Internet indirection infrastructure. In *Proc. ACM MobiSys*, May 2003.