Anonymity in XIA

Nicolas Feltman, David Naylor

December 13, 2011

1 Introduction

The desire for anonymous communication in the Internet has made it a much-discussed issue in both technical and non-technical circles. Users have identified several reasons for wanting online anonymity; in [2], interviews revealed motivations ranging from protecting personal safety to hiding political activity from governments to unknown fears. In one shocking interview from this study, a Romanian woman describes being abducted after posting personal information about herself on a Web site.

Several standard approaches to anonymity have emerged, normally involving some number of explicitly identified proxies (e.g., Tor). Although none of these methods are perfect, they generally meet most users' needs from a technical standpoint. Where many fall short, however, is in making these systems easy to understand and configure for non-technical users; part of our project addresses this issue.

Our goal is three-fold. First, we consider existing methods for achieving anonymous communication in the context of the eXpressive Internet Architecture (XIA) [1]. The question we ask ourselves is: Does XIA *break* anything? Second, we explore a more interesting question: Does XIA *add* anything? As we discuss below, novel features of XIA in many cases allow these existing techniques to be implemented more elegantly and in some cases enable entirely new approaches. Finally, we bring users and developers into the loop. We introduce an extension to the XIA socket API, which provides application developers a dead-easy way to use anonymous communication over XIA. Then, we create a preference pane providing OS level control over the extended API functionality. Anonymization settings are controllable by the user through an intuitive, easily understood GUI.

The rest of this paper is organized as follows: in Section 2 we introduce pertinent features of XIA and review common terminology related to anonymity from the literature. Section 3 examines the use of existing approaches in XIA as well as new ones made possible novel features of the architecture. In Section 4 we compare these approaches to one another in terms of the threats against which they provide protection and we compare the "current" version of each method to its XIA counterpart. Finally, Section 5 presents our implementation and in Section 6 we summarize and conclude.

2 Background

Before we discuss anonymity in XIA, we will first discuss XIA itself to highlight the features we will make use of later. Then, we will bring some precision to the term "anonymity" by presenting a review of the terminology used in previous work.

2.1 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.

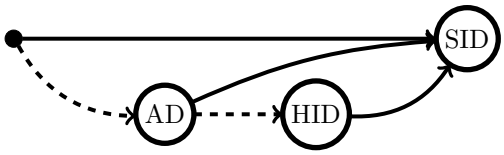


Figure 1: A typical DAG with two fallbacks. Note that higher edges have higher priority; SID is the primary intent, so the network attempts to route it first. If a router does not know the location of SID, or does not understand services, it uses the first fallback and instead routes the packet to SID's 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.

2.1.1 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 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).

2.1.2 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. Figure 1 depicts a typical DAG.

2.1.3 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.

2.2 Anonymity

We adopt terminology proposed by Pfitzmann and Köhntopp [3] to precisely describe various meanings of the term *anonymity*:

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

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.

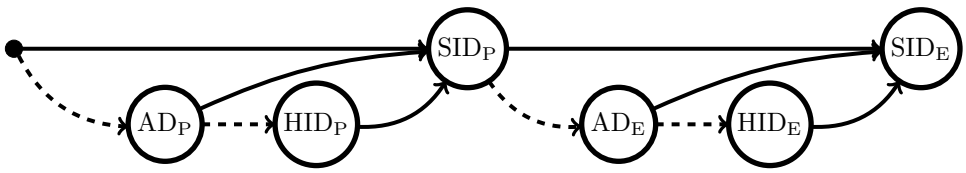
3 Approach

3.1 Proxies

One simple way for a client to achieve anonymity is to route all traffic through a third party, called a "proxy." This proxy forwards all communications in both directions, and generally looks indistinguishable from any other client. In this way, the proxy is the only agent that needs to know the address of the client.

We consider the example of Amy talking to Betsy using Paul as a proxy. Betsy alone is unable to link the messages with Amy unless Amy's identity is revealed in the messages themselves. Similarly, an attacker listening to the communication between Paul and Betsy would be unable to infer Amy's identity. Alternatively, an attaker listening to the communication between Amy and Paul might be able to link those messages with Betsy, since Betsy's address must be communicated from Amy to Paul at some point. This can easily be thwarted by encryption of Betsy's address. An attacker listening to both of Paul's communications (or Paul himself) might be able to link those communications to each other by comparing content or timing. This can be made very difficult by using several layers of proxies and encryption, such that the chance that all of them are compromised becomes low ([4]).

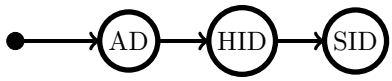
To promote maximum orthogonality of features, we decided to describe and implement proxies as an *in-network service*, by specifying the desire to send traffic through the proxy in the address DAG of the original packet, rather than encapsulating the packet inside another one addressed only to the proxy. For instance, the following DAG discribes communication with the end host at SID_E via the proxy at SID_P, with proper fallbacks for both. We go into more detail on the manipulation of these DAGs in §5.1



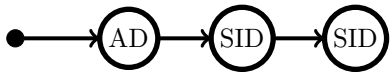
3.2 Temporary Source Addresses

A conceptually simpler approach to achieving anonymity is to use a temporary source address that is unsinkable to you. In today's IP Internet there isn't really a good way to do this. If IP addresses are statically assigned, periodically changing your address is a hassle and may require the involvement of your network administrator, who is unlikely to be willing to assign you a new address every time you want a new identity. Even if addresses are assigned dynamically and you obtain a new one from a DHCP server as often as you'd like, this solution still has major drawbacks. First, most users won't be able to do this from home. A user with a typical home Wi-Fi network can change his computer's IP address to his heart's content, but as his packets pass through his router's NAT, the source address will be overwritten with the public IP assigned to him by his ISP. Further limiting the utility of this approach, your IP address is the source of all traffic you send from your machine, regardless of the application; in some cases, you might want messages from one application to be unlinkable to messages sent by another.

Fortunately, XIA makes the use of temporary source addresses much more feasible. First, consider what normally happens when an XIA application initiates communication with some remote entity. When a host first connects to the network, it contacts a DHCP-like server in its local AD and registers its host ID. Then, when the application opens a socket the operating system will assign the socket an ephemeral service ID (akin to ephemeral ports today). Any packets sent through this socket are given a source address formed form the combination of the host's HID and this ephemeral SID:



Now, suppose we want to use a temporary source address to achieve anonymity. Upon creating the socket, the operating system generates an ephemeral SID, as before, only now it takes the additional step of registering it with the local AD as if it were another host ID via the same mechanism it used to register its actual HID when the machine joined the network. Now packets can be sent with this source address:



Note that we have overcome all of the limitations of the temporary address approach in IP: first, since the operating system can simply generate a new public/private key pair and use it to derive a new SID, there is no need to manually obtain a valid available IP address. Second, as XIA uses 160 bit addresses and is in no danger of running out, there will presumably be no need for NATs, so there is no concern that a router will rewrite our temporary source address and compromise our anonymity. Finally, we no longer have the coarse granularity problem temporary IP addresses posed; each *socket* gets its own temporary address.

Of course, this strategy is not entirely without limitations. Clearly, anyone using this technique must trust his or her local AD not to reveal the mapping between machine and temporary ID; this may not always be reasonable. On the other hand, this can be looked at as a feature rather than a bug. If an AD maintains a log the SIDs registered with it, law enforcement agencies could potentially access them with a subpoena if criminal activity is suspected.

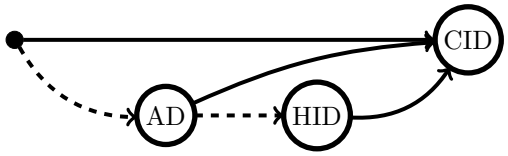
3.3 Principal-Based Traffic Control

A novel approach possible only in XIA is principal-based traffic control. Broadly speaking, a user may to prevent applications from sending packets that are addressed using certain principal types, as the use of a particular principal reveals something about the communication.

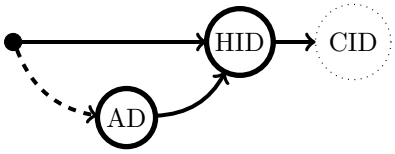
For instance, the presence of a CID makes it clear that a packet's purpose is to retrieve a piece of content. Furthermore, it is conceivable that someone might implement a "reverse lookup" service that does the opposite of the naming service: translates XIDs back into human-readable names. Thus, anyone sniffing our privacy-seeking users's traffic who sees a content request can find out the name of the content from the CID.

Additionally, as new principal types are added to the network in the future, they may similarly reveal information users would prefer to keep private.

In the current XIA design, the operating system would simply drop any packets making use of a disallowed principal (and optionally alert the user). But if a mechanism for partial DAG encryption were adopted, as discussed in §5.2, the system could rewrite DAGs in a way that conceals the use of a particular principal. For example, consider a content request with the following DAG:



Rather than dropping this packet, the DAG could instead be rewritten as follows, where the dotted line indicates that the CID node is encrypted (with host HID's public key):



4 Comparison

Before describing our implementation, we will take a moment to draw comparisons among the approaches described above in terms of the threats against which they protect, as well as compare each method to itself in terms of its implementation and utility in today's IP Internet and XIA... SAY MORE HERE?

4.1 Approaches vs. Threats

	Other Party (e.g., a company)	**Small 3rd Party (e.g., identity thieves)**	**Large 3rd party (e.g., a government)**	**ISP**	**Facebook Friends**
Proxy-based service (e.g., Tor)					
Temporary source ID					
Principal type filtering					
Real life pseudonym					

4.2 XIA vs. IP

	XIA	**IP**
Proxy-based service (e.g., Tor)	Elegantly implemented with DAGs (§3.1)	Packets for final destination are encapsulated in packets to the proxy
Temporary source ID	Generate new SID and register with local AD via HID registration mechanism (§3.2) Per-socket granularity	Request to be assigned a new IP address from DHCP server or network administrator Per-host granularity
Principal type filtering	Fine grained traffic control (§3.3)	N/A

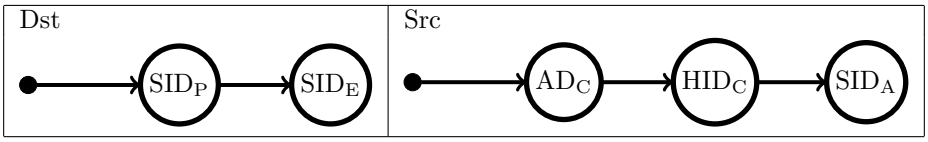
5 Implementation

We designed a simple proxy service for anonymizing XIA traffic which we refer to as Envoy. Due to limitations of the underlying XIA codebase, we have implemented only a functional equivalent of Envoy which runs on a single machine and does not handle multiple connections at once. Our version of Envoy is written in Python and communicates over XIA using the Xsocket interface. In §5.1 we describe in detail how we use DAGs to route traffic through Envoy. The process of implementing this service brought to our attention a handful of issues relating to in-network services, which we discuss in §5.2.

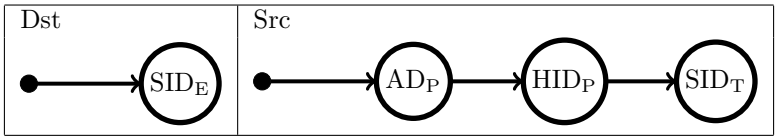
In keeping with our goal to bring users and developers into the loop, we also implemented an extension to the Xsocket API aimed at giving application developers easy access to anonymization tools (§5.3) and a preference pane enabling users to have the final say as to what applications can and cannot do with respect to anonymity (§5.4).

5.1 Single Proxy DAG Manipulation

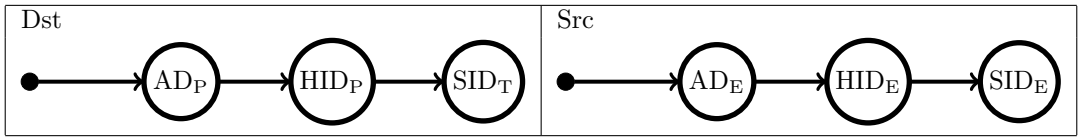
This version of the protocol assumes that direct communication or authentication between the client and Envoy happens through some other channel. The client begins by obtaining DAGs for the Envoy service and for the end service. We will refer to the public addresses of Envoy and the end service as SID_P and 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):



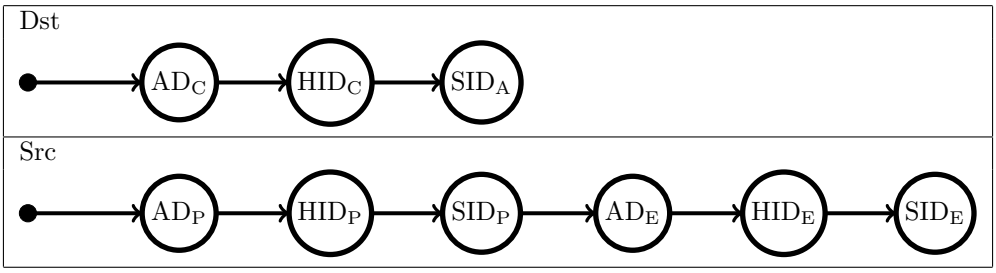
Upon receiving this initial packet, Envoy routes it to a particular machine, henceforth HID_P, to handle this connection. Since this is the first packet from AD_C:HID_C:SID_A, Envoy creates a new ephemeral service ID, SID_T and adds this pair to a local table. (Responses from the end service, SID_E, are sent to Envoy 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:



Envoy, 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.

5.2 Issues Related to In-Network Services

In-Network Services (INSs) are handled poorly in the current Internet (they are primarily implemented as transparent middle-boxes that violate the end-to-end principle). A distinctive feature of XIA is its potential to resolve many of the issues present today through its flexible DAG-based addresses and intrinsically secure IDs. In the course of implementing our proxy, we uncovered a few issues that can be generally applied to all INSs in XIA.

Firstly, as we hope that INSs will become a distinctive feature of XIA, the XIA library needs to offer first class support for implementing them. This will involve, as a first step, expanding the XSocket API; to see how this might be useful, consider this illustrative example: an AD has implemented a firewall as an INS, through which it requires all inbound traffic to pass. (It could enforce this policy by instructing all edge routers to add the firewall to the destination DAG of all inbound packets, if it is not already present.) Currently, however, the firewall application's only option is to receive data via the `Xrecv()` or the `Xrecvfrom()` call. This presents a problem: each of these calls only exposes the packet's data and source address to the application, but a firewall will likely want access to the entire packet header. Thus, perhaps the `Xrecvfrom()` call could be enriched to provide the application with the entire packet heard rather than just the source DAG.

Additionally, when the the application passes the packet back to the socket layer to be forwarded to its final destination, the socket layer must be aware that the packet is not starting at the beginning of the DAG but rather at the INS's node somewhere in the middle and mark the "current node" pointer in the DAG accordingly.

Another as yet unaddressed issue is that of DAG "permissions." That is, who is allowed to modify a DAG? Only the sender? The receiver? What about INSs? Furthermore, are there any restrictions to what modifications are allowed? In the protocol we describe in §5.1, as packets pass through the proxy, the proxy strips its node out of the destination DAG so that the final destination isn't aware it is being communicated with via a proxy.

Finally, we suggest that XIA allow for partial encryption of DAGs. In the case of an anonymizing proxy, the part of the DAG after the proxy needs to be encrypted with the proxy's public key so that messages between a client and a proxy do not publicly reveal the identity of the final destination.

5.3 For Developers: XSocket API Extension

The XSocket API, as the name suggests, allows developers to communicate with sockets over XIA, much like network applications today use the BSD socket API. We extended this functionality to include support for some of the anonymity methods we have described by implementing a thin layer over the XSocket API (using Python for quick prototyping).

These new and modified 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()`, `Xrecv()`, and `XgetCID()` calls. Table 1 presents the interface of our extension to the XSocket API.

5.4 For Users: Control and Transparency

Making developers' lives easier is only half the story — good software is only as good as its user interface. Even the most sophisticated mechanism for anonymous communication is useless if users don't understand or trust it. As anonymity is becoming a first-class concert to users [2], we propose that operating system vendors build support for various forms of anonymous network communication into their systems, as well as easily understood controls. Although the ideas in this section aren't necessarily specific to XIA, re-architecting the Internet is a pretty good time to find out what users want and build it into the new architecture; thus, we think this work is still valuable to XIA.

We implemented a sample "System Privacy Settings" preferences pane, shown in Figure 2. It gives users control over the anonymity methods in §3, all of which except "Temporary IDs" are implemented. The OS

Function	Description
`Xconnect(sock, dDAG)`	Opens a connection with `dDAG` using system anonymization settings, which might include adding a node for a proxy service to the DAG or replacing the host's HID with a temporary SID in the source DAG.
`XconnectCustomAnonymizer(sock, dDAG, pDAG)`	Opens a connection with `dDAG` that sends all traffic through a proxy specified by `pDAG`. The user will be prompted to grant or deny the application's request to deviate from system anonymizer settings.
`XconnectWithoutAnonymizer(sock, dDAG)`	Opens a socket with `dDAG` that doesn't send traffic through any anonymizer. Useful if the application wants complete control over where the proxy is placed in the DAG. The user will be prompted to grant or deny the application's request to bypass system settings.
`XgetCID(sock, cDAG)`	Instructs the socket layer to retrieve the content specified by `cDAG`. Like `Xconnect()`, this modifies the DAG to send the request through an anonymization service as per system settings.
`XgetCIDCustomAnonymizer(sock, cDAG, pDAG)`	Like `XconnectCustomAnonymizer()` but for content requests.
`XgetCIDWithoutAnonymizer(sock, cDAG)`	Like `XconnectWithoutAnonymizer()` but for content requests.

Table 1: New or modified XSocket API calls and their descriptions.

configures the behavior of the socket layer to conform to the user's choices (see Figure 3).

But control alone is not enough. Once users have customized the behavior of their systems, they would like a way to verify that their systems are acting on their preferences. To this end, our prototype includes a chart (Figure 4) which displays to the user counts of three different classes of packets: packets sent through the anonymizer specified in the system preference pane (i.e., packets sent from `XgetCID()` or from connections established with `Xconnect()`), packets sent through a different, application-specified anonymizer (`XgetCIDCustomAnonymizer()` or `XconnectCustomAnonymizer()`), and packets sent through no anonymizer at all (`XgetCIDWithoutAnonymizer()` or `XconnectWithoutAnonymizer()`). Though our interface is simplistic and unpolished, it serves to emphasize the need for system transparency.

6 Conclusion

References

[1] Ashok Anand, Fahad Dogar, Dongsu Han, Boyan Li, Hyeontaek Lim, Michel Machado, Wenfei Wu, Aditya Akella, David Andersen, John Byersy, Srinivasan Seshan, and Peter Steenkiste. XIA: An architecture for an evolvable and trustworthy internet. Technical Report CMU-CS-11-100, Department of Computer Science, Carnegie Mellon University, February 2011.

[2] Ruogu Kang, Sara Kiesler, Peter Kinnaird, Colleen Stuart, and Laura Dabbish. Perceptions about anonymity on the internet. 2011.

[3] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity — a proposal for terminology. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin / Heidelberg, 2001.

[4] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Proxies for anonymous routing. In *Proceedings of the 12th Annual Computer Security Applications Conference*, ACSAC '96, pages 95–, Washington, DC, USA, 1996. IEEE Computer Society.

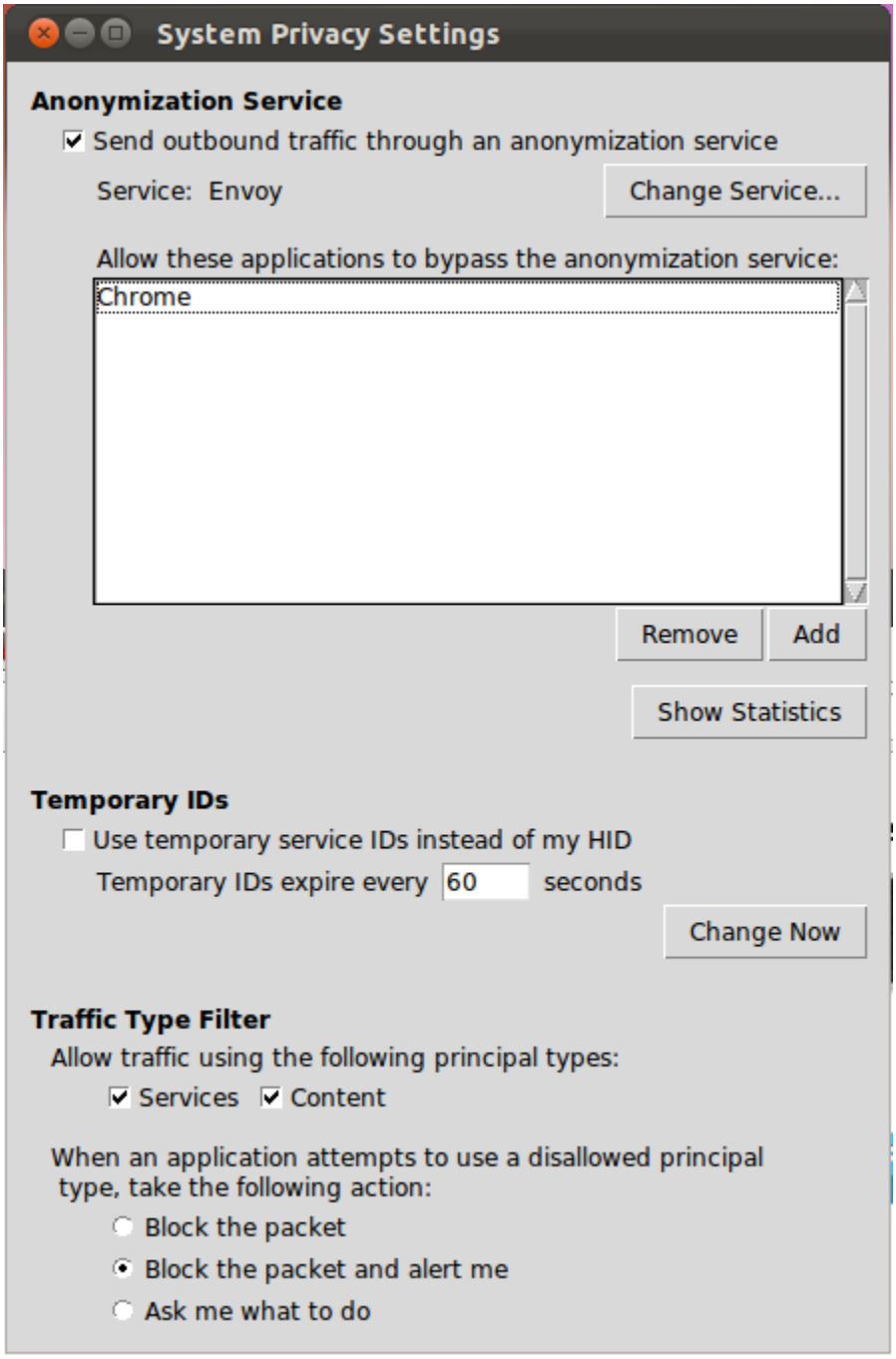


Figure 2: A preference pane lets the user tell the OS how to anonymize network traffic.

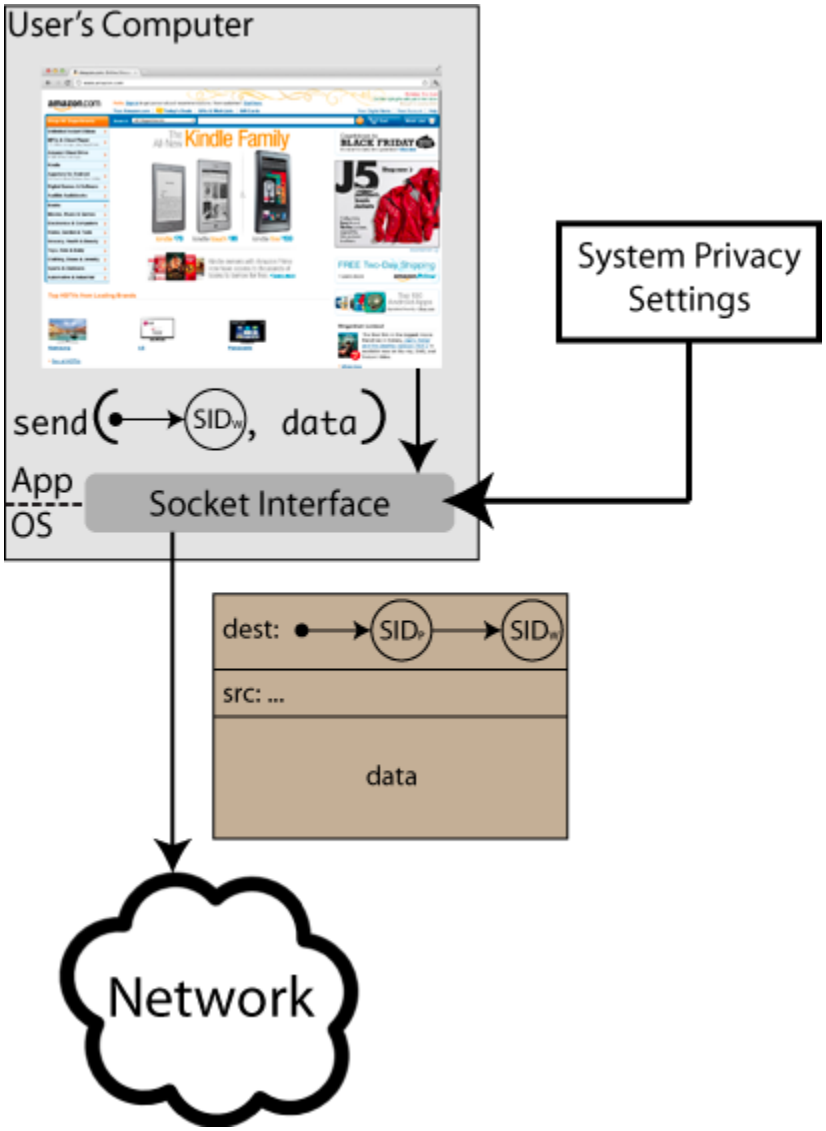


Figure 3: System-wide controls affect the behavior of the socket layer, bypassing applications entirely, allowing users to ensure all traffic is sent as per their anonymization settings.

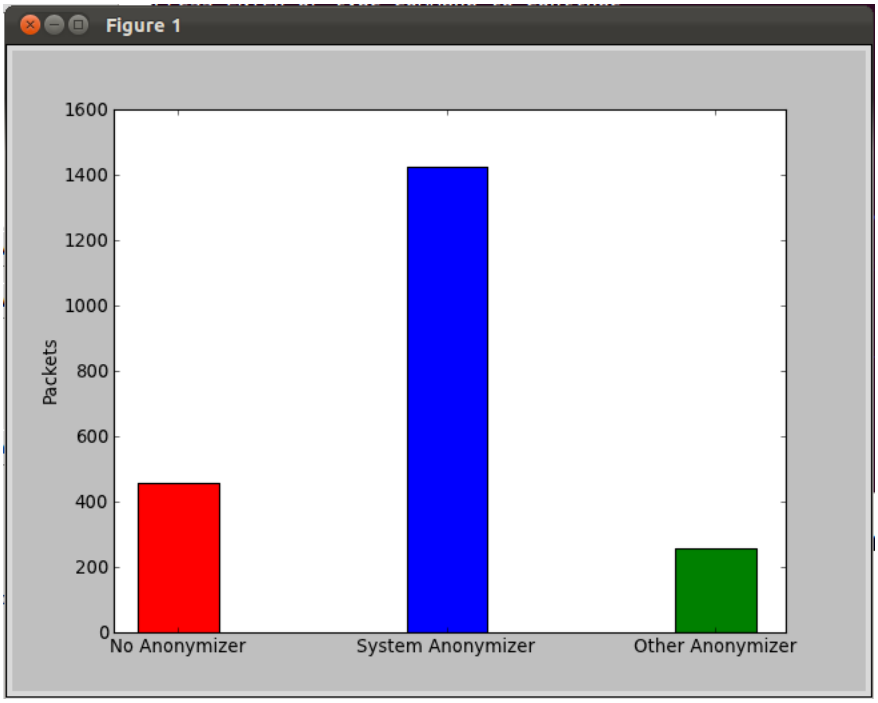


Figure 4: An example of feedback the operating system might give a user.