Anonymity in XIA: Developer Tools and User Control

Nicolas Feltman and David Naylor

1 Introduction

The desire for anonymous communication in the Internet has made it a much-discussed issue in both technical and non-technical circles. Several standard approaches to anonymity have emerged, normally involving some number of explicitly identified proxies due to the dumb nature of current routers. Although none of these methods are perfect, they generally meet most users' needs. Little work has been done, however, exploring ways to make these methods easily utilized by application developers and easily understood by end users.

Our goal is three-fold. First, we consider existing methods for achieving anonymous communication in the context of the eXpressive Internet Architecture (XIA). As we discuss below, novel features of XIA in many cases allow these existing techniques to be implemented more elegantly. Second, we introduce an anonymous socket API extension, which provides application developers a dead-easy way to use anonymous communication over XIA. Finally, we consider application users. Using the anonymous socket API, we will create a preference pane providing OS level control over the extended API functionality. Anonymization settings will be controllable by the user through an intuitive, easily understood GUI.

1.1 XIA

Some features of XIA have implications when it comes to anonymity. We briefly describe the key ideas here and elaborate on their impact on anonymous communication later.

1.1.1 Principal-Based Communication

In contrast with today's host-based Internet, XIA provides a framework for communication among *principals*. The idea of principal-based communication is that users should be able to address packets directly to their primary *intent*. For example, a user searching for books on Amazon wishes to communicate with `www.amazon.com`; he doesn't care which particular Amazon server responds to his request.

In this example, `www.amazon.com` can be viewed as a *service* principal. In the case where communication with a particular machine truly is the intent, traditional host-based

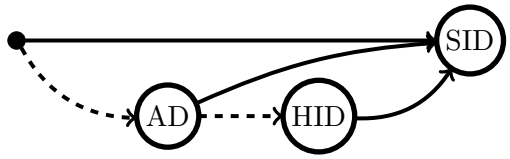


Figure 1: A typical DAG with two fallbacks. Note that higher edges have higher priority; SID is the primary intent, so the network attempts to route it first. If a router does not know the location of SID, or does not understand services, it uses the first fallback and instead routes the packet to SID's AD. Once in the AD, if a router cannot find SID, the packet is routed to HID, the second fallback, before finally being delivered to the service.

communication can still be achieved via the *host* principal. Other principals include static *content* (e.g. images) and *autonomous domains* (similar to today's autonomous systems).

1.1.2 DAG-Based Addressing

Addresses in XIA are represented as DAGs (directed acyclic graphs). Using DAGs allows senders to give the network very detailed instructions about how a packet should be routed. DAGs also allow senders to provide *fallback* routes to be used in case the network cannot find the sender's primary intent or does not understand a new principal type. For example, in the scenario above, the user's browser might include the address of a particular Amazon server as a backup in case a packet reaches a router that doesn't know how to find the service `www.amazon.com` directly.

1.1.3 Intrinsic Security

All addresses (or, better put, identifiers) in XIA are *intrinsically secure*; exactly what this means varies among principals. For example, a content identifier (CID) is the cryptographic hash of the content itself, enabling anyone receiving the content to verify its integrity. Hosts and services are required to have a public/private key pair; therefore their corresponding identifiers (HIDs and SIDs) are simply the hashes of public keys. A host can sign any communication it generates with its private key and anyone can publicly verify the signature using the host's ID.

2 Levels of Anonymity

We adopt terminology proposed by Pfitzmann and Köhntopp to precisely describe varying degrees of anonymity:

Anonymity The state of not being identifiable within a set of subjects, the *anonymity set*.

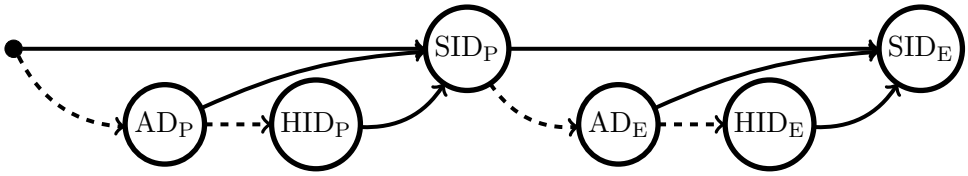


Figure 2: A host can use an in-network proxy service with a DAG like this one.

Unlinkability Two or more items (e.g., subjects, messages, events, actions, etc.) are no more and no less related than they are related to any other item.

Sender Anonymity A particular message is not linkable to any sender and no message is linkable to a particular sender.

Recipient Anonymity A particular message cannot be linked to any recipient and no message is linkable to a particular recipient.

Unobservability The state of messages being indistinguishable from no messages at all.

3 Approach: Proxies

The first approach we will discuss for providing anonymity is the use of *proxies*. The client hides its identify by routing all of its communications with a service provider through an auxiliary party called a proxy. Since the service provider only ever interacts with the proxy and never with the client directly, it does not need to know the identity (in XIA, the host ID) of the requester.

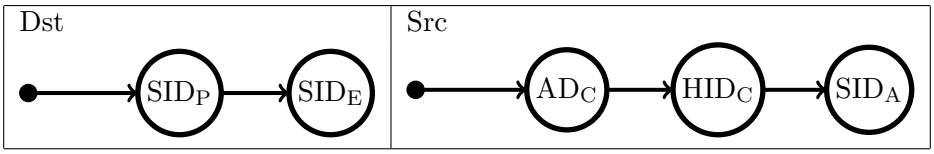
In this section we will describe one possible way to implement a proxy protocol as an in-network service in XIA.

3.1 Proxy Addresses

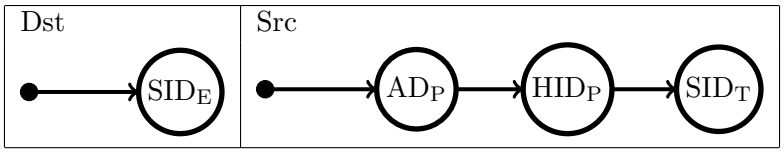
One key feature of our proxy protocol design is including the proxy's address with the address of the final service provider in a single DAG. This means that from the point of view of a client application developer, the use of a proxy can be expressed entirely within the end service's address DAG. Although we considered introducing a new principal type for proxy services, we found that they could be expressed sufficiently with the standard service principal type. The DAG in Figure 2 represents communication with the end service SID_E via the proxy SID_P, with fallback paths for both. We do not cover in this document how the proxy's DAG is obtained.

3.2 Protocol

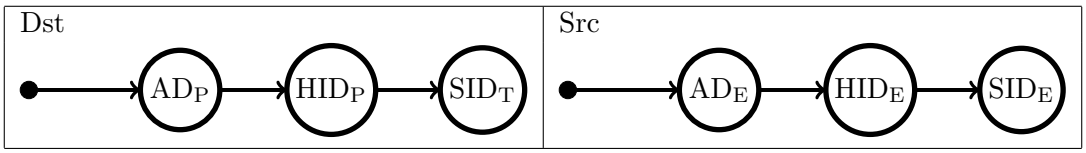
This version of the protocol assumes that the client application does not need to communicate any connection parameters with proxy. The client begins by obtaining DAGs for the proxy and for the end service. Let the proxy service have the public address SID_P and the end service have the address SID_E. Let the client application be running on a host HID_C with an ephemeral service identifier SID_A (for demultiplexing). It sends the initial packet with the following destination and source addresses (fallback paths omitted):



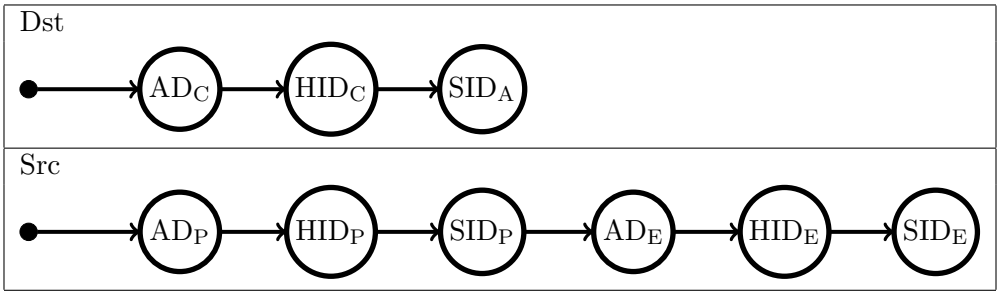
Upon receiving this initial packet, the proxy service routes it to a particular machine, henceforth HID_P, to handle this connection. Since this is the first packet from AD_C:HID_C:SID_A, the proxy creates a new ephemeral service ID, SID_T and adds this pair to a local table. (Responses from the end service, SID_E, are sent to the proxy at SID_T, which looks up the client address corresponding to SID_T in it's table and forwards the response to the appropriate address.) The machine then forwards the initial packet with the following destination and source addresses:



The end service is not aware that it is being communicated with via a proxy. It chooses a machine, HID_E, to handle this connection, and responds with the following source and destination addresses:



The proxy, in the interest of keeping minimal state, does not remember this bound source address but instead simply sends it back to the client who is responsible for including HID_E in future packets.



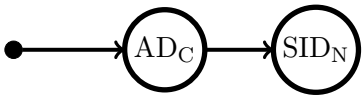
The client receives the bound source DAG, sending all future communications to it. Thus, the client continues communicating with the host via the proxy, which swaps references to AD_C:HID_C:SID_A with AD_P:HID_P:SID_T, and the end service thinks it is maintaining a persistent connection with AD_P:HID_P:SID_T.

4 Approach: Temporary Service IDs

Our second method for anonymization involves explicit cooperation of the client's AD. In this method, the client requests a temporary service ID, or *pseudonym*, from the AD through a special mechanism. It then uses this pseudonym (bound within the AD) as its return address. This level of anonymity is accepable for some applications.

4.1 Protocol

The client, running on host HID_C, starts by requesting a psuedonym from its administrative domain, which has the address AD_C. The AD, which must explicitly support this operation, then generates a temporary service identifier, SID_N, and propagates forwarding information $SID_N \rightarrow HID_C$ to its routers. Note that it is critical that AD_C not advertise this information to other ADs; only it knows the association between SID_N and HID_C From here, the client simply uses the following DAG as the source address in its packets, excluding the usual HID_C node:



5 An API for Developers

As we have discussed above, XIA allows for simple, if not elegant, ways to employ techniques for anonymization. For instance, we explored how an application can achieve fine-grain control over the use of a proxy service through DAG manipulation. We also proposed the use of temporary SIDs as pseudonyms.

Both of these strategies incorporate anonymity by leveraging core features of XIA. Thus, it is both possible and useful to provide a standard set of tools implementing these techniques to application developers — it would be silly for each developer to implement his/her own functions for adding a proxy service to all outgoing DAGs, for example.

We will implement these tools in the form of an extension to the XSocket API. (The XSocket API, as the name suggests, allows developers to communicate with sockets over XIA, much like network applications today use TCP or UDP sockets.) These new functions allow developers to specify how they want to achieve anonymity (i.e., by routing traffic through a proxy of their choice) just once, after which they can send packets using the specified service with no extra effort, using the standard `XSend()` and `XRecv()` calls. In § 5.2 we present the interface of our extension to the XSocket API. (As our implementation will involve repeated manipulation of DAGs, we also propose new standard DAG manipulation functions. We expect this list to grow as we implement the new API functions.)

5.1 DAG Library Extension

Function	Description
`AppendDag(dag1, dag2)`	Appends `dag2` to the end of `dag1`. This is used to create a proxy-routed address.

5.2 XSocket API Extension

Function	Description
`XEnableAnonymizer()`	Instructs the socket layer to route all traffic through an anonymization service. If none is specified, a default is used.
`XDisableAnonymizer()`	Instructs the socket layer not to use an anonymization service.
`XSetAnonymizer(dag)`	Sets the DAG of an anonymization service.
`XEnableTemporarySID()`	Instructs the socket layer to use a temporary SID as a source address.
`XSetTemporarySIDDuration(duration)`	Sets the duration of temporary SIDs to `duration` seconds. When a temporary SID expires, a new one is generated.

6 System-Wide User Controls

By introducing an anonymization preference pane, we allow users to ensure that all outbound traffic from their system is anonymized according to the specified settings (Figure 3).

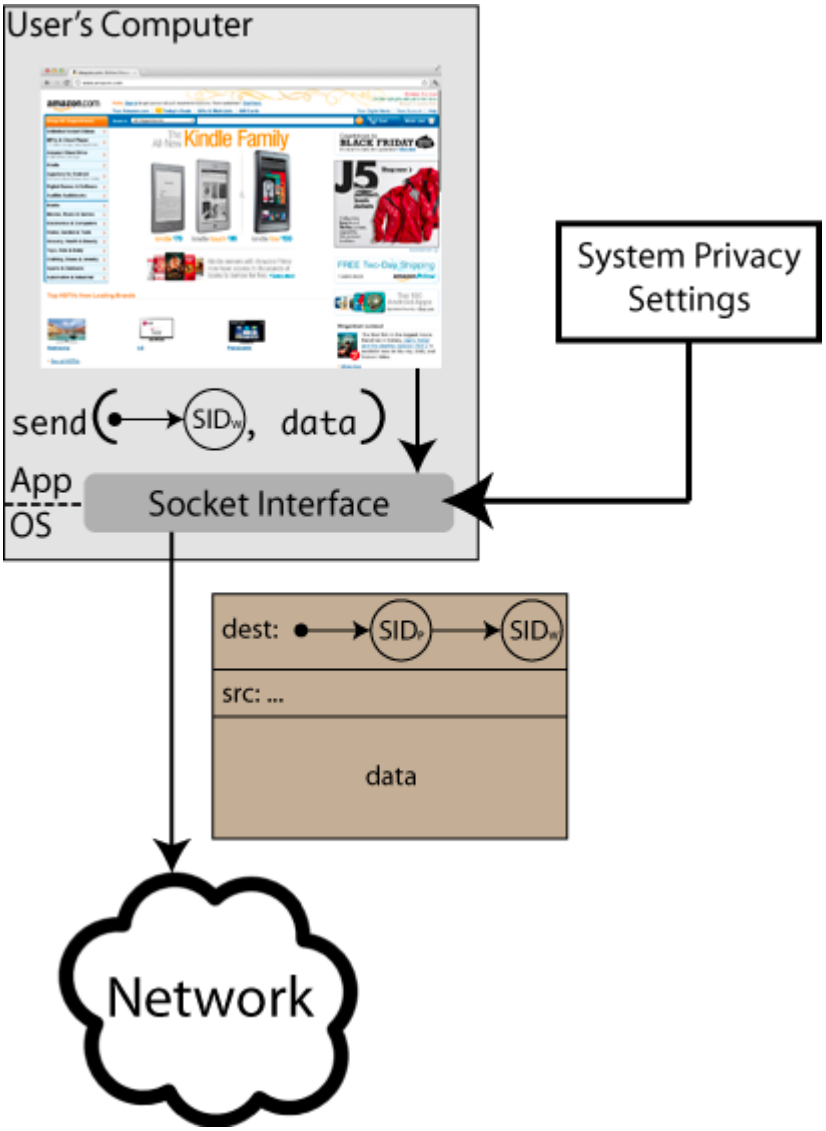


Figure 3: System-wide controls bypass applications entirely, allowing users to ensure all traffic is sent as per their anonymization settings.