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 Andersen,
John Byers[†], Srinivasan Seshan, and Peter Steenkiste**

January 2011
CMU-CS-11-100

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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

This research was funded in part by the National Science Foundation under award number CNS-1040801.

Keywords: future Internet, network architecture, service, content, evolvability, intrinsic security

Abstract

Motivated by limitations in today's host-based IP network architecture, 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. 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 and protocols in the XIA data plane. We outline key design requirements relating to expressiveness, intrinsic security, and native support for multiple communicating principals, and then describe how we realize them in XIP, a new internetwork protocol. We demonstrate how rich addressing and forwarding semantics of XIP facilitate evolvability and flexibility, while keeping core network functions, notably packet processing, simple and efficient. Case studies demonstrate how XIA benefits both existing applications, and more novel ones, such as secure service migration. Finally, our evaluation of an XIA prototype demonstrates that XIA's advantages can be realized on today's hardware.

1 Introduction

The "narrow waist" model 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. What it does not facilitate, almost by design, is a clean, incremental adoption path for the network to support new capabilities, e.g., behaviors other than IP header-based forwarding that are better suited for certain usage models. Continuing improvements in storage and processing have made it possible to dramatically expand core network functionality, but the networking community has only been able to use these capabilities by deploying so-called "middleboxes" or "L7 switches" that reach through the IP headers into application-layer content. However, middleboxes are often viewed as either harmful, fragile (e.g., transparent proxy caches) or as innovation-inhibiting (e.g., firewalls).

To better support emerging usage models and new capabilities, prior work has argued that a future network should be *x*-centric for several values of *x*. Content-centric [17, 35] networks could better support Web, video, P2P, or other forms of content retrieval. Service-centric [31, 30] networks could provide powerful primitives such as service-level anycast. Even user-centric networks are a possibility [11]. We recognize that all of these arguments have merit, and embrace them in our design. Indeed, the central tenet of our *eXpressive Internet Architecture (XIA)* is that the network should be able to natively support all of these usage models if they prove beneficial, and in addition *be able to evolve* to support as-yet-unforeseen types of communication as demand materializes. While XIA has the intrinsic capability to evolve, we warn the reader up front that our goal is not to create a silver bullet that makes such evolution trivial. Instead, we set ourselves the narrower goal of designing a network that makes evolution *possible*.

In addition to the poor support for new types of communication, today's Internet is oft-criticized for being difficult to "secure": vulnerabilities range from network-level concerns such as preventing address spoofing, route hijacking, and DoS attacks to higher-level concerns such as DNS spoofing and even spam and phishing attacks. The difficulty arises because security was not a first-order design consideration for the original Internet. This has forced us to adopt spot fixes and band-aid solutions today. Unfortunately, these are ultimately ineffective and contribute further to the network's fragility. To avoid this predicament, another key tenet we follow is to build security into the core architecture as much as possible, without impacting the network's efficiency or the ability to evolve.

Thus, the goals of XIA are: preserve the strengths of today's existing architecture while substantially improving security, and building in the ability to evolve network functionality over time to natively support widely different types of use, including those unforeseen today.

To achieve these goals, we center our architecture on the abstraction of a *principal*: a named entity such as a host, a domain, a service, or a specific piece of content. In our principal-centric approach, a communicating application explicitly identifies the communicating principals (at a minimum: a source and a destination), and these principals' IDs form the basis for packet forwarding. As a concrete example, in XIA a web browser might specify the destination of a request as a named chunk of content (using our XIP protocol, a replacement for IP). Alternatively, it might specify the destination as a service capable of generating the content, or to a host known to store the content. A critical distinction from previous work is that new principal types can be defined at any time as the need arises. Moreover, to support evolvability and for maximum flexibility, *XIP packets can contain paths to multiple principals*: the above content request might contain many destinations in priority order. The network is then free to forward and process XIP packets based upon whichever of these principals it knows how to support, using opportunistic optimizations where possible.

We build security into XIA by requiring that *all* principals be *intrinsically secure* [2]: it should be possible for an entity to validate that it is communicating with the correct principal without needing access to external databases, information, or configuration. Content naming based upon the cryptographic hash of the content is one such intrinsically secure identifier (the receiver of the content can verify that its hash matches the desired name); so also is naming hosts based upon the hash of their public key. Intrinsically secure XIA identifiers directly prevent some security problems (e.g., address spoofing and route hijacking), and can help create effective defenses for others (e.g., network DDoS and cache poisoning attacks). While security innovations are not a focus of the paper, intrinsic security is a core architectural theme.

This paper presents the design, prototype implementation, and evaluation of XIA. We start by describing an example application scenario (§2) and use it to motivate the design requirements for XIA (§3). We then describe the XIP internetworking protocol, addressing format and XIP router per-hop processing. We highlight our design decisions in each that help integrate multiple principal types into the network via a consistent protocol and enable applications to flexibly use the network and leverage its functionality while keeping the architecture free from excess complexity (§4). We then describe plausible support for multiple common principals, including content, services, hosts and networks (§5), indicating the steps needed to add a new principal to XIA.

Through our design exercise, we find that XIA can make it much easier to accomplish a variety of important tasks, e.g., supporting continuous operation for mobile users; implementing on- or near-path content caching; and migrating running applications between computers. We illustrate these benefits through two case studies: how a "typical" Web browsing scenario can be enhanced, and how to enable seamless access to a service that supports secure process migration (§6). We describe a preliminary evaluation of our prototype, showing that XIA both simplifies application implementation and improves performance. We find that XIP packet processing is only marginally slower than IP, and that existing hardware can easily support XIP forwarding tables (§7).

While we cover key aspects of XIA in this paper, there are several relevant architectural issues that we do not address, such as transport protocols, network defenses, routing control planes, management support for each principal, as well as the economics and policy issues underlying support for evolution. Most have solutions in existing or proposed architectures that we can adopt; we discuss some of them in §8. We present outstanding challenges and conclude in §9.

2 Motivating Example

Many important applications on the Internet today already interact with multiple principals of different types. As one example, the process of fetching a web page involves three principals. The first step is an interaction with a *service*, such as `site.com`. This service identifies or creates the content that should be provided to the user, and the browser must then obtain the *content*. Today, this last step is conflated with communicating with the service, i.e. the server transfers the content to the specific *host* on which the browser is running.

Unfortunately, most existing and proposed network architectures offer native support for (and consequently, optimize around), a single type of principal (e.g., IP for hosts, or CCN [17] for content). Such a narrow focus requires that communication with other principal types be shoehorned into protocols for the primary type, which adds overhead and complexity (e.g., additional levels of indirection or lookup, sometimes involving construction of an overlay network). This additional complexity also makes securing the network more difficult, as it can often introduce new elements that must be trusted for the network to function correctly.

Two examples highlight these problems. In today's Internet, content retrieval is often optimized using content delivery networks (CDNs) which add complexity in the form of DNS redirection, URL rewriting, and caching heuristics, and require that both users and web sites trust the CDN to deliver correct, unmodified content. An alternative approach is the use of peer-to-peer networks, which require a combination of overlay networks, application-level support and circumvention of firewalls and NATs. Analogous problems arise when supporting host-to-host communication within purely content-based architectures. For example, CCN routers must "remember" the ingress point of a content request so that they can forward the response back toward the originating host [17], adding state proportional to the number of outstanding flows. Supporting traditional host-based applications (e.g., VoIP) over CCN is an even larger challenge, requiring substantial additional support [16].

These problems vanish if the network provides native support for multiple, intrinsically secure principal types. Such an architecture can allow rich interaction between principals – for example, a content router that lacks a "route" to a piece of content may revert to host-based communication to retrieve content from a known server. Moreover, it can simplify application development, can avoid performance overhead from indirection, and can enable late binding to hosts or services.

As an example, retrieving a webpage in XIA can be done as follows:

1. The browser finds the (intrinsically secure) identifier of the `site.com` service using mechanisms such as web search, bookmarks, secure DNS, etc. This identifier is the hash of a public key that can be used to verify communication with and content from the serving site.

2. The browser sends a request to the `site.com` service by sending a packet that uses the secure service identifier as the destination address. The network will deliver the packet. 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.

3. The `site.com` service returns a response addressed to the requesting host's secure host identifier with a list of intrinsically secure content identifiers for the chunks of data that make up the web page. The response is signed, allowing the browser to verify, using the service's public key, that the response was generated by the intended service.

4. The browser retrieves the content using a sequence of request packets that use the content identifiers as the destination address, expressing the primary intent of the application (to reach any source of the identified content). The network can 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. Since not all content will be cacheable, and not all routers are expected to maintain routes to arbitrary content IDs, 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 example leaves out many details, and alternate approaches are also possible. We elaborate on the XIA architecture in the next two sections and we then discuss a number of extensions of this example in §6.

3 XIA Design Requirements

We have structured the design of XIA around three key design requirements:

1. Users and applications must be able to express their intent. Moreover, the architecture must support a wide range of potential expressions of intent. Doing so may give the network significant flexibility for in-network optimizations to satisfy that intent by allowing the network to observe and act upon it directly. Enabling in-network, principal type specific mechanisms facilitates our goal of evolvability, below. Because not all routers are expected to operate upon all intents, communicating parties must be able to specify alternate action(s) if routers cannot operate upon the primary intent.

2. Principal types must be able to evolve. It must be possible to seamlessly introduce new principal types over time and modify functionality for existing principal types (as opposed to requiring a flag day to upgrade the network to support a new principal types). In addition, the complexity of adding support for a new principal type should be reasonably low. It must be possible to add principal types incrementally, e.g., at first adding end-to-end support, then providing partial network support, and then global network support.

3. Principal identifiers should be intrinsically secure. As discussed in the introduction, it should be possible for a communicating entity to validate that it is communicating with the correct principal. The semantics of intrinsic security can and will vary by principal type. 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—intrinsically secure XIA identifiers for host and content should be able to reflect these requirements, respectively. The architecture does not specify how these identifiers are obtained, and it should support many options for doing so (e.g., root of trust, social relations, etc), but these options should have well-defined semantics and interoperable interfaces.

In support of these requirements, we next present XIA's data plane, which can be viewed as two components that together support end-to-end communication. The first is the **eXpressive Internet**

Protocol (XIP), an internetwork layer protocol that defines the common addressing and header format and the per-hop processing that applies to *all* principals. We present this protocol in §4. XIP provides the basic machinery to dispatch packets to the second component of the XIA architecture: **principal type-specific support**, which can be highly customized based on the forwarding and intrinsic security requirements of a principal. The host principal type handler, for example, might use traditional Internet routing and forwarding techniques; a content principal handler might check a local cache before falling back to host routing; and a hypothetical multicast principal type might forward a packet on multiple outbound links. We show how principal type-specific support can be added for basic principal types in §5.

Applications communicate using the multiple principal types through well defined, principal type-specific interfaces. We develop bare-bones interface designs to depict what such interfaces *might* look like for particular principal types, but we emphasize that they serve primarily as examples to illustrate network evolution using XIA. As particular principal types become heavily used, we anticipate that more robust, standard interfaces would emerge over time. We use these example interfaces to illustrate how XIA makes it easy for applications to use multiple principal types, thereby enabling interesting use cases that are difficult to support today: in §6, we provide concrete examples in the context of Web and banking applications.

In the remainder of this report, we will sometimes use the term "principal" as a short for principal type, if the difference is clear from the context.

4 eXpressive Internet Protocol

A distinguishing feature of XIA is that it facilitates communication between a rich set of principals. We therefore partition the logic defining communication in XIA into two components. First, the principal-independent core XIP protocol, its address format, packet header, and associated packet processing logic, forms the basic building block of per-hop communication in XIA. A key design element of XIP is a flexible format for specifying multiple paths to a destination principal, allowing for "fallback" 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 have typed XIDs that refer to hosts, services, contents, and administrative domains. We refer to these as HIDs, SIDs, CIDs, and ADs, respectively.

In this section, we first elaborate on the definition of a principal type, and we then describe the key features of the XIP protocol. We discuss the APIs and per-hop processing for individual principal types in §5.

4.1 Principal Type Definition

For each *type* of principal, the designer of the principal type must define:

1. The semantics of communicating with a principal of that type.

2. A unique XID type.

3. A method for allocating XIDs and mapping these XIDs to intrinsic security properties of any communication.

4. Any principal-specific per-hop processing and routing of packets that must be coordinated or made consistent in a distributed fashion.

In combination, these four features define the *principal-specific support* for a new principal type.

The semantics (definition 1) help define the communication intent associated with a particular principal type (i.e., define what it means to send/receive a packet to/from an XID). 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 well expressed with the existing types. For example, neither the host nor administrative domain types (HIDs and ADs) may be well-suited to expressing a desire to communicate with all nodes in a geographic area. Later research could, therefore, define a "GeoCast" [24] XID type to meet this (hypothetical) need.

The second and third definitions are used to generate the addresses that the network operates upon. These intrinsically secure XIDs should be flat and globally unique, even if, for scalability, they are reached using hierarchical means. Therefore, XIDs should be generated in a distributed and collision-resistant way, ideally without the help of a central authority. 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 manner, e.g., using a hash of the principal's public key.

The fourth definition sets forth correctness requirements for in-network optimization for handling specific principals. 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. An example of this type of processing is in-network content caching. Other type-specific support requires more coordination. For example, a group communication principal type would require that router implementations agree upon routing or tree-construction protocols.

4.2 XIP addresses

XIA's first design goal—expression of intent through native support of multiple principals—implies that the components of an XIP address may have many types. The goal of evolution further means that these components may have types unsupported by current forwarding hardware. These requirements lead to the *fallback* path mechanism we describe below.

Similarly, while XIA uses several flat, cryptographically-imbued identifiers to meet our third goal of intrinsic security, scalable global flat routing is challenging [33, 5]. Our architectural goal is to achieve long-term flexibility. If those scaling challenges are overcome sufficiently for a particular XID type, implementations will be able to move to flat addressing if they desire. Otherwise, they can use traditional scoping and hierarchical approaches to achieve scaling. For example, it should be possible to route to a particular publisher to reach a particular content CID.

To meet the need for fallback and scoping, XIP addresses are therefore structured as a directed acyclic graph (DAG) with five properties:

Each node is an XID, each edge is the next hop in a path. Nodes may be of any XID type; the address structure therefore need not change to accommodate new principal types. To forward along a path, a router must be able to select a next-hop interface that will carry the packet towards the next XID in the DAG. Note that in contrast to today's CIDR IP addresses, which consist of a single identifier, XIP addresses typically include multiple identifiers.

Routing begins at the untyped entry node, which does not have an XID, and ends at any "sink" XID node with out-degree zero. The figures in §4.3 illustrate this. In the spirit of flexibility, XIA places no requirements on whether any XIDs on the path must be reachable; a router that cannot find a next-hop interface to *any* of the next XIDs on the current path will send an ICMP-like unreachable error to the source. We anticipate that such issues would be addressed through convention or standards.

The address is a single connected component. Every component has exactly one entry node, and at least one sink.

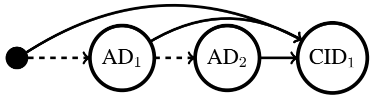
Outgoing edges are tried in the order they are listed. If the router cannot forward using the first edge, it tries the next edge, iterating this process until finding the next hop, or running out of edges, implying that no sink is reachable.

Bounded out-degree. Bounding the out-degree of a node limits the processing required at each hop and ensures that the individual entries in the DAG are fixed-length to facilitate fast processing.

4.3 Addressing style examples

XIP addresses provide considerable flexibility; in this subsection, we present six (non-exhaustive) "styles" of how they might be used to achieve different goals.[1] In these illustrations, the entry node is the leftmost node, and the rightmost node(s) are the sink(s). If multiple edges leave a node, the topmost has the highest priority.

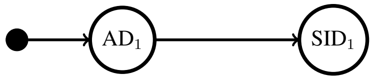
Shortcut routing. In the first example, the destination address encodes the final intent of the session, and a path that can be used to reach that intent. The first edge from every node is to the final node:



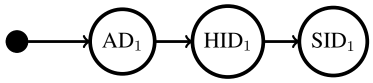
granting permission to use, e.g., a local replica of that intent. As a concrete example, this style would permit a content-caching router to directly reply to a CID query without forwarding the query to the origin. Because intrinsically secure XIDs allow easy detection of unauthorized responses to the final intent, we anticipate that this would be a common style of addressing in XIA.

Binding. Some communication sessions must be bound to a particular source and destination. An example is legacy HTTP communication: While the *first* packet could go to any node capable of serving the SID, subsequent TCP packets must go to the same host. The first packet could be destined to

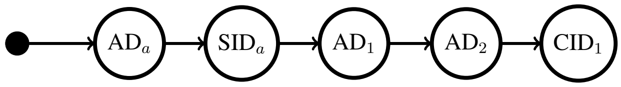
[1] In these examples, we omit a discussion of how to authorize sources to use the paths they specify, e.g., as with source routes. Our goal is to explore the flexibility of the addressing, but implementations that permit some types of routes would require additional authorization mechanisms.



A router inside AD$_1$ could then route the request to a particular node that provides SID$_1$. The node would reply with a bound source address:

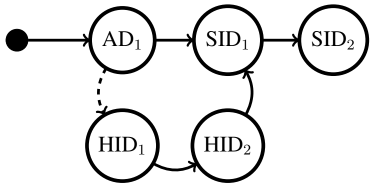


Source Routing can be accomplished using a linear chain of addresses. The example below, inspired by scenarios from DOA [37], routes packets through a service running in a third party domain, AD$_a$:SID$_a$:



One concrete case, in use today, occurs when the owner of AD$_2$ subscribes to VeriSign's DoS mitigation filter [36] that "scrubs" packets before they are sent to AD$_2$. (For clarity we show the true destination, but in practice, the final address would be modified so that only the filter can send to AD$_2$.) Today, VeriSign provides this service using BGP, but DAG addresses provide a more general mechanism for source routing and delegation [5].

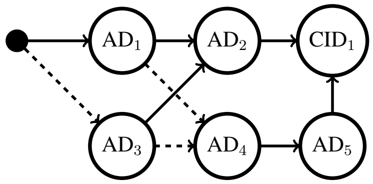
Infrastructure Evolution. This addressing style provides support for fallbacks. For example, AD$_1$ could incrementally deploy a middlebox as a service, SID$_1$. When a packet arrives at a router in AD$_1$ that does not know how to handle SID$_1$, it falls back to the second edge containing a reliable path to SID$_1$:



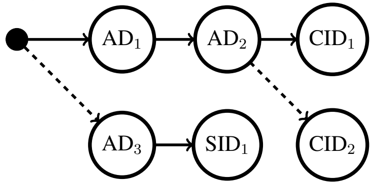
SID$_1$ could also be a new type of principal.

DAG addresses can also support more speculative styles:

Multiple Paths. Having multiple paths to a sink node allows routers to choose among possible paths if the preferred one fails. An address in this style can arise naturally in datacenters or multi-homed smartphones (e.g., WiFi and 4G).



Multiple Sinks. An address with more than one sink node could represent a sophisticated form of "anycast" that could reach multiple types.



	Ver	NxtHdr	PayLen	HopLimit	ND	NS	LN	
0:	XidType			ID				P[N]
. . .	. . .			. . .				. . .
ND-1:	XidType			ID				P[N]
0:	XidType			ID				P[N]
. . .	. . .			. . .				. . .
NS-1:	XidType			ID				P[N]

Figure 1: XIP packet header.

Whether or not multiple sinks is worth supporting requires further research. Our prototype does not yet do so; using multiple sinks in conjunction with shortcut routing imposes substantial overhead because the router must examine all sinks.

4.4 XIP header and per-hop processing

DAG-based addressing is the central mechanism allowing XIA to meet its goals of flexibility and evolvability. This flexibility, however, must not come at excess cost to efficiency and simplicity of the network core. In this section, we detail a possible design for the XIP header and per-hop forwarding behavior. We show in §7 that this design satisfies both the demand for flexibility and for efficiency.

4.4.1 Header Format

Traditional fields: The XIP header (Figure 1) begins with single byte fields for header version, next header, payload length (two bytes), and hop limit, shown in the top row.

Variable-length DAG fields: To support DAG addresses, the single byte fields NS and ND indicate the size of the source address DAG and destination address DAG, respectively, and LN points to the previously-visited node in the DAG. Following the fixed-length portion of the header are the destination and source address DAGs stored in topological order as adjacency lists.

Each element in an adjacency list requires 28 bytes: 4 bytes to identify the principal type (XidType), 20 bytes for the principal's ID, and an array of four 1-byte pointers that represent the out-edges from the node. By topological ordering, the final node in each address (i.e. the one at offset ND or NS) is guaranteed to be a sink node. By definition, each sink has out-degree of zero; thus (to save space) the edge array of the final sink node stores the edges of the *entry node* of the address. Other sinks explicitly store zero out-edges.

Nodes in the DAG can have out degree at most four. This choice balances: (a) overall header size, per-hop processing cost and having 32-bit memory alignment of key fields of the header to simplify router processing; and (b) retaining the DAG's flexibility to support many styles of addressing.

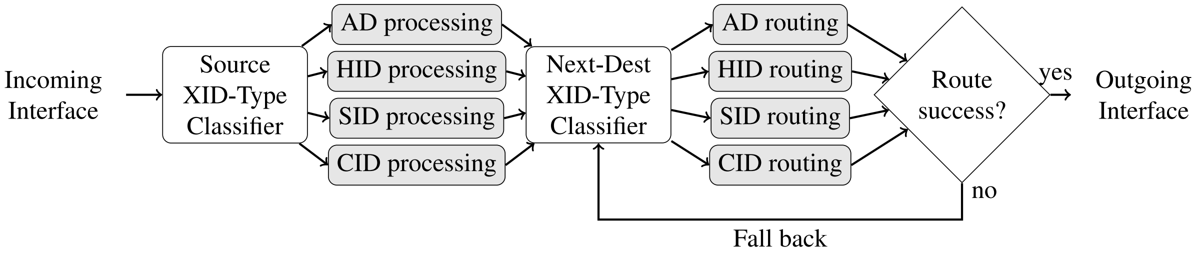


Figure 2: Simplified diagram of an XIP router.

We expect that 32-bit XidType fields are (more than) sufficient to handle what users will need for future principal types and versions. Using a large namespace for types also allows delegating ranges of values to many "principal type registries" who, in turn, grant unique type IDs.

4.4.2 Per-hop Forwarding

Figure 2 depicts a simplified view of 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 principal types. Our design isolates principal-type specific logic so that there is minimal interdependence between these code paths; logic to support new principals (see implementation details in §7.1) can readily be added.

When a packet arrives, the router first invokes source-XID specific processing based upon the XID type of the sink node of the source address DAG. Such processing may be necessary for certain packets, such as for a reply packet to a CID request. For example, the CID processing element on a router that sees a reply packet from AD_1:HID_1:CID_1 only needs to look at CID_1 to decide on caching the content, or not. If the source XID-Type classifier has no processing element associated to a given XID type, the packet is handed immediately to the next classifier to forward the packet. Note that it may be possible to perform Source and Next-Dest processing in parallel.

After source-specific processing, the router's Next-Dest classifier determines the appropriate forwarding action. This classifier iteratively examines the principal types of the outbound edges of the last-visited node (field LN in the header) in order. If it can forward the packet using a given edge, it does so. If no outgoing edge of the last-visited node can be used for forwarding, the destination is considered unreachable and an error is generated.

To decide if it can forward along a given edge, the router invokes a principal-specific component that examines the XID (fields XidType and ID) of the next node using a principal-specific forwarding mechanism; principals can thus use customized forwarding logic ranging from simple route lookups to packet replication or diversion. To facilitate debugging and tracing, the router sets the most significant bit of the entry of the edge entry used to forward the packet.

Any node in an XIP address DAG can have any principal type. To provide this flexibility while keeping the router simple, the destination classifier in Figure 2 ensures that a packet with no known principal types as next hops will fail. This places on applications (or their network stacks) the responsibility for composing addresses properly to provide fallback edges around new XIDs so that they can be routed using well-supported XIDs such as ADs and HIDs, similar to those shown

(a) Network and Host

Function Definition	Description
bind(socket, keypair)	Bind to a HID or AD. Keypair specifies the public/private associated with host/network
recv(socket, message)	Receive message for host or network.
send(socket, dest, message)	Send the message to host or network.

(b) Service

Function Definition	Description
bind(socket, keypair)	Binds socket to a SID Keypair specifies the public/private used for the service (optional)
recv(socket, message)	Receive message from peer
send(socket, dest, message)	Send the message to peer

(c) Content

Function Definition	Description
getContent(socket, addr, buffer)	Retrieves the content specified by addr from network; addr contains CID and possibly a fallback
putContent(socket, content)	Registers the content as available

Table 1: API for each principal.

in the "Infrastructure Evolution" addressing style in the previous section.

5 Principals

This section elaborates on plausible support for the most common principals—networks, hosts, services and content—reflecting the principal type definition steps from §4.1. Our descriptions are meant to serve as examples; as principals are heavily used, different interfaces and per-hop behaviors than those we identify may become standardized. We also highlight the advantages that principal-specific support in XIA offers to different entities, including applications, users and ISPs.

5.1 Networks and Hosts

Semantics The network and host principals represent autonomous routing domains and hosts that attach to the network. Hosts have a single identifier that is constant regardless of the interface they use or the network they attach to.

In today's Internet, the endpoints of communication are application processes. Sending packets to an IP destination does not make much sense without the associated protocol and port numbers. In XIA, this notion of processes as the true destination is explicit; networks and hosts identifiers primarily provide control over routing. As a result, a typical TCP-style communication would be achieved by sending a packet to an AD:HID:SID. The AD and HID provide a form of loose source routing, where the AD forwards the packet to the host, and the host "forwards" it to the appropriate service or process (SID).

There are two exceptions to this rule, where an extension header can be added to a packet destined to an AD or HID for: 1) XIP control messages; 2) Accessing a host or AD-based service lookup function. Service lookup is similar to today's portmap or DNS SRV records. It allows other entities to find the SID associated with a particular service on a particular host or domain. For example, an entity could query for the SID associated with the SSH service on a particular host, or query for the SID associated with DNS service for a particular domain.

XIA provides a simple socket send and receive API (Table 1) for AD and HID principals in order to support the above interactions. The bind API is used to announce the desire to process messages for a particular HID or AD (derived from the keypair) to the network's and host's routing system. This ensures that the network and host principals can always be used for routing. Note it is also desirable to register a host or AD name in a naming service, so users can retrieve AD and HID identifiers based on human readable names. System dependent libraries provide other convenience functions (e.g., getServiceByName() for service lookup) on top of the socket API.

Addressing and Intrinsic Security The format of the AD and HID are similar to those of the network and host identifiers used in AIP [2]. ADs and HIDs are generated by hashing the public key of an autonomous domain or a host, binding the key to the address and making it self-certifying.

Per-Hop Processing and Routing While we believe that XIA can and should take advantage of future innovations in global, scalable routing, we can also use today's techniques to achieve this goal: Hierarchical routing based upon autonomous domains. For now, therefore, every XIA router is required to support AD and HID routing. The self-certifying property of the identifiers enables various security features to be enforced in the network to deal with problems such as source spoofing, route hijacking and denial-of-service as shown by AIP [2].

5.2 Services

Semantics Services represent an application service running on one or more hosts within the network. Examples range from the SSH daemon running on a host to Web servers, Akamai's global content distribution service or Google's search service. Each service defines its own protocols for interaction. For example, a Web server will use HTTP to exchange messages between a client and the service.

XIA provides a simple interface (Table 1) to send and receive datagrams to and from a service. The main difference in this API from the one for hosts and networks is in the behavior of `bind()`. For services, `bind()` adds the SID (derived from the keypair) to the host's routing table and service lookup interface (and optionally to the network's routing tables). This ensure that packets destined to this SID at the associated host (HID) are sent to the application. The service should also register the SID with a registry so applications can search for the service using a human readable service name. If an application calls `bind()` with an empty keypair, the system generates an ephemeral SID (by generating a random public/private keypair), which is never announced to either network routing or service lookup. This distinction is similar to that of well-known and ephemeral port numbers today.

Addressing and Intrinsic Security The XID for a service, called an SID, is a 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. Any entity that is 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 any encryption that is desired.

Per-Hop Processing and Routing Because each service defines its own protocols for interaction, it is typically difficult for routers to perform any specialized processing for all services. As a result, we envision that SID-specific processing would typically be performed at endpoints, while routers typically perform default processing, routing, and forwarding. If the motivation is strong for in-network processing of a particular SID, it should be given its own principal type that more directly expresses its intent, and this principal can then be operated upon.

5.3 Content

Semantics The content principal allows users to express their intent to retrieve content without regard to location. Sending a request packet to a content ID initiates retrieval of the content from either a host or a network content cache. A transport protocol handles data delivery, including reliability and congestion control. XIA's content principal support is transport protocol agnostic. A content specific extension header can be used to provide information about the request, such as the preferred transport protocol or whether the request is for a portion of the content.

Table 1 shows the API that supports the semantics for content principals. When `getContent()` is called, the node constructs an address using the CID and Fallback addresses provided by the application. It then sends out a CID request packet to the address, which initiates a transfer in response. When the content is received, it is returned back to the user. The underlying socket API allows specifying the transport layer to be used for the CID transfer.

The `putContent()` call allows a host to register the availability of content to requests from other entities. When `putContent()` is called by the application, the transport layer caches the content, puts the CID into the routing table of its host, and immediately returns. Adding the routing entry for the CID allows subsequent CID requests to be forwarded up the stack when a CID request packet reaches the host. CID request packets contain a content-specific extension header which allows the requester to specify the details such as preferred transport protocol type, or partial content request. The CID response packets also must contain a content-specific extension header that specifies the length of the content, offset and length of the part that is contained in the packet. This enables reconstruction of content in intermediate routers when CID response packets contain transport layer information relevant only to the endpoints of the communication. We evaluate caching in XIA in §7.2.

In some cases, the owner of a CID may not wish that content be cached by the network. To support this, the content-specific extension header in a CID response includes a “do not cache” directive for the routers located outside the owner's AD. A similar field could be used by a CID requester to specify that the network not serve cached content – this could be useful in situations where the content received from a router is garbled either due to on-path errors or a failure at the router cache. The requester can instead retrieve content from the original source or a designated fallback.

Addressing and Intrinsic Security Content identifiers (CIDs) are the cryptographic hash (e.g., SHA-1, RIPEMD-160) of the associated content. The self-certifying nature of this identifier allows any network element to verify that the content retrieved matches its content identifier. In practice, large content may need to be chunked into smaller pieces for efficient transfer [28]. The CID principal itself is agnostic to chunking schemes (each chunk has its own CID). Our initial view is that chunking should occur at the application, to be able to take advantage of application-specific knowledge such as headers, metadata, or application-level boundaries. Within this, however, we hope that de facto standards would emerge over time for general chunking (e.g., using Rabin fingerprinting based chunking [23]) and for specific formats such as MP3s or movies.

Per-Hop Processing and Routing Native support of CIDs could take two likely forms: first, routing based upon CIDs, and second, caching of content. Because networks may have large numbers of CIDs, directly routing on only the CID is a challenge for future research, but an interim step seems likely using shortcut routing, in which routers might keep a local table that maps a subset of CIDs to nearby sources of those CIDs, while still using the full path in the event of a cache miss.

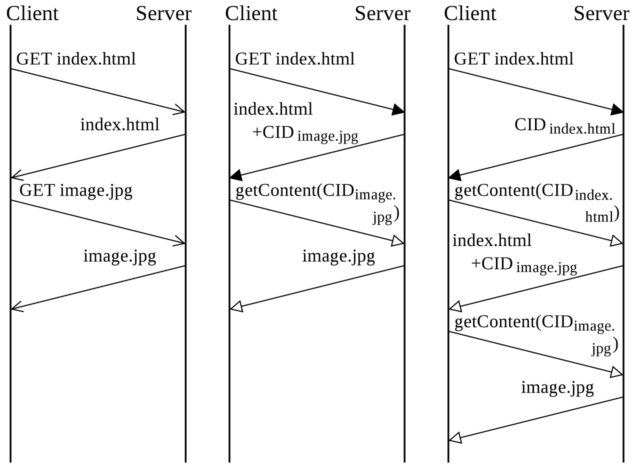
The second native support for CIDs enables routers to optimize content transfers. For example, routers can cache and directly serve content, either across users as in a traditional content cache, or within a transfer, to reduce end-to-end retransmission delays in the event of packet loss. Such caching could occur either locally at each router, or using various forms of network-wide cache coordination [1].

6 Networking with Multiple Principals

This section first revisits the web browsing example presented in §2 to show how XIA's multiple principals can support legacy applications, but more importantly, to highlight how this *benefits* applications. It then presents a richer example illustrating how XIA can easily facilitate more complex goals such as process migration and user mobility, things which are quite poorly supported today.

6.1 Web Browsing in XIA

In today's Internet, retrieving a simple web page with an embedded image requires two HTTP requests: one contacts the "web site" (a service) to retrieve the main page, and a second retrieves the embedded content image. In XIA, these two steps can be implemented in a variety of ways



(a) Normal HTTP (b) XIA mode 1 (c) XIA mode 2

Figure 3: Timeline diagrams of today's Web transfer via HTTP vs. XIA modes. → denotes an IP packet, a solid arrowhead denotes a SID (XIP) packet, and a hollow arrowhead denotes a CID (XIP) packet.

depending on how caching is used, how users bootstrap the acquisition of trusted XIDs, and the degree of support for XIA principals in the network. We initially assume the client has access to a trusted name resolution service such as secure DNS to translate a human-readable service name into an SID, and that all XIA principals are extensively supported in the network. We will relax these assumptions later.

Content download To retrieve the main page (e.g., index.html), the client will use the name service to translate `site.com` to an SID, and will then send a request to the service using the SID_{site} in the destination address. The network will deliver the request to an instance of the `site.com` service, possibly using opportunistic optimization (more below). As shown in Figure 3, the server has several options for responding. The instance can respond with the contents of the requested page (panel (b)). Alternatively, the service could provide $CID_{index.html}$, which the client could retrieve from any replica using the content principal (panel (c)). Although this incurs an additional RTT, it allows responses to be served from replicas or caches.

Likewise, XIA allows several options to retrieve the embedded image, e.g., using the principal types CID and SID, a content name (e.g., a URL), or a combination of the above. The service owner can select a suitable option, based, for example, on the image size. A client should use a site-provided CID or SID to retrieve the content only if she trusts the site to provide a valid XID (the client cannot check validity of the XIDs due to their cryptographic nature). A human readable name must be used otherwise. Note that for dynamically generated content, only SIDs or some other principal that reaches the HTTP server would be practical.

The choice of principal to use for retrieving the object is a tradeoff between efficiency arising from in-network optimization versus the degree of control that the user grants to the network to satisfy the request, resulting in different control points [8]. When the user uses a CID, she allows the network to serve the content from *anywhere*. The use of a SID potentially also allows the network to use one of a number of service instances, but options may be limited since service replication is typically more complex than caching. Finally, the use of an HID:SID limits the network to deliver the request to a particular device. Another consideration is that the choice of principal also affects the user's privacy, since different principals expose different types of information in the network header.

Evolving support for XIA principals: ISPs might wish to deploy support for CID-based content caches to reduce transit costs from other ISPs. Similarly, a service provider could also pay third-party providers to replicate their service worldwide, and have tier-1 ISPs route SID-based service requests to the nearest replica. But how do we support the above communication is there if only partial support for services or content in the network? In such a setting, users can still express their intent with these XID types, but they should also use appropriate fallback mechanisms.

Let us look at what the destination address may look like for some of the requests for this example. Suppose `site.com` is hosted on host HID_{site} in domain AD_{site}, and the client has obtained SID_{site} for the service. Then for fetching index.html, the destination address of her request would have SID_{site} as the primary intent, and fallback as $AD_{site} : HID_{site} : SID_{site}$. If an intermediate router supports SID_{site}, the request will be routed to the nearest service instance. Otherwise, the request will be routed towards $AD_{site} : HID_{site}$. This ensures that the intent for the service will be met, while also allowing the application to leverage benefits of network optimization for SID_{site} where available.

To support fallback (assuming XIA mode 2 in Figure 3) the client would use $CID_{index.html}$ as the primary intent in the destination address, then $SID_{site} : CID_{index.html}$ (if known) and then $AD_{site} : HID_{site} : CID_{index.html}$ as its fallbacks. If the intermediate router cannot forward $CID_{index.html}$, the first fallback, SID_{site}, will be tried. If the router also cannot support SID_{site}, the request will be forwarded towards $AD_{site} : HID_{site}$. XIA thus enables reachability even with limited support for principals, and allows the network to leverage the CID- and SID-specific benefits of in-network optimization.

Flexible trust management. The above example provides a strong "chain of trust" for obtaining content once the browser knows what SID it should contact: the SID is used to validate the content CIDs, which can validate the received content. However, reaching this trusted chain requires bridging the gap from semantically meaningful identifiers ("nytimes.com") to self-certifying SIDs or CIDs. By creating a foundation by which the rest of a network transaction can be completed securely, XIA's self-certifying principals can act as a "narrow waist of trust" in which multiple mechanisms, such as secure DNS, trusted search providers, web-of-trust/PGP models, or physical exchange could all serve as the trust root for subsequent communication. We plan to explore of the design of an API for this flexible trust management in future work.

6.2 Running A Service on XIA

In this section, we look at how services interact with applications using service and content principals through a lifecycle of an on-line banking service. We show how the service and users benefit from XIA's intrinsic security and flexible binding as well as the flexibility of using multiple principals.

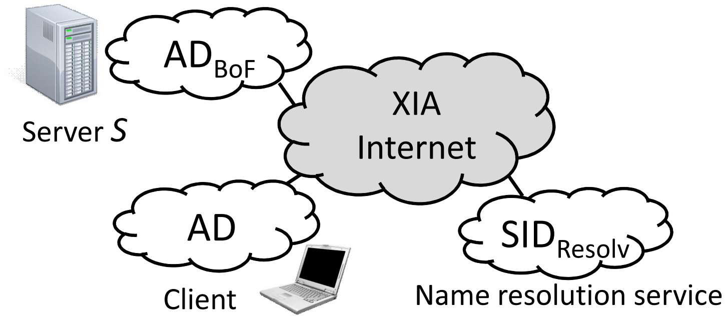


Figure 4: Bank of the Future example scenario.

In Figure 4, Bank of the Future (BoF) is providing a secure on-line banking service hosted at a large data center using the XIA Internet. We start with details of how BoF's service is published, so that its SID becomes known to clients. Then we discuss how binding between a client and the service is realized. Content transfer uses mechanisms presented in §6.1, so we focus on XIA's support for advanced features like process migration and client mobility. Our description focuses on a banking interaction between a BoF client C (HID_C) and a particular on-line banking server process P_S running on a server S whose HID is HID_S We also assume the service has a public key that hashes to a service ID, SID_{BoF}.

Publishing the Service When process P starts on the server, it creates a SID socket and binds to SID_{BoF} by passing its public/private key pair using the `bind()` API call. This adds SID_{BoF} to S's routing table, and depending on the route propagation policy, the route to SID_{BoF} is advertised to the network. In this case, the routing entry is only propagated within its administrative domain, AD_{BoF}. To make the service readily accessible, the service must also associate a human readable service name (e.g., "Bank of the Future On-line") to SID_{BoF}. The service publishes this association through a name resolution service (SID_{Resolv}).

Name resolution The client can resolve the service ID from human-readable names using a name resolution service. As a bootstrapping step, the SID of the name resolution service is required. This can be obtained using the AD or HID's local service lookup. For example, the client sends a packet with destination address set to its own AD with an AD extension header that specifies that it is a service lookup request for a name resolution service. When a router gets this packet, it responds with the SID of the name resolution service.

Binding The client now connects to the service. It sends out a packet destined to $AD_{BoF} : SID_{BoF}$ using the socket API. The source address is AD:HID_{client}:SID_{client}, where AD is the AD of the client, HID_{client} is the HID of the host that that is running the client process, and SID_{client} is the ephemeral SID automatically generated by `connect()`. The source address will be used to construct a return packet to the client.

The packet is routed to AD_{BoF}, and then to S, one of the servers that serves SID_{BoF}. After the initial exchange, both parties agree on a symmetric key. This further means that state specific to the communication between two processes is created. Then the client has to send data specifically to process P running at S, not any other server that provides the banking service. After the server binds the service ID to the location of the service, $AD_{BoF} : HID_S$, then communication may continue between $AD_{BoF} : HID_S : SID_{BoF}$ and AD:HID_{client}:SID_{client}.

Content Transfer When a user requests particular content, the server can either transfer the content directly using service principal or provide a content ID for the client to fetch it from the network, as in the Web example (§6.1).

Process Migration The initial binding of the banking service running on process P to HID_S can be changed when the server process migrates to another machine, for example, as part of load balancing. Suppose this process migrates to a server with host ID $= HID_{NewServer}$. With appropriate OS support for process migration, a route to SID_{BoF} is added on the new host's routing table and a redirection entry replaces the routing table entry on HID_S. After migration, the source address in subsequent packets from the service is changed to $AD_{BoF} : HID_{NewServer} : SID_{BoF}$. Notification of the binding change propagates to the client via a packet with an SID extension header containing a 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.

Client Mobility HID's association with an AD is also not permanent. A client may move and be attached to another AD, AD_{new}. The new source address of the client will then be AD_{new}:HID_{client}:SID_{client}. When a new packet arrives at the server, the server will update the client's address.

Intrinsic security Even though locations of both the server and the client have changed, the two SID end-points –and therefore their cryptographic verifiability– remain the same. Both the server and the client can verify that they are talking to the service whose public key hashes to SID_{BoF}.

7 Evaluation

In this section, we evaluate the following benefits and costs of XIA in the context of our preliminary XIA prototype:
(i) Benefits: How does XIA benefit application design and performance? In §7.2, we show that that using multiple principals can simplify application design, and that applications can benefit from the principal-specific in-network optimizations allowed by the architecture.
(ii) Costs: How does XIA affect routers packet processing costs and forwarding table storage requirements? In §7.3, we show that the baseline packet processing cost of an XIA router is only marginally higher than to that of IPv4. We also show that existing memory and storage technologies can be used to maintain global HID, SID, and even CID forwarding tables.

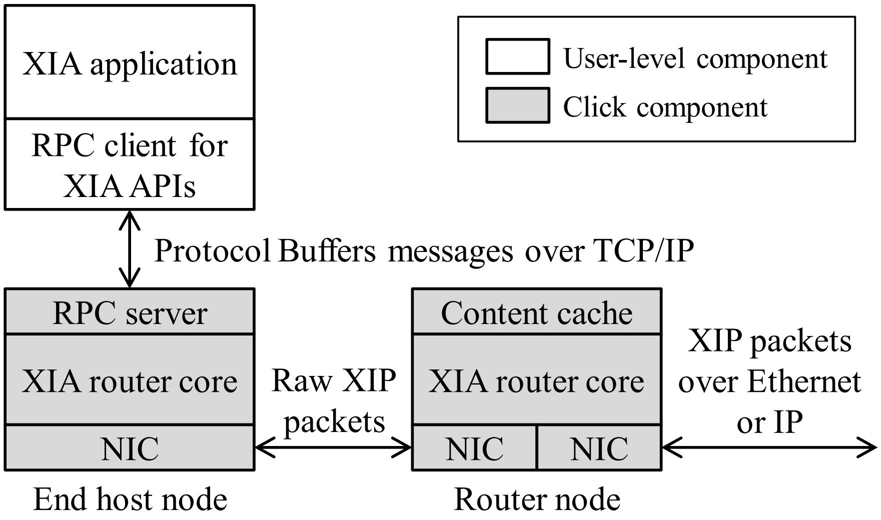


Figure 5: Block diagram of our XIA prototype.

7.1 Prototype Implementation

To address these questions, we prototyped XIA using the Click modular router [19] as depicted in Figure 5. XIA-enabled end hosts (left) implement user-level applications, which invoke Click-based XIA functions over a TCP connection, with API calls encapsulated as Protocol Buffers [13] messages. The XIA host stack is implemented as a Click module consisting of an RPC server, the XIA router core, and a configurable NIC. XIA routers (right) implement NICs, the XIA router core, as well as a content cache.

The prototype can run either across the network, encapsulating XIP packets within IP or Ethernet packets, or within a single Click process, in which case the hosts and routers communicate using raw XIP packets. The extensible XIA router core currently performs routing, forwarding, and per-hop processing for ADs, HIDs, CIDs, and SIDs.

7.2 Surfing the Web with XIA

We have implemented the simple web browsing scenario from §6.1, allowing an unmodified web browser to retrieve web pages using a combination of service and content principals. XIA's CID support enables router-based caching to substantially improve page download time, and that fallback support helps avoid extra RTTs by avoiding end-to-end retries when, e.g., such CID caching is only partially deployed.

Implementation: Native web browsers communicate with our client-side proxy, which in turn uses XIA to communicate with a web server. We have implemented a simple XIA-based web server that serves SID requests for web pages. In our implementation, URLs are modified to include both a human readable name and the XIA SID or CID.

Browsers request these modified URLs; the proxy parses the relevant XIA information (e.g., SID, CID) and generates an appropriate request. The web server controls decisions such as whether an SID request for index.html should return the content itself or its corresponding CID, as described

Mode	Time (ms)
XIA mode 1 without caching	61.3
XIA mode 2 without caching	91.7
XIA mode 1 with caching	41.0
XIA mode 2 with caching	51.4
XIA mode 1 without fallback	181.6
XIA mode 2 without fallback	332.4

Table 2: Webpage retrieval time in various scenarios.

in §6.1. Once the proxy receives the content, it serves it to the browser. Implementing the proxy and server was simple, requiring only 305 SLOC of Python code.

Performance: We measured the time it takes to get the full page with an embedded image using two retrieval modes from Figure 4(a). In mode 1, the server sends the content of index.html in response to an SID request. In mode 2, it sends back the CID of index.html. In both cases, the image is fetched using its CID. We conducted both transfers twice, the first time with a cold CID cache and the second time with a warm cache.

We measured transfer times by emulating a simple topology in Click: The client host, two routers, and the web server host, connected in a chain with three links with 5ms latency and 1Gbit/s bandwidth. The router closer to the client caches data based on the CIDs of webpage and the embedded image. The index.html and image files require 50 bytes and 14KB respectively.

The performance of XIA mode 1 without caching is similar to what we would get in the current Internet (Table 2). This mode incurs the same number of RTTs to fetch the page as an IPv4-based browser would. With caching in mode 1, the total page download time improves by 33%, showing the benefits of XIA content caching. Because the page is very small, mode 2 performance, which transfers the page by CID and therefore requires an extra RTT, is worse than mode 1, but when cached still outperforms today's retrieval.

Finally, the results highlight the benefit of in-network support for fallbacks. Without such support, if the original intent is not satisfied by the network, the client would have to both detect the failure (using timeouts or explicit failure notifications from the network), and retry end-to-end using a new message to the fallback intent. The final two lines in Table 2 show the high cost of using timeouts instead of fallbacks for CID retrieval using a timeout of 4×RTT before transmitting a SID request to the web server.

7.3 Router Performance

To understand the relative costs of XIP processing compared to conventional IPv4 processing, we first compare our prototype to Click's IPv4 forwarding speeds. We then explore storage requirement on routers for per-principal forwarding tables. Our prototype consists of ∼4400 SLOC of C/C++ excluding the original Click code.

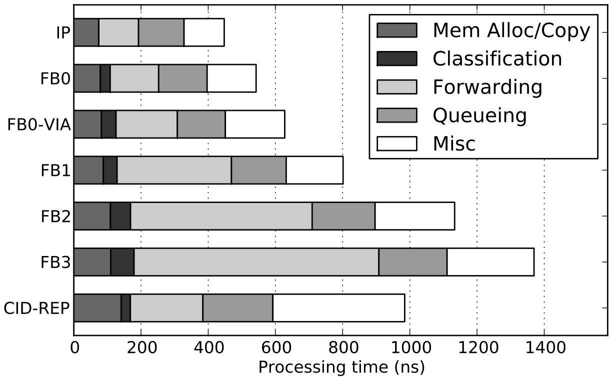


Figure 6: Performance breakdown of in-memory packet processing on our XIA prototype. The forwarding table has 351 K entries, and each packet has a 64-byte payload.

7.3.1 Packet processing costs

We compare forwarding rates using a relatively modest 2.83 GHz Core2 Quad Q9550 processor with 4GB DDR SDRAM and 12MB of L2 cache. Our prototype uses only one core at any time. Using `perf` [26], we obtain a detailed performance breakdown through sampling-based profiling.

Evaluation scenarios: The `IP` baseline refers to IPv4 packet processing consisting of route lookup, TTL update, and fragmentation checking without IP option header processing. `FB`*n* refers to XIP packet processing in which the XIA router evaluates exactly *n* fallbacks for ADs and then forwards based on the $(n + 1)$-th route lookup. In `FB0-VIA`, the router additionally adjusts the last-visited node field in the packet header to point to the following node in the DAG; other scenarios keep the last-visited node constant, reflecting the case in which the DAG node has not yet been reached. Finally, in `CID-REP`, the router also sends a copy of the CID response packet to the local content cache.

To produce the bar chart in Figure 6, we used a forwarding table of 351K based on a RIB snapshot obtained from the Route Views Archive Project [25] generated on Jan 1, 2011 at 0:00 UTC, resulting in a forwarding table containing 351K entries. IP uses this table directly; XIA pessimistically uses the same number of entries for the AD forwarding table by mapping each CIDR block to an AD.

The experiments execute within a single Click process to omit NIC overhead and emphasize the processing overhead differences between XIP and IP. Our results are averaged over a total of 10 runs that forward 100M packets each to a random (working) destination address.

Baseline XIP processing (`FB0`) is only 21% slower than IPv4 (`IP`). The overhead primarily results from the larger size of an AD identifier (20 bytes) than an IPv4 address (4 bytes), allowing the IPv4 forwarding table to fit in L2 cache where the AD forwarding table does not. Additional processing in `FB0-VIA` adds another 16% because a second forwarding table lookup is needed after the `LN` field update. More fallback options slow processing. The most complex case, (`FB3`), is 2.5× slower than the baseline (`FB0`). As expected, much of the variability in processing time is due to forwarding lookups in principal-specific forwarding tables.

	Forwarding Table	Public Key Store
HIDs (100M)	6.25 GB	50 GB
SIDs (2 Billion)	125 GB	1 TB

Table 3: Forwarding table size and public key store size of an AD with 100 million hosts.

We expect that the difference between IPv4 and XIP to be even smaller in practice. Our experiments deliberately omit overhead such as NIC I/O and link-layer encapsulation to focus on packet processing; including these fixed costs for both protocols would shrink the relative performance difference. Finally, while we tried to make the prototype's performance reasonable, we did not heavily optimize it; techniques such as processing fallback routes in parallel could substantially reduce the latency of the multiple fallback packets.

The overall low cost of XIP packet processing compared to IP processing suggests that XIP's forwarding logic could be implemented efficiently in high speed routers. Of course, packet forwarding in today' high-speed routers is not based on general-purpose processors, so more research is needed to explore high speed implementations of XIP's forwarding logic.

7.3.2 Large forwarding tables

While XIA supports hierarchical addressing using stacks of ADs in the destination DAG, we suspect that the flat and globally unique XIDs would tempt implementers to create large forwarding tables for principals such as hosts (within an entire organization), SIDs, or content. To understand how much leeway network designers have in making tradeoffs about routing table size and storing routes in cache vs. slower storage, we first estimate the forwarding table sizes needed for various principals. Given these estimates, we study the effects of table sizes on router performance.

Table sizes How large a forwarding table would an organization (AD) need if it wanted to use simple, direct routing to XIDs within a single AD identifier? Some of the largest organizations today have tens of millions of hosts. Google is reported to have about 1 million servers and is preparing to manage 10 million in the future [22, 12]. The largest cable operator, Comcast, serves about 23 million cable subscribers and 17 million high-speed Internet customers [9]. We estimate the number of SIDs by assuming that each host uses up to 20 ephemeral SIDs at a time on average.

We estimate the table size using a hash table with a load factor of 50% where each hash table key requires 64 bytes. Only the index and the XID (e.g., for an HID, the 20 byte hash of the host public key) are stored in the forwarding table; public keys needed for route verification are stored in a fast solid state drive or other cheap memory. Table 3 shows the storage needed for an AD with 100 million hosts. Even for large organizations, the HID and SID forwarding tables can fit in DRAM, keeping in mind that many routers in an AD would not need such large forwarding tables (e.g., edge routers could use make extensive use of default routes).

A content CID routing table stores only content hashes, not any auxiliary public keys. The size of the table is small relative to the size of the content, but the potential amount of content is massive. Assuming a 512KB chunk size for large objects (for comparison, BitTorrent uses a chunk size between 256KB and 4MB), and a hash table that uses 64 bytes per XID entry (32 bytes, 50%

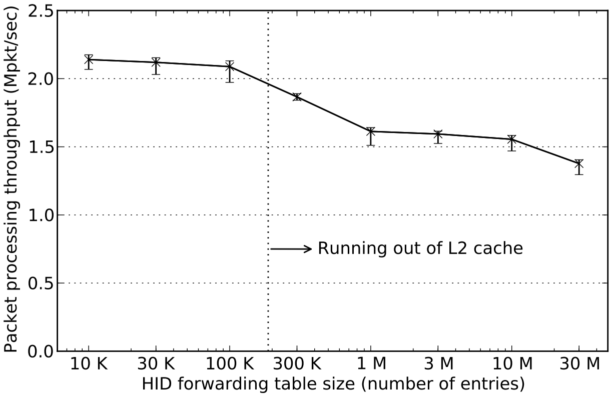


Figure 7: HID table size vs. packet throughput.

occupancy), the forwarding table is 0.0125% of the content size.

To see how this scales to large content bases, we look at the examples of YouTube (large objects) and the World Wide Web (small objects). As of June 2009, YouTube has 1.8 billion videos [7]. Assuming a conservative average video size of 100MB, the estimated content size is 168PB. Thus, the net forwarding table size is about 21TB. On the other hand, the average object size is small (7.8KB) in the World Wide Web [6]. The web contains at least 60B pages as of Jan 2010 [10], and an average web page contains about 65 objects [6]. If a web page has such many objects, and each object maps to a single, unique chunk, a forwarding table for the entire web requires about 227TB.

Organizations with such vast amounts of content would likely use an AD hierarchy to route CIDs. An alternative approach (that could also be applied in conjunction with hierarchy) is to partition the forwarding tables across a cluster of routers, similar to SEATTLE [18] or DHTs. Using these techniques one could, theoretically, store YouTube's 21TB index across roughly 672 machines each with 32GB of DRAM. We leave a detailed exploration of such alternatives for future work.

XIA's DAG addressing provides flexibility to allow many organizations to directly route HIDs, SIDs, and perhaps even CIDs, while enabling larger organizations to impose hierarchy when needed for scaling, and remaining open to future innovations in scalable routing.

Impact on performance On the same Core2-based system, we measure the effects of the forwarding table size on XIP packet processing speed. We focus on the HID forwarding table.

Figure 7 plots forwarding table size against router processing speed on a semi-log scale. Each point is the average of 10 runs, each of which uses 100M baseline XIP packets (FB0) with 64-byte payloads. As we add table entries, the throughput of packet processing decreases gradually. Packet processing with a large forwarding table scales fairly well on our prototype. From 10K to 30M entries (a 3000× increase), the performance drops by only 36%; the largest drop (23%) occurs when the table exceeds 187.5K entries, at which point the 12MB L2 cache is filled. This experiment picks random destinations to ensure worst-case cache performance.

8 Related Work

Substantial prior work has examined the benefits of network architectures tailored for individual principals; in general, we view this work as complementary to ours, and in many cases have drawn upon them for the design of individual principals. The set of relevant architectural work is too large to cite fully, but includes several proposals for content and service centric networks, such as CCN [17], DONA [20], TRIAD [14], SCAFFOLD [31] and many others. Many innovations linger within this prior work that we have not yet incorporated from a desire to err on the side of supporting multiple principals over an exhaustive implementation of a single principal type.

Extensibility through naming and indirection: In support of multiple principals, prior research has examined solutions that handle principals such as content and services using naming or overlay-based indirection. For example, the 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 at the cost of additional indirection and opportunistic optimizations such as in-core content caching. *i3* [33] similarly uses an overlay infrastructure that mediates on sender-receiver communication to provide enhanced flexibility with similar costs. Of course, these overlay-based architectures can more easily be deployed atop today's Internet, while XIA requires more substantial architectural changes.

Extensibility through programmability has been pursued in varying degrees through many efforts such as active networks [34], aiming to ease the difficulty of enhancing already-deployed networks. 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—though the active networks-style approaches provide a possible complimentary approach—but emphasizes instead creating an architecture in which new functionality can be incrementally deployed.

Architectures that evolve well have been a more recent focus. Ratnasamy et al. propose modifications to IP to enhance its evolvability [29]; compared to XIA, this work seems more easily deployable, but does not admit the flexibility of XIA's multiple principal support. 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 [27]. 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 over the past several years has examined creating *virtualizable networks* in which links can be partitioned to allow many competing Internet protocols to run concurrently [3, 38, 4, 32]. Clark, in particular, presents a compelling argument for the need to enable competition at an architectural level [8], 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 (e.g., perhaps moving to a fully circuit-switched network) than XIA. Substantial research remains in both moving all of these architectures closer to fruition and in comparing their many strengths.

Borrowed foundations: XIA borrows many foundational concepts from prior work. *Self-certifying identifiers* were explored in the Host Identity Protocol (HIP) [15] protocol, as well as distributed systems such as the Self-Certifying File System (SFS) [21]. The Accountable Internet Protocol (AIP) [2] 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 [20], use them to ensure the authenticity of content. SCAFFOLD [31] similarly names services based upon the hash of a public key. This prior work demonstrated the substantial power of these intrinsically secure identifiers, which XIA in turn generalizes to an architectural requirement for a more generic set of principals.

9 Conclusion and Future Work

The eXpressive Internet Architecture supports expressiveness, evolution and trustworthy operation through the use of multiple network-layer principal types imbued with intrinsic security. XIA integrates prior work on self-certifying and content-based naming into a new internetworking protocol, XIP, which natively supports multiple principal types through expressive addressing and fallback mechanisms. XIA supports evolution without sacrificing efficiency. With built-in support for evolution in XIP, we expect our prototype to evolve as we refactor it into a full-fledged XIP router, and as we develop and refine new principal types in our implementation.

Substantial research remains to address issues such as crafting transport protocols that take advantage of content caching; adapting or engineering suitable intra- and interdomain routing protocols for HIDs and ADs; and incorporating trustworthy protocols that leverage intrinsic security. We view this weight 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.

References

[1] Ashok Anand, Vyas Sekar, and Aditya Akella. SmartRE: an architecture for coordinated network-wide redundancy elimination. In *Proc. ACM SIGCOMM*, pages 87–98, Barcelona, Spain, August 2009.

[2] David G. Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Accountable Internet Protocol (AIP). In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008.

[3] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the Internet impasse through virtualization. *IEEE Computer*, 38, April 2005.

[4] Muhammad Bilal Anwer and Nick Feamster. Building a Fast, Virtualized Data Plane with Programmable Hardware. In *Proc. ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, Barcelona, Spain, August 2009.

[5] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. A layered naming architecture for the Internet. In *Proc. ACM SIGCOMM*, pages 343–352, Portland, OR, August 2004.

[6] Joachim Charzinski. Traffic properties, client side cachability and cdn usage of popular web sites. In Bruno Mller-Clostermann, Klaus Echtle, and Erwin Rathgeb, editors, *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, volume 5987 of *Lecture Notes in Computer Science*, pages 136–150. Springer Berlin / Heidelberg, 2010.

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

[8] David Clark, John Wroclawski, Karen Sollins, and Bob Braden. Tussle in cyberspace: Defining tomorrow's Internet. In *Proc. ACM SIGCOMM*, pages 347–256, Pittsburgh, PA, August 2002.

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

[10] Maurice de Kunder. The size of the World Wide Web. `http://www.worldwidewebsize.com/`, January 2011.

[11] Bryan Alexander Ford. *UIA: A Global Connectivity Architecture for Mobile Personal Devices*. PhD thesis, Massachusetts Institute of Technology, September 2008.

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

[13] Protocol Buffers. `http://code.google.com/apis/protocolbuffers`.

[14] Mark Gritter and David R. Cheriton. TRIAD: A New Next-Generation Internet Architecture. `http://www-dsg.stanford.edu/triad/`, July 2000.

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

[16] Van Jacobson, Diana K. Smetters, Nicholas H. Briggs, Michael F. Plass, Paul Stewart, James D. Thornton, and Rebecca L. Braynard. VoCCN: Voice Over Content-centric Networks. In *Proceedings of the 2009 Workshop on Re-architecting the Internet (ReARCH '09)*, pages 1–6. ACM, 2009.

[17] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, December 2009. ACM.

[18] Changoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in SEATTLE: A scalable ethernet architecture for large enterprises. In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008.

[19] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[20] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A Data-Oriented (and Beyond) Network Architecture. In *Proc. ACM SIGCOMM*, Kyoto, Japan, August 2007.

[21] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP)*, pages 124–139, Kiawah Island, SC, December 1999.

[22] Rich Miller. Google envisions 10 million servers. `http://www.datacenterknowledge.com/archives/2009/10/20/google-envisions-10-million-servers/`, 2009.

[23] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, Banff, Canada, October 2001.

[24] Julio C. Navas and Tomasz Imielinski. GeoCast–geographic addressing and routing. In *Proc. ACM MOBICOM*, pages 66–76, Budapest, Hungary, September 1997.

[25] University of Oregon. RouteViews. `http://www.routeviews.org/`.

[26] Performance counters for linux, 2010. `https://perf.wiki.kernel.org/`.

[27] Lucian Popa, Ali Ghodsi, and Ion Stoica. HTTP as the narrow waist of the future Internet. In *Proc. ACM Hotnets-IX*, Monterey, CA, USA., October 2010.

[28] Himabindu Pucha, David G. Andersen, and Michael Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI*, Cambridge, MA, April 2007.

[29] Sylvia Ratnasamy, Scott Shenker, and Steven McCanne. Towards an evolvable Internet architecture. In *Proc. ACM SIGCOMM*, Philadelphia, PA, August 2005.

[30] Umar Saif and Justin Mazzola Paluska. Service-oriented network sockets. In *Proc. ACM MobiSys*, San Francisco, CA, May 2003.

[31] SCAFFOLD: Service-centric architecture for flexible object localization and distribution. `http://sns.cs.princeton.edu/projects/scaffold/`.

[32] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Can the production network be the testbed? In *Proc. 9th USENIX OSDI*, Vancouver, Canada, October 2010.

[33] Ion Stoica, Daniel Adkins, Shelley Zhaung, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *Proc. ACM SIGCOMM*, pages 73–86, Pittsburgh, PA, August 2002.

[34] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. *ACM Computer Communications Review*, 26(2):5–18, April 1996.

[35] Dirk Trossen, Mikko Sarela, and Karen Sollins. Arguments for an information-centric internetworking architecture. *ACM Computer Communications Review*, 40:26–33, April 2010.

[36] VeriSign. Internet Defense Network, 2010. `http://www.verisign.com/ddos-protection/index.html`.

[37] Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, Robert Morris, and Scott Shenker. Middleboxes no longer considered harmful. In *Proc. 6th USENIX OSDI*, San Francisco, CA, December 2004.

[38] 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.