Expressive Privacy Control with Pseudonyms

Seungyeop Han, Vincent Liu, Qifan Pu, Simon Peter
Thomas Anderson, Arvind Krishnamurthy, David Wetherall
University of Washington
{syhan, vincent, qp, simpeter, tom, arvind, djw}@cs.washington.edu

ABSTRACT

As personal information increases in value, the incentives for remote services to collect as much of it as possible increase as well. In the current Internet, the default assumption is that all behavior can be correlated using a variety of identifying information, not the least of which is a user's IP address. Tools like Tor, Privoxy, and even NATs, are located at the opposite end of the spectrum and prevent any behavior from being linked. Instead, our goal is to provide users with more control over linkability—which activites of the user can be correlated at the remote services—not necessarily more anonymity.

We design a cross-layer architecture that provides users with a *pseudonym* abstraction. To the user, a pseudonym represents a set of activities that the user is fine with linking, and to the outside world, a pseudonym gives the illusion of a single machine. We provide this abstraction by associating each pseudonym with a unique, random address drawn from the IPv6 address space, which is large enough to provide each device with multiple globally-routable addresses. We have implemented and evaluated a prototype that is able to provide unlinkable pseudonyms within the Chrome web browser in order to demonstrate the feasibility, efficacy, and expressiveness of our approach.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—Security and protection; C.2.6 [**Computer-Communication Networks**]: Internetworking

Keywords

Privacy; IPv6; Pseudonym; Web tracking

1. INTRODUCTION

Technology trends and economic forces are moving us toward a world in which personal information is valuable, and companies have an incentive to obtain as much of it as possible. Targeted advertising, for example, has revolutionized revenue generation, but while this is good for companies, the implications for user privacy are less positive.

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.
ACM 978-1-4503-2056-6/13/08.

Tracking has evolved from individual websites keeping tabs on their users. Today's third-party trackers are ubiquitous enough to capture a significant fraction of users' web browsing behavior, and they are continuously moving towards wider deployment and even collusion with other services. Google is an extreme example of what is possible, as they theoretically have access to a user's complete logs of email, web history, phone calls, contacts, location information, and much more in addition to being an advertiser. There is no evidence that Google is currently aggregating information on that scale, but the capability exists.

We are not trying to argue that linking of user activities is *always* bad. In fact, it can be a useful tool that benefits both the user and the web service—banks deter fraud by detecting when a new computer is used to access an account, advertisers support useful services by providing relevant ads/product suggestions, and analytics platforms help websites understand and design for user behavior. However, the line between gathering appropriate information and violating privacy is currently undefined. Our position is that where to draw the line should be up to the user.

While existing defense mechanisms like cookie management systems and other browser tools are able to control tracking to a certain extent, they generally do not handle implicit information like IP addresses. Tracking using IP addresses is just as accurate as cookies, and in today's network, unblockable. Lower-level solutions like Tor [6], proxies, or NATs can make IP addresses less meaningful to trackers, but they are coarse-grained, lack flexibility, and in the case of Tor, can hurt performance.

Our goal is instead to provide users with *more control* over what is linkable and what is not. We do this by co-designing modifications across all relevant software layers in order to protect users' privacy. Central to our design is the concept of a **pseudonym**, which represents a set of user actions and provides an unlinkable abstraction of a single machine.

In particular for the network layer, we leverage the size of the IPv6 address space. Unlike today's world, where IP addresses are limited and thus, every activity of every user on a machine (or every machine behind a NAT) is lumped into the same abstraction of a single machine, IPv6 addresses are able to provide a different grouping. The larger address space of IPv6 can potentially allow every activity of every user to have a different address (in fact, there are more than half a million addresses per square nanometer of the Earth's surface).

However, it is not enough to give each user a pile of IPv6 addresses if the entire pile can be traced back to the user. Pseudonyms also need to encompass more than just an IP address as activities can be linked using traditional application- and system-level information. For example, in the case of browsers, potential sources of linkability can be found in cookies, HTML5 LocalStorage, Flash

LSOs, and implicit information like user-agent and system fonts [27]. Our system ensures that all of these elements are consistent within a pseudonym, and privacy-preserving across pseudonyms.

Contributions:

In this paper, we introduce a psuedonym abstraction that gives users control over linkability across layers. To the best of our knowledge, this system is the first to provide integrated cross-layer control over privacy and present it as a first-class abstraction. To this end, we develop a system that allows for each machine to have numerous pseudonyms and design a complementary network layer that can handle a large number of seemingly random, flat addresses with no path dilation, no increase in routing table size, and negligible performance decrease, all while maintaining network transparency. We also delve into a case study of browsers, where we explore issues related to the isolation of pseudonym information, and policies aggregating activities into pseudonyms. Our simple policy system allows us to implement a variety of protection mechanisms, each of which has different functionality-privacy tradeoffs. Finally, we have implemented a Chrome extension and cloud gateway service that approximates a pseudonymous browser in today's Internet (i.e., an IPv4 Internet without network/OS support). Our prototype is publicly available at `http://netlab.cs.washington.edu/project/pseudonymous`.

2. MOTIVATION

Advertisers and web services have strong incentives to track the behavior of users on the Internet—often without user consent, control, or knowledge. A recent study conducted by Roesner *et al.* [23] shows the prevalence of third-party trackers across both popular and randomly sampled domains.

Tracking can sometimes be beneficial to all parties involved, by providing customization and security along with a revenue model for many web services. On the other hand, tracking can just as easily be abused. For example, they might correlate the collected traces to other sources of users' personal information, such as gender, zip code, name, or sensitive search terms [13] to obtain detailed profiles. These profiles can then be used to perform price discrimination [17] or be used by advertisers to display behavior-based or profile-based ads, which sometimes reflect sensitive interests (e.g., medical issues or sexual orientation) [26].

In this section, we first outline the techniques employed by services to track users and show tracker prevalence on the web. Then we discuss why existing solutions such as NATs and anonymizing proxies are not sufficient to support the broad range of policies that a user might desire.

2.1 Tracking Methods

The tracking mechanisms available to webservices/advertisers are numerous and span the entire network/application stack. These mechanisms are often out of the control of the user, or otherwise difficult and inconvenient to spoof. They include:

- **IP address:** IP addresses let adversaries correlate user activity without requiring additional application-level information. Even when the IP address changes, as long as other information persists across the change, the new IP address can be linked to the user. It is relatively easy for services to link activities generated by the same IP.
- **Application information:** Often, the application itself will leak information. Web browsers, for example, have cookies, Flash LSOs, and HTML5 LocalStorage that can be used to track the behavior of users.

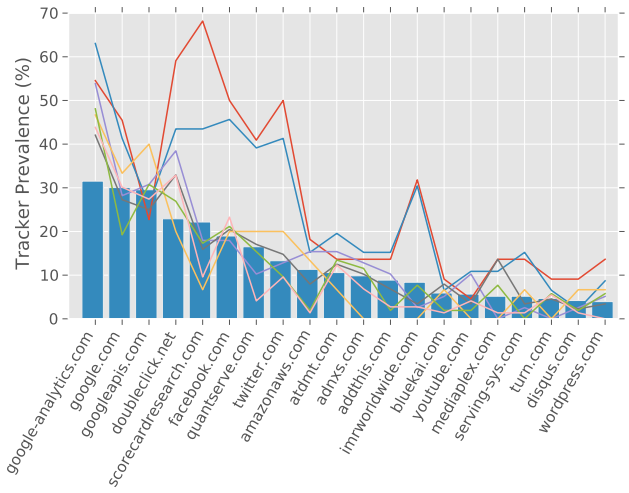

Figure 1: Prevalence of encounters with the top 20 third-party trackers within a total number of 406 distinct domains visited over all traces. Each line represents a separate user's trace.

- **DNS:** The address of the server accessed by the user can reveal information as well. CDNs, for example, may use DNS to distribute requests, and if these entries are reused, subsequent accesses can be linked to the original.
- **System information:** These include time zone, screen size, system fonts, etc. They can be mined through applications. These values are often important for proper functionality, but studies have shown that the set of values is often unique across different users [7].

2.2 Tracker Prevalence

As reported in previous studies, third-party trackers are prevalent on the web. We conducted small-scale measurements by using web usage traces of authors and collaborators to see how often we encountered third-party trackers while webbrowsing. The dataset covers three days of web traces from eight users. The traces were generated via a Chrome web browser extension developed by the authors that logs details of each HTTP request made by the browser. Those details include information such as the URL accessed, the HTTP header, an identifier for the browser tab that made the request, a timestamp, and what caused the request (e.g., a recursive request for some embedded object or a result of the user clicking a link). Figure 1 shows the frequency of encounters with the top 20 third-party trackers within our traces[1]. The users visited 406 unique domains during the trace collection. Out of the 406 domains, 281 (69.2%) domains contained at least one third-party tracker.

2.3 Current Defenses

Most web browsers expose cookie management policies to users (e.g., blocking third-party cookies or only allowing session cookies). Browser add-ons can add finer-grained policies, but regardless of the expressiveness or the flexibility, these application-level solutions do not handle lower-level information like IP addresses.

Network-layer solutions like Tor, proxies, and NATs attempt to alleviate this problem, but none of these are ideal. The default policy on the Internet is that all of a user's behavior can be tracked regardless of what they do, and while these systems provide an alternative policy, the one they enable lacks flexibility: they make IP

[1]We used the list of trackers from [23].

addresses meaningless. For example, a user might want to allow Amazon to track searches/purchases in order to receive better recommendations, but at the same time may not want to allow trackers like DoubleClick to correlate interactions across websites. In many cases, there is a fundamental tradeoff between functionality and privacy, and the above solutions do not provide users with control over this tradeoff.

In addition to a lack of flexibility, Tor sacrifices performance because it is an overlay with an attack model that is stronger than what is necessary for an anti-tracking system. Proxies, to a lesser extent, also have performance issues because of path dilation and bandwidth restrictions. NATs may not incur the same penalties, but instead destroy network transparency and violate key tenets of Internet design [5].

The other issue with proxies/NATs is that websites already use IP addresses as hints to increase security[2]. In our system, we attempt to allow applications to leverage IP addresses as a powerful tool for accountability.

3. APPROACH

We allow users to manage a large (potentially unlimited) number of **unlinkable pseudonyms** so that they can choose which pseudonyms are observed by which remote servers. In this case, a pseudonym refers to a set of identifying features that persist across an *activity*—the definition of activity here is dependent on the user's policy.

3.1 Threat Model

The fundamental goal of our system is to prevent remote services, with which a user interacts, from linking the user's activities except in ways that the user intends. The adversaries that work against the user and our system are remote trackers that wish to build a more complete profile of that person and/or to perform service discrimination. Their objective is to correlate any and all behavior of each user in an effort to link the user's pseudonyms. To do so, they may also collude with endhosts owned by users in our system to find out pseudonym mappings of other hosts in the system.

For web browsing, adversaries may use any information revealed in the packet including, but not limited to, cookies, local storage, fingerprinting, and unique URLs. They may also collude, span multiple services and include other types of entities like CDNs and DNS servers. Their ability to link two activities can be sometimes be probabilistic, particularly in the case of browser fingerprinting, but as soon as an adversary sees sufficient evidence that two activities are from the same user, they, and any associated activities, are thereafter tainted.

	Persistent ID	**Transient ID**
Self-contained	Banking Websites	Blogs
Pervasive	Social Media Sites	Analytics

Table 1: Examples of potential adversaries.

Within this framework, websites have varying levels of tracking potential and requirements for functionality. On one axis, there are some trackers that are pervasive and some that only track first-party accesses, and along this axis, services can have differing levels of prevalence (e.g., Google Analytics has a presence on 297 of the top 500 domains, while bluekai has presence on 27 [23]). On the other axis, services require varying levels of persistent identifying information to function. This sometimes depends on the expectations of the user in question, and can range from sites that require login, to sites with recommendation systems, to sites that do not utilize persistent identifiers for functionality and where each access is essentially independent. (See Table 1 for examples of this categorization.)

Note that our model does not include intermediaries like routers and ISPs, especially users need to trust their first-hop ISPs. LAP [9] assumes similar threat model for users who want to have intermediate privacy level without sacrificing latencies. In order to defend against these types of attackers including the first-hop ISPs, we would need to launder packets through other machines, such as in Tor [6] or Herbivore [8].

3.2 Pseudonyms

We take the approach that a pseudonym is, to the outside world, the abstraction of a single machine and, to the user, simply a collection of activities that they wish to correlate. Tor, proxies, and NATs, fail to provide this abstraction, and we lose not only the user's control over what activities are correlated, but the ability to use IPs as a single machine identifier as they were intended to be. While one might argue that this removal of IPs as a useful identifier provides more anonymity, *what we are trying to provide is more control, and more control is not synonymous with more anonymity.*

In our approach, a single user may have multiple pseudonyms and may appear to be multiple separate machines/users. This is somewhat similar to Chrome's notion of profiles except that linkability (intra-pseudonym) and unlinkability (inter-pseudonym) need to extend across multiple layers, not just the application-layer. Pseudonyms of the same user should not have information that is both common between each other and unique among all pseudonyms in the system. In general, we either want identifying information to be unique or shared among the pseudonyms of a large set of users across the entire system—the bigger the set, the better.

Pseudonyms should also provide consistent information across the end-to-end connection, and thus maintain network transparency. This prevents accidental leakage of information within the contents of a packet and strengthens the end-to-end principle, avoiding problems endemic to NATs (such as the difficulty of setting up peer-to-peer connections).

3.3 Desirable Policies

With pseudonyms, our system is able to concurrently support a variety of potential policies. Below we list a few of the use cases that illustrate the power of our approach.

Sort by Identity: Users use their computers for a variety of different purposes and for a variety of different activities. It is sometimes incorrect to relate these activities, particularly if the computer is used by multiple users or for varied interests and activities. For example, a user may want to create separate pseudonyms for her work-related behavior and what she does in her free time.

Banking Websites: Banking websites track users in order to combat fraud. A common security feature of online banking makes the observation that it is much more likely for access from a new computer to be fraudulent than for accesses from a computer that the user has used in the past. For this purpose, a user may want to use a single pseudonym for all visits to her bank, to convince the banking website that she is the same person who used it previously.

Separate Sessions: Unrelated requests for particular services or resources may need to remain private. For example, differ-

[2] http://bit.ly/VAj9cI

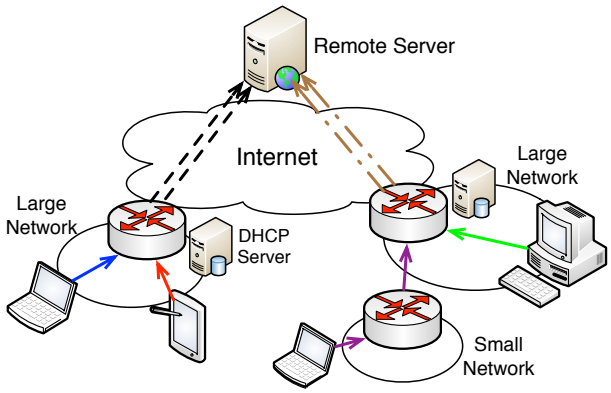

Figure 2: Overview of the Architecture. Color of the lines indicates identifiability—the same colored lines are not discernible.

ent files downloaded by a single user from BitTorrent need not be linked together and could each be in a separate pseudonym without significantly affecting usability of the system. Note that, in this particular example, the pseudonym crosses applications as an adversarial BitTorrent tracker can potentially link the BitTorrent activity to the HTTP download of the .torrent file (and thus the web history of the user).

Block Third-Party Tracking: Even if a user is fine with giving out information to a website to maintain customization or provide analytics, third parties can aggregate information to create comprehensive profiles for individual users. Blocking third party cookies is not sufficient to protect against this because of the information listed in Section 2.1. Stronger protection can be obtained by using a separate, random pseudonym for requests to third-party sites.

4. DESIGN

We now outline our design and the changes we make to the network, operating system, and application layers. The pseudonym abstraction necessarily needs to be implemented across many layers—specifically, adversarial web servers can potentially use any information in the network layer and up in order to link pseudonyms and track users. Each layer needs to change to accommodate the ability to allocate/deallocate addresses, and a leak at any layer can potentially break the illusion of separate machines.

Beyond this, our system needs to have answers for a few key challenges:

- What do we need to tackle at each of the layers (network, system, application)?
- What network or operating system support is necessary to handle numerous IPs for each machine?
- How are packets classified into activities or pseudonyms and what potential polices are possible or useful?

At a high level, our system modifies client machines and applications to allow for usage of pseudonyms, and pseudonyms are mixed with those of other users in the same network/ISP or partnering networks. Networks are responsible for providing IP mixing for their customers, and DHCP servers are responsible for allocating multiple IP addresses to the client upon request (i.e., provide an IP address per pseudonym). Only clients, their applications, and routers within participating networks are modified in our system. See Figure 2 for an overview of our system.

4.1 Network Layer

In this subsection, we introduce the addressing and routing schemes we use in the network layer to enable unlinkable pseudonyms. The basic goal is to provide each host a large number of IP addresses that can be tied to pseudonyms, and as a consequence, the network layer design should meet the following three requirements:

R1 Proper mixing: The externally visible addresses of pseudonyms corresponding to a host should appear to be randomly chosen. This will prevent destinations and routers external to the mixing ISP from being able to link the activities associated with a host.

R2 Efficient routing: Addressing and routing within the ISP should be efficient. For example, the size of routing tables with the addition of pseudonyms should be comparable to that of routing tables without them, and the per-packet overheads should be minimal.

R3 Easy revocation: The network should be able to periodically remap the association of pseudonyms to externally visible addresses in order to minimize the risks associated with brute force attacks and compromised keys.

Traditionally, addresses in the Internet consist of a network prefix and a host identifier, which can be further divided into a subnet number, and a host number. Routers in the Internet then use longest-prefix matching in order to efficiently route over such addresses. However, if each subnet and computer has a preset prefix, this would leak information to adversaries about where a particular IP address comes from, and the scheme would fail to meet the requirement R1. We introduce an addressing method where the non-network-prefix portion of addresses *appear* to be randomly chosen from a flat address space, but are still efficiently routable (i.e., can be done at line speed with the same number of routing table entries).

Large Networks: We first consider large networks, where there are enough users to achieve unlinkability by just mixing within the network. In this case, the portion of the address not included in the network prefix should appear to be completely random so as to not leak any information about the subnet or host.

A naïve approach would be to associate each pseudonym with a randomly generated IP addresses in the network and route based on IP addresses within the ISP. However, this approach requires routers to maintain routing tables that are proportional to the number of actively used pseudonyms, with the random address allocation preventing the use of aggregation of routing table entries. Such a scheme would fail to meet R2.

We instead leverage symmetric key encryption to encode the host identifier in a way that is not discernible to outside adversaries and yet can be efficiently routable inside the ISP. At a high level, addresses are constructed using an initial base address consisting of a network prefix, subnet identifier, host identifier, and pseudonym number, with the first three values representing exactly what they do in today's systems. The addition we make is to give each host multiple pseudonym IDs in order to generate distinct addresses for each pseudonym.

The base address is used to generate an **encrypted address** that obfuscates identifying information and will appear to be drawn randomly from a flat address space. More specifically, the portion of

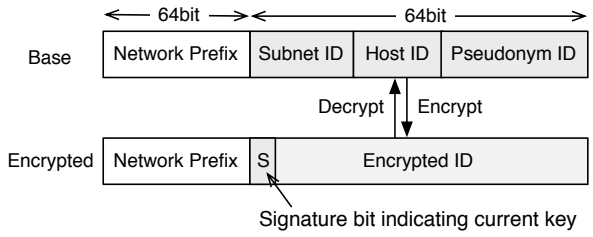

Figure 3: Translation of a base address and encrypted, IPv6 address. Note that some of this address space may be reserved for a DMZ. The base address is not known to anyone except the mixing network's routers and DHCP servers; The outside world and end host only see the encrypted address.

the address that includes the subnet ID, host ID, and pseudonym ID is encrypted using a symmetric key as shown in Figure 3. Assuming the network has a /64 prefix, the later part of the base address is encrypted with 64-bit symmetric key block encryption[3]. The key is only known by the routers and DHCP servers, as are all pieces of the base address to protect against known-plaintext attacks.

On each incoming packet, routers within the network perform a single symmetric key decryption on the destination address to uncover the base address, which they can then use to forward packets efficiently. Base addresses can be longest-prefix routed and aggregated on subnets or hosts, exactly as in the current Internet. We therefore do not increase routing-table state compared to current IPv6 networks. Also note that we maintain network transparency as routers do not overwrite the IP address, and the end host sees the same encrypted address as the other end of the connection.

Guessing the symmetric key with a brute-force attack is computationally very difficult; while a host knows that the encrypted addresses provided to it are obtained from base addresses that have a common set of bits, exploring the entire key space to identify the secret symmetric key is computationally challenging (e.g., our prototype uses TDES for encryption, where the key space has 2^{168} keys). However, best practices for key management require ISPs to periodically change the encryption keys. To allow easy revocation (R3), networks can create a new key while allowing users to phase out old keys by appropriating a *signature* bit at the beginning of the encrypted ID and requiring that all encrypted addresses that use a particular key have that bit set at a specific value—there can only be two active keys at any given time and all addresses without the correct bit for the current key will be thrown out. This may decrease the available address space unevenly for particular hosts, but the average should simply be a factor of 2. Each router maintains up to two symmetric keys (the active key and possibly the old key) and determines which key to use based on the initial bit of the encrypted ID.

When a machine wishes to allocate IP addresses, it broadcasts a DHCP request along with its hardware address and the number of desired pseudonym IDs. The DHCP server will generate the requested number of random pseudonym IDs, and use them to generate a set of base addresses. It will then encrypt them with the active secret key and then send them back to the host. If any of the addresses do not have the correct signature bit for the current key, they will be thrown out and replaced before being sent to the host.

[3]The recommended allocation for end sites is at least one /64 block [19]. In the case of larger blocks, we can simply ignore the higher bits or segment the users across multiple /64 blocks.

This increases the work done by the DHCP server by a factor of two, which we believe is acceptable overhead.

It is also possible to design almost stateless DHCP in this system. If the conversion between hardware address to host ID is consistent (e.g., with a consist hash function), then the DHCP server does not need to store any mappings, it simply needs to know the encryption key and hash function and can generate pseudonym IDs on-the-fly. Any machine with those two pieces of information can serve as a DHCP server. It is possible that this technique can result in redundant pseudonym IDs, but the host is required to maintain the set of encrypted addresses currently being used by all of its applications and will filter out duplicates before passing on the DHCP results to the requesting application. Note that the same pseudonym ID should not be used twice for the same host and key in order to prevent unintentional linking of activity. If we run out of pseudonym IDs, either the network administrator should have allocated a larger portion of the address for the pseudonym ID or the host should acquire a new host ID.

Lastly, we allow for truly static IP addresses within a network (e.g., for web servers) by reserving prefixes for DMZ IP addresses. For example, the network can specify that all addresses starting with `0000` or `1000` be reserved for the DMZ. Such addresses are not encrypted, and like the 1-bit signature, the DHCP server will throw out and regenerate any encrypted addresses that overlap with this range. The length of these reserved prefixes is variable and so is the amount of extra work done by the DHCP server.

Smaller Networks: For smaller networks that do not have enough users to hide individuals effectively, one solution is to merge their address pools and delegate address assignments to a larger, possibly adjacent, network. Allocation requests from hosts in the smaller network will be forwarded to the larger network's DHCP server. Because there is no mutual trust among the two networks, we add two more requirements for this case:

R4 The larger network (L) should not learn about or control the mixing of the smaller network's (S's) hosts.

R5 S should not learn L's encryption key.

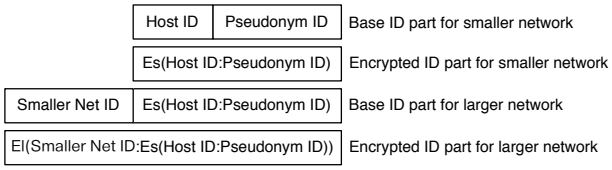

Figure 4: Hierarchy of encrypted addresses for smaller and larger networks. Note that prefix is not shown here. The final encrypted ID is prefixed with the larger network's prefix.

To fulfill these requirements, address allocation is done hierarchically as shown in Figure 4. Upon receiving a DHCP request from a host, S generates psuedonym IDs for the host. Then, it encrypts ⟨hostID:pseudonymID⟩ with a smaller block encryption cipher. The encrypted address is forwarded to L, and L attaches it to S's ID assigned by L, and encrypts it again with L's encryption method. Finally, this resulting address is used along with L's network prefix as the externally visible IP address for the endhost.

Since this address has L's network prefix, incoming packets will be first routed to L. Upon receiving a packet, L decrypts the address with its own key and finds out that it needs to send the packet to S.

Since S might not be adjacent to L, L encapsulates the incoming packet with an address of Prefix(S):$E_s\langle$HostID:PseudonymID$\rangle$ and routes it towards S's ingress routers. Upon receving this packet, S can locate the final destination within its network by decrypting the address stored in the encapsulation header with its key. The encapsulation is stripped off just before the packet is delivered to the endhost. Note that potential MTU violation may lead fragmentation, but it could be made to work reliably in IPv6 [24].

As HostID is encrypted together with pseudonym ID before a packet is sent to L, L cannot determine whether two IDs are allocated to a single host. Also, L's encryption key is not exposed to S. Thus, this mechanism meets R4 and R5. Support for key revocation and DMZ can be achieved just as with larger networks (as discussed earlier).

Deployability: The proposed architecture above does not require changes to the IP address format or inter-domain routing. Thus, it is partially deployable (i.e., only participating networks can adopt the above addressing and routing mechanisms without affecting other networks). While the design described in this section provides an appropriate long term solution, one which allows network operators to natively contribute to greater privacy, it requires changes to an ISP's network components (e.g., DHCP servers and routers).

To ease adoption without requiring changes to all routers in the network, translators can be deployed in the network at the appropriate choke points through which all traffic passes. Before packets are introduced into the network, the translator performs a decryption and rewrites the IP address. Additionally, just before a packet arrives at the end host in the network or just before the packet is delivered to the application at the end host, another translator reencrypts the address, thus avoiding packet header decryption at every hop inside the ISP. Modifications to DHCP servers are still required in order to provide end hosts with encrypted addresses, but these modifications are relatively minor.

Moreover, we can also use overlay solutions to traverse portions of the Internet that do not support IPv6. These include tunneling over an IPv6 broker and application-layer proxies (e.g., HTTP proxies) We will describe the design and implementation of one such proxy and the corresponding browser extension in Section 6.

4.2 Operating System Layer

The operating system maintains the set of addresses and acts as a mediator between the DHCP server and the application. The number of addresses maintained is a preconfigured value that depends on the memory and processing power limitations of the machine in question. That being said, microbenchmarks have shown that, with minor modifications, modern operating systems can handle thousands of addresses with negligible penalties (see Section 7.1).

The OS, like the application and network layer, can also be a source of linking information. DNS lookups and caching are an example of this problem, and for this reason, pseudonyms do not share DNS caches. CNN can point a user to a particular Akamai server and set a long TTL, but it will only be cached for that particular pseudonym. Lookups should be done using a DNS resolver that could be accessed by any node in the mixing network. For example, if Comcast is mixing addresses for its customers, then (a) customers should be able to use any Comcast DNS resolver, (b) the resolvers should be indistinguishable, or (c) the customers should use a third-party resolver (e.g., Google, OpenDNS).

```
1: n = 100;
2: pseudonyms = allocpseudonym(n);
3: bzero((void*)&hints, sizeof(hints));
4: hints.ai_family = AF_INET6;
5: hints.ai_socktype = SOCK_STREAM;
6: hints.ai_protocol = IPPROTO_TCP;
7: getaddrinfo("www.google.com","80",&hints,&dest);
8: for (i = 0; i < n; ++i) do
9:     sock = socket(INET6, SOCK_STREAM, PROTO_TCP);
10:    bind(sock, &pseudonyms[i], sizeof(pseudonyms[i]));
11:    connect(sock, dest → ai_addr, dest → ai_addrlen);
12:    send(sock, data, dataLen, 0);
13:    close(sock);
14: end for
```

Figure 5: Example code that allocates 100 pseudonyms and uses them to send data using 100 different IP addresses

4.3 Application Layer

Each application is responsible for setting policies and associating requests to pseudonyms. We dive into the policies and application-level concerns for one very common application in Section 5: web browsers. The policies of other applications as well as the manner in which they protect against information leakage are beyond the scope of this work.

The interface we provide to applications is simply a modification to the socket interface. We add a function called `allocpseudonym()` that takes as a parameter an integer representing the number of desired pseudonyms. The function returns an array of `sockaddr_in6` structures with information about each pseudonym's address. Similarly, we also provide `freepseudonym()` that applications use to release an address back into the pool of available pseudonyms. IP addresses can be leased/refreshed for a period of time, but there is maximum number of held addresses for both machines and applications. An example usage is listed in Figure 5.

A default pseudonym exists for backward compatibility and ease of programming for applications that do not require unlinkabililty. Also note that multiple applications can use the same pseudonym if they wish to associate with each other. For example, a standalone banking application may wish to associate itself with the pseudonym that the customer uses inside the browser to access its website.

5. A PSEUDONYMOUS WEB BROWSER

Any Internet communication can potentially be used to track users, but in this paper, we concentrate on a particularly common avenue of attack: web requests from browsers. For every application, we need to answer two questions: (a) what is contained within a pseudonym and (b) how is traffic classified into pseudonyms. Below, we discuss these issues in the context of web browsers.

Components: Within a web browser, tracking of users is often done through explicit mechanisms like browser cookies, Flash LSOs, or other forms of local storage [23]. Thus, each pseudonym needs to have separate storage for these entities in addition to using separate IP addresses. There is also plenty of implicit information that can be used to correlate user behavior [7, 27], and the browser must manage all of these pieces of information as part of a pseudonym.

Policies: Recall that a pseudonym represents a set of linkable activities. It is possible to provide users with the ability to explicitly specify a pseudonym for each web request manually; however,

Privacy Protection Mechanism	Policy	Pseudonym ID to Use
Trivial	Every request uses the same pseudonym	default
3rd-party blocking	For 3rd-party request, use random pseudonym	3rd-party?requestId:default
Per browsing session: opening a new tab or typing new URL makes a new session	Pseudonym per tab and new page (no referer)	tabId.(sum += referer?0:1)
Per 1st-party (Milk [25]): force 3rd-party to set different cookies for each 1st-party	Pseudonym per 1st-party domain	domain(tabUrl)
Per tab (CookiePie [2]): Use different cookies for each tab	Pseudonym per tab	tabId
Per page: Use different cookies for each page	Pseudonym per page	tabUrl
Private browsing	New pseudonym for private browsing window	private?windowId:default
Per request: No activity should be correlated	Pseudonym per request	requestId
Time-based (TorButton [21])	Allocate a new psuedonym every 10 minutes	Time.Now / 10 minutes

Table 2: Example pseudonym policies. 1st column: name and description for the privacy protection mechanism. 2nd column: implementation with pseudonym. 3rd column: the policy's pseudonym ID mapping.

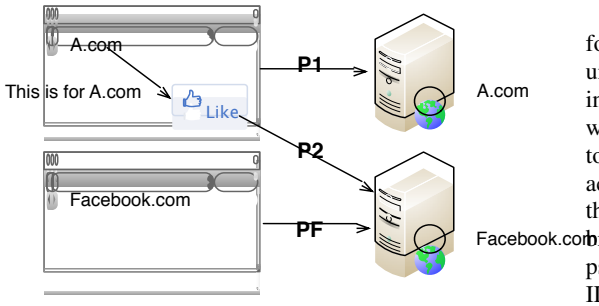

Figure 6: An example of pseudonym policies. Here, the website A.com has a Facebook widget for registering likes. A web request to A.com results in a third-party web access to Facebook, which would contain information regarding the original page.

this quickly becomes infeasible. Therefore, we provide policies to allow the system to assign these pseudonyms automatically.

To a first approximation, there are two properties that characterize the usefulness of a particular policy: website functionality and user privacy. These properties are subjective and sometimes at odds with each other. While most users probably expect logins to still work, some may be willing to trade recommendation system accuracy or targeted ads for more privacy, and some might not. In our system, we intend to provide a policy framework that is flexible and expressive enough to allow for a broad range of policies.

To illustrate how a typical policy might work, Figure 6 shows an example scenario where the Facebook social widget is embedded on a web page, A.com, and each of the three resulting requests is assigned some pseudonym (i.e., P1, P2, or PF). Without any privacy protection mechanism, pseudonyms P1, P2 and PF are the same. The other extreme is that every request uses a different pseudonym (P1 $\neq$ P2 $\neq$ PF). We can also imagine a policy that blocks third-party tracking (P1 = PF $\neq$ P2) and a policy that assigns a pseudonym per first-party site (P1 = P2 $\neq$ PF). In the latter three cases, Facebook cannot track the user's visits to `A.com`, but they do not allow Facebook's Like button to function either. Note that in the remaining case (P1 $\neq$ P2 = PF), Facebook can track the user, as it knows the Like button request is coming from `A.com`. Each policy exhibits a different point in the functionality-privacy tradeoff, although some combinations (e.g., P1 = PF $\neq$ P2 by default, but link PF and P2 when the user explicitly intends it, as shown in ShareMeNot [23]) allow users to have both.

In our system, pseudonyms are assigned to requests based on information about the request as well as the state of the browser (e.g., unique tab ID, tab URL, unique request ID, whether private browsing mode was enabled, whether the request targets a third party website, etc.). Specifically, policies define a mapping from requests to pseudonym IDs, where IDs are unique, arbitrary strings of characters. A pseudonym per-tab policy, for instance, might simply use the tab ID as the pseudonym ID. Similarly, a Chrome-like private browsing policy might use the window ID of the request as the pseudonym ID for private requests, and use a default pseudonym ID (see Section 4.3) for all other requests.

This policy framework is expressive enough to implement many of the recent privacy protection mechanisms that have been proposed to defend against web tracking. Most of them focus on separating browser cookies or blocking specific requests, but we are able to extend these mechanisms into pseudonym policies, which cover both IP addresses and cookie stores. Table 2 shows how several privacy protection mechanisms can be implemented via pseudonym policies.

Users may also combine policies (e.g., a per-tab policy with third-party blocking). More complex policies, such as one that emulates ShareMeNot, can be implemented with a toggle button indicating whether third-party authentication is allowed (if allowed, we use the default pseudonym, otherwise a random ID is chosen). Even more complex policies involving arbitrary relationships, like search term keyword clustering, can be implemented as well, but the point is that our system is powerful enough and expressive enough to support a vast array of policies.

6. IMPLEMENTATION FOR WEB BROWSERS

We have implemented an approximation of our ideal design that enables pseudonyms for web browsing in today's Internet. We are able to provide pseudonyms without modification to the browser, operating system, or network. Ideally, these layers would have support for pseudonym allocation and usage, and IPv6 would be available without tunneling through a broker, but our system provides a proof of concept and path to partial adoption. Note that IP address separation across pseudonyms only works when the destination server is using IPv6 addresses; however, cookie separation works even with IPv4 servers.

Our prototype implementation consists of two main components: a gateway service that allocates/translates IPv6 addresses and a browser extension that controls pseudonyms and their usage. The

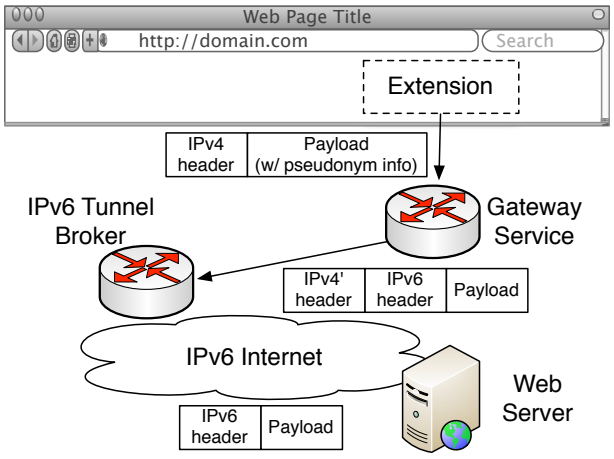

Figure 7: Overview of the Implementation.

gateway service is implemented as a stand-alone proxy running on Linux, and the browser extension is made for the Chrome web browser. The overall architecture is illustrated in Figure 7. These components work together to allow each pseudonym to have its own set of cross-layer features that include:

- **IP addresses:** The external IP address for each connection is explicitly indicated by the extension in the header of the HTTP request. The extension allocates/deallocates IPv6 addresses by communicating with the gateway.
- **Name resolution:** Requests for web pages arrive at the gateway with domain names rather than IPs. Name resolution takes place on the gateway instead of the local machine so DNS mappings cannot be used to track hosts.
- **Cookie/LocalStorage:** Sets and gets of local storage are intercepted and modified by the extension so that cookies for separate pseudonyms are isolated from each other.
- **Hardware specs, system information and browser details:** Browser details like the `user-agent` or fonts can be manually set to values that are generic so as to not identify the user. Users can tell how unique their browser details are by utilizing services like Panopticlick [7]. Ideally, this tool would be able to suggest values, provide per-gateway statistics, and potentially utilize per-pseudonym value changes, but this is left for future work.

Each of these pieces of identifying information needs to be consistent within a pseudonym, but crafted such that they are unlinkable to each other. Note that we consider higher-level leakage of information such as form data orthogonal.

6.1 Utilizing IPv6 Without Modification to the Network

Our implementation is designed to be deployable immediately even though we rely on IPv6 addresses. While actual use of IPv6 is not yet common—only about 14.6% of ASes are running IPv6[4]—the support is there in both OSes and many of the larger services. Because of these limitations, traffic from our system is tunneled through an IPv6 tunnel broker called `Hurricane Electric`.[5] This does not affect the unlinkability of any pseudonyms since the tunnel brokers only see that traffic is coming from our gateway servers. Thus, each client effectively has a single **private** address assigned by its ISP, and many **public** IPv6 addresses assigned by our gateway servers. In the future, widespread IPv6 deployment will allow us to circumvent the tunnel broker and potentially increase performance.

[4] `http://bgp.he.net/ipv6-progress-report.cgi` as of Oct., 2012.

[5] `http://tunnelbroker.net`. There are also other tunnel brokers like `SixXS`, `http://sixxs.net`.

6.2 Gateway

Our system utilizes a gateway service that handles allocation of addresses and laundering of packets so that individual clients can use our system without network support. Gateways are implemented as transparent HTTP proxies running on Linux machines. Clients can potentially use multiple gateways to send traffic. Each of these gateways controls a /64 subnet of IPv6 addresses assigned through an IPv6 tunnel broker as explained in Section 6.1, allowing us to dole out and mix traffic over an address space of 2^{64} addresses—much larger than the entire IPv4 address space. The gateway has the following responsibilities:

Allocation/Deallocation: Gateways run an HTTP web server to handle IP allocation/deallocation requests that are generated by the browser extension. The addresses are assigned as they are in our ideal design—the public addresses given to each user are efficiently mappable to the private address and vice-versa without having an easily discernible pattern. When a client requests an IP allocation, the gateway creates a set of base addresses using its own network prefix and the IPv4 address of the requester as host ID. It then generates encrypted addresses as detailed in Section 4.1 before sending the list of IP addresses back to the client. We implemented Triple DES (TDES) using `OpenSSL` to perform 64-bit block encryption on the non-network-prefix portion of the address.

Forwarding: After allocation is complete, clients are responsible for crafting web requests to include the encrypted IPv6 address that is to be used for that connection, which it does by adding a string to the header's `user-agent` field. When the gateway receives requests, it reads, decrypts, and removes the IPv6 address from the header and verifies that the client is allowed to use the address. If the user's identifier matches, it checks whether the pseudonym is active for the particular user. (The state of pseudonym IDs is stored in a bit vector.) If the address is valid and active, it then sets up a connection to the endpoint with a socket bound to the specified IP address and uses the connection to forward requests.

As described above, web requests arrive at the gateway with a domain name, allowing the gateway to ensure that all hosts use the same DNS resolver and mappings.

6.3 Browser Extension

We have built a Chrome browser extension to implement the client-side component of our system, but the same functionality can be implemented in other browsers that have an extension architecture (e.g., Firefox). The architecture and layout of the extension is illustrated in Figure 8. The extension is responsible for managing pseudonyms and assigning them to requests according to policies. It does this by separating cookies on a per-pseudonym basis, intercepting every request made from the web browser, and adding a tag to indicate to the gateway which IPv6 address to use. These functionalities are described in more detail below:

Policy Specification: Policies are defined along with a mapping function that is used to define which pseudonym to use for each request. As a starting point, we implemented each of the

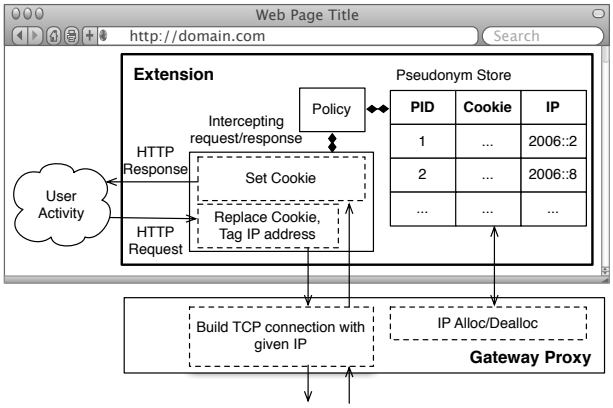

Figure 8: Diagram of the structure of our implementation and its main components: the extension and gateway proxy.

policies detailed in Table 2, but more sophisticated users can use JavaScript to define their own request to pseudonym ID mappings.

Request Interception: The extension adds listeners for web requests and responses with `chrome.webRequest` APIs (i.e., `onBeforeSendHeaders` and `onHeadersReceived`), which allow it to read and modify HTTP requests and responses. When an HTTP request is captured by the extension, it uses the policy-defined mappings to determine which pseudonym to use for the request. Incoming HTTP responses can be mapped to a specific pseudonym by using a unique identifier that Chrome provides called *request ID*.

Tagging: To notify the gateway of which IPv6 address to use, the extension appends the desired IPv6 address to the `user-agent` passed in the HTTP header, which will be examined and stripped at the gateway. While HTTPS packets are encrypted and therefore cannot be examined at the proxy, the HTTPS tunnel is built using the `CONNECT` protocol. A `CONNECT` packet includes a `user-agent` field that we can use to communicate pseudonym information[6].

Address Allocation/Deallocation: When the extension is low on IPv6 addresses, it sends an allocation request to the gateway in the form of an HTTP request. It releases these addresses if they are ephemeral and will not be used again. It also maintains a reserve pool of addresses (currently 10) to prevent requests from being blocked due to the lack of free IPs.

Cookie Storage: To separate cookie stores on a per-pseudonym basis, we prepend a pseudonym ID onto each cookie key. This makes every cookie unique to the particular cookie <path, pseudonym> pair. Cookies are either set through the an HTTP response's `set-cookie` header or using JavaScript. HTTP requests and responses are all intercepted and the cookie-related headers rewritten so that outgoing requests contain the original cookie keys, and incoming responses contain cookie keys that have been prepended with the psudonym ID. Cookie (as well as HTML5 LSO) accesses through JavaScript getters and setters are overridden by custom JavaScript functions (see Figure 9).

[6]As of the time of writing, Chromium/Chrome has an issue where it does not expose the CONNECT request to the extension. This has been classified as a bug: `http://code.google.com/p/chromium/issues/detail?id=129572`.

```
1: document.__defineSetter__ ('cookie', set_cookie);
2: document.__defineGetter__ ('cookie', get_cookie);
function set_cookie(rawCookie)
1: tuple = rawCookie.split('=')
2: remove_cookie(pseudonymID+ '_' +tuple[0])
3: __cookie+ = pseudonymID+ '_' +value+ ';'
function get_cookie()
1: ret = "", cookies = __cookie.split(';')
2: for all c in cookies do
3:    PID = c.substring(0, c.indexOf('_'))
4:    if PID = pseudonymID then
5:       noPIDcookie = c.substring(c.indexOf('_')+1)
6:       ret+ = noPIDcookie+ ';'
7:    end if
8: end for
9: return ret
```

Figure 9: Pseudocode for overriding JavaScript cookie code. A similar technique can be used for HTML5 LocalStorage.

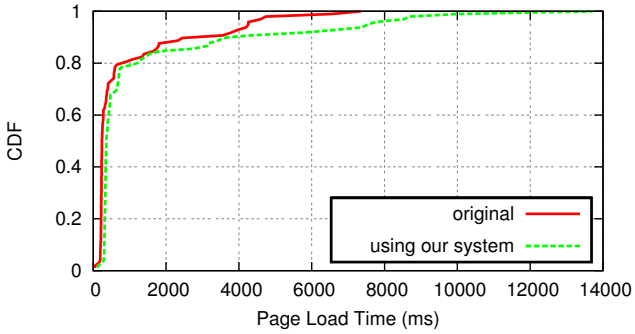

Figure 10: CDF of page load time for Top 100 IPv6 websites tested with and without our system.

7. EVALUATION

In this section, we present the evaluation of the performance and privacy aspects of our system. We seek to answer whether our system scales to the required number of pseudonyms under various privacy policies and how different policies compare with regards to the preservation of privacy. We drive the evaluation of our system by a combination of performance measurement and web-usage trace study, utilizing the HTTP request traces presented in Section 2.2.

7.1 Performance

To evaluate the performance implications of our system, we utilize a combination of an end-to-end comparison of page load time and micro benchmarks using our prototype implementation. We find that current technology already has more than enough capacity to implement each aspect of our system.

End-to-End Performance: To evaluate the overall performance of the implementation, we conducted an end-to-end measurement of page load times of the top 100 Alexa[7] websites that support IPv6. Note that 25% of the overall top 100 Alexa websites support IPv6, and that the 100th IPv6 supporting web site was ranked 869th in the whole ranking.

We use the Chromium benchmarking extension[8], which collects HTTP connection performance data. We measure web page load

[7]`http://www.alexa.com/topsites`
[8]`http://bit.ly/14ErTju`

	User	1	2	3	4	5	6	7	8
Trivial	Pseudonyms	1	1	1	1	1	1	1	1
	Activities	651	4657	359	872	63	692	599	615
	Collections	38	101	338	515	91	336	179	166
Per tab	Pseudonyms	36	123	64	174	36	186	111	412
	Activities	9.50/41	14.15/67	5.02/31	4.48/40	1.75/17	3.48/30	4.33/33	1.21/16
	Collections	0.94/9	0.81/12	5.00/22	2.64/34	2.31/8	1.55/7	1.45/20	0.31/5
Per browsing session	Pseudonyms	140	858	178	350	38	244	227	670
	Activities	4.65/28	5.43/33	2.02/27	2.49/34	1.66/17	2.84/17	2.64/30	0.92/16
	Collections	0.27/9	0.12/12	1.90/10	1.47/13	2.39/5	1.38/6	0.79/9	0.25/5
Per 1st-party	Pseudonyms	425	730	393	655	158	534	579	1319
	Activities	0.04/15	0.03/19	0.04/16	0.05/36	0.03/4	0.04/23	0.03/17	0.00/5
	Collections	0.06/26	0.11/77	0.56/220	0.49/318	0.35/55	0.31/166	0.21/122	0.09/121
Per page	Pseudonyms	1276	7701	915	2152	178	1412	1689	3356
	Activities	0.51/28	0.60/33	0.39/27	0.40/34	0.35/17	0.46/17	0.35/30	0.18/16
	Collections	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0
3rd-party blocking	Pseudonyms	22982	29313	9233	16132	1437	11287	18605	33983
	Activities	0.17/13	0.27/19	0.10/16	0.15/36	0.08/4	0.07/21	0.06/17	0.04/5
	Collections	0.00/24	0.00/73	0.03/216	0.03/307	0.05/54	0.02/158	0.01/116	0.00/114
Per request	Pseudonyms	63062	59045	17586	49252	4906	33644	42645	79321
	Activities	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0
	Collections	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0	0.00/0
Time-based	Pseudonyms	155	109	67	126	153	151	106	350
	Activities	8.34/78	19.57/79	4.58/42	9.38/60	0.38/20	3.89/40	7.37/59	1.40/23
	Collections	0.23/13	0.91/9	4.42/18	3.43/20	0.52/11	1.83/20	1.54/20	0.35/5

Table 3: Trace study results under different privacy policies. The numbers show average/maximum over all pseudonyms. Pseudonyms shows the number of pseudonyms created. Activities shows the number of observed page views by third-party trackers. Collections shows the number of page views and links followed, as observed by first-party trackers.

times from a desktop Linux machine for each website with and without our system. When using our system, the gateway is located within the local network and the tunnel broker is in the same city. For each iteration, we clear the browser cache and close all open connections. Figure 10 shows a CDF of median page load time over 10 iterations of this measurement. We can see that our implementation adds only small overhead. Median and 90th percentile page load time of the top 100 sites increased from 237ms to 367ms, and from 3.6s to 4.3s, respectively.

OS-Level Performance: The number of pseudonyms supported by our system is limited by the number of IP addresses we can assign concurrently to a network interface without performance degradation. For example, the Linux operating system enforces a configurable[9] default limit of 4096 addresses.

We determine the number of pseudonyms that would be required under the different privacy policies and compare this to the maximum number of pseudonyms. The results are shown in the *Pseudonyms* row in Table 3 and we see that only in the restrictive cases of policies 4 and 5 do we exceed the default limit. The average number of pseudonyms is typically below 1000 for the other policies.

Assigning a large number of addresses to each network interface is feasible today without modification to the Linux kernel. Linux can already assign and utilize 4096 addresses on a single interface with negligible slowdown in performance. There is no reason to believe that a very large number of pseudonyms cannot be used concurrently.

We evaluated the performance of socket operations when a large number of addresses is allocated to a network interface and found no slowdown for most socket operations, including socket creation, connection setup, and sending data (via `connect()`, `accept()`, and `send()`). Binding the socket to an address using the `bind()` system call, however, incurred linear slowdown of up to 8x, depending on the address being bound. Binding addresses took 1 microsecond in the best case and 8 microseconds in the worst case. This is due to the data structure used to store assigned IP addresses. Linux uses a linear list of all IP addresses and a hash of the IP address as an index into the list. The hash only takes the upper 64 bits of an address into account and the list has to be searched linearly for IP addresses that change only the lower 64 bits. With a sufficiently large address pool, traversing the list can incur significant overhead. This can be fixed simply by changing the hash function to incorporate all bits of the IP address. Further tests on an unmodified Linux kernel revealed that OS-level routing tables are not affected by an increase in addresses, and only the data structure that holds the list of valid addresses is affected.

Router Performance: Another potential performance issue is that we add a decryption step for every packet at each router. Although this does increase the complexity of routers, we view this as an acceptable tradeoff for fine-grained privacy control. Recent research shows that encrypting and decrypting entire packets at line speed is possible [10, 12]. Our proposed changes are less extreme in that we only encrypt a portion of the address. From our microbenchmarks, our current TDES implementation decrypts at 2.24 million packets per second with one core of an Intel Xeon E5620 2.40 GHz CPU. This scales to 17.92 million packets with 8 cores, and assuming 1500 bytes per packet, results in over 200 Gbps.

7.2 Privacy Preservation

To investigate how much privacy can be preserved by our system, we analyze how much information a web tracker can collect

[9]Via `/proc/sys/net/ipv6/route/max_size`.

about each user in the trace study under each of the privacy policies presented in Table 2 except d *private browsing*, which was not used by any user in our trace study.

To do so, we use collusion graphs [1] that describe page visits among web domains. Nodes in a collusion graph represent domains and directed edges from node *a* to node *b* represent HTTP requests made by the browser to domain *b* due to an action carried out on a page of domain *a*, such as following a hyperlink.

Figure 11 shows an example collusion graph of a single web session extracted from our traces. It includes a lookup of technical documentation on Google and navigating to a corresponding page (`us.pycon.org`). The page in question uses Google Analytics to track page visits, which accesses several sub-domains to display active content.

Each privacy policy results in a different number of generated pseudonyms. We create a collusion graph for each pseudonym and subsequently evaluate how many different page views a third-party tracker is able to observe. The *Activities* row in Table 3 shows this number for the investigated policies. We see that a separate pseudonym *per browsing session* can drastically reduce the amount of information third-party trackers are able to collect about a user over a vanilla web browser (*trivial*) and that narrowing this policy does little to reduce the amount of information any further (e.g., *per page*). First-party trackers can be limited with a more restrictive policy. A pseudonym *per request*, for example, eliminates all third-party and most first-party tracking, but these policies can also detrimentally impact the browsing experience. The domain-based policy (*per 1st-party*) can provide for a better browsing experience than the policy of a pseudonym *per browsing session* because it allows first-party sites to generate personalized profiles (e.g., a Google search profile).

Anonymizers, such as Tor+Torbutton, which use a timeout-based approach to anonymity and change pseudonyms every 10 minutes (equivalent to our *time-based* policy), are less effective in combating third-party trackers than our simple policy of changing pseudonyms upon every newly entered URL (i.e., per browsing session). However, the difference is small and the amount of information gathered by first-party tracking is almost identical. This is because users typically conduct multiple activities in a relatively short time span on a website, such as entering multiple search terms, before manually navigating on to a new site. In the case of user #5, time-based anonymization is, on average, even more effective than the policy of a pseudonym *per browsing session*. This particular user conducted long web sessions with little activity, such as reading technical manuals and watching online movies.

7.3 Summary

Our results show that our system is practical and can be deployed today. Furthermore, it is evident that configurable privacy policies are desirable. Because users have differing expectations for privacy and functionality (e.g., suggestions or directed advertising), there is no one-size-fits-all policy. Even browsing behavior affects this choice, as a slower-paced browsing session can benefit just as much from a timeout-based privacy policy as from a domain-based or a link-based policy. Thus, a system that allows for more privacy control is beneficial.

8. DISCUSSION

Without Application Cooperation: For much of this paper, we rely on applications to create and enforce policies for connections that they generate. Unfortunately, legacy applications do not provide this support and may allow tracking. To fix this, we can sandbox the entire application. This sandboxing can be done with minimal OS-level support, or can be done by adding pseudonym support to VM software and running applications within a VM. Alternatively, we can intercept traffic from the application and classify it at a lower level, but this potentially requires application-specific knowledge and cannot handle encrypted communication.

Browser Extension Usability: There are still plenty of improvements we can make to our browser extension that would improve the usability of our system. These might include an indication of what pseudonym is being used for a particular resource or website and a way to change the pseudonym and reload the page. Though a user may have many pseudonyms, we can make management and reuse easier by allowing users to name important pseudonyms or by suggesting pseudonyms if they have previously logged into the website. We can also decrease the need for manual exceptions and unnecessarily restrictive policies with whitelists/blacklists. These concerns are left for future work.

9. RELATED WORK

There have been a number of studies that characterize web tracking [14, 15, 23]. Additionally, Eckersley [7], Yen *et al.* [27], and Nikiforakis *et al.* [20] point out that implicit information, such as user-agent or system fonts information, along with IP address can effectively fingerprint hosts.

As privacy concerns get more attention, many privacy-protecting mechanisms are being proposed and deployed. Modern browsers have private browsing modes, but these are not very effective at mitigating web tracking [4]. Chrome and Firefox support multiple profiles to separate identities, but require manual control and still leak implicit information. In addition to these, privacy-protecting extensions have also been proposed. For example, Milk [25] prevents third-party trackers from building user profiles across web sites, and ShareMeNot [23] and Kontaxis *et al.* [11] provide protection against tracking using social media widgets. The above mechanisms are somewhat effective within their threat models, however, they cannot prevent adversaries from tracking using IP addresses as they are all application-layer solutions.

Also related to this work, there are network- and overlay-layer systems that provide anonymity using onion routing [6, 16]. These proposals are orthogonal to our work as they do not address application-layer privacy and attacks. Our network-layer mechanism provides a subset of the properties provided by these proposals, but is simpler and more efficient. Additionally, our solution focuses on creating unlinkable pseudonyms and giving users more contol over linkability, which is difficult to achieve with Tor as it uses a limited number of exposed IP addresses (one from the current Tor exit node at a given time). Torbutton [21] is a browser extension that addresses application-level security and privacy concerns, such as separating the cookie store between Tor and non-Tor sessions, but provides only limited control over linkability. Privoxy [3] is a web proxy with comprehensive filtering capabilities and can modify web requests based on user configuration. Unfortunately, since it is not attached to the browser, it cannot track activities closely. Moreover, Privoxy cannot handle HTTPS requests as the connections are encrypted. Our system can potentially include most features of Privoxy in our gateway service. The Address Hiding Protocol [22] proposes to assign a random IP address (still from the same ISP) to hide the original source from the destination. RFC 3041 proposes a privacy extension to stateless IPv6 address autoconfiguration that lets hosts generate temporary IPv6 addresses by hashing interface identifiers. These schemes do not address application-

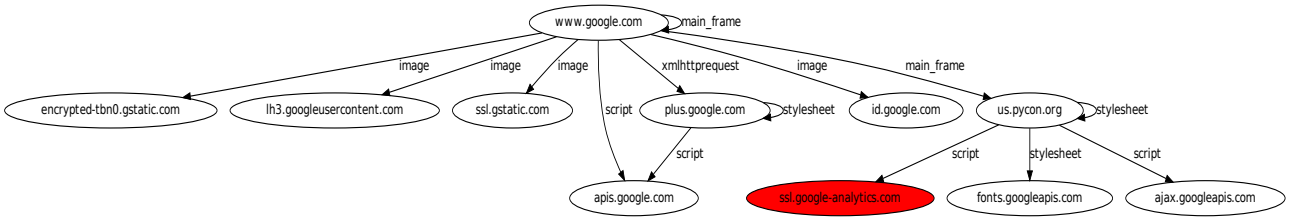

Figure 11: Example collusion graph generated from a single web session. Nodes represent domains. Edges show which domains caused HTTP requests to which other domains. Edge labels denote the type of request. Red nodes are web trackers.

layer issues nor do they provide for efficient routing of packets to randomly assigned addresses [18].

10. CONCLUSION

This paper presents an abstraction called a pseudonym, where each device and therefore users are able to control and use many, indistinguishable identities. The pseudonym abstraction gives users control over which activities can be linked at remote services and which cannot. We have designed a cross-layer architecture that exploits the ample IPv6 address space and provides application-layer mechanisms for management. In particular, we dive into the design/implementation of pseudonyms in an especially important application: the web browser. Our design provides the ability for users to choose expressive policies for controlling the privacy/functionality tradeoff on the web. Our prototype system consists of a browser extension and a gateway proxy and it proves the feasibility of our concept.

11. ACKNOWLEDGEMENTS

We gratefully acknowledge our shepherd John C.S. Lui and the anonymous SIGCOMM reviewers. We thank Franzi Roesner and Tadayoshi Kohno for their thoughtful comments. This work was partially funded by Cisco and NSF Grant CNS-1040663.

12. REFERENCES

[1] Collusion. `http://bit.ly/XZgEM6`.
[2] Cookiepie. `http://bit.ly/11e3HFE`.
[3] Privoxy. `http://privoxy.org`.
[4] AGGRAWAL, G., BURSZTEIN, E., JACKSON, C., AND BONEH, D. An analysis of private browsing modes in modern browsers. In *Usenix Security* (2010).
[5] CARPENTER, B. Internet transparency. RFC 2775, 2000.
[6] DINGLEDINE, R., MATHEWSON, N., AND SYERSON, P. Tor: The second-generation onion router. In *USENIX Security* (2004).
[7] ECKERSLEY, P. How unique is your web browser? In *Symposium on Privacy Enhancing Technologies* (2010).
[8] GOEL, S., ROBSON, M., POLTE, M., AND SIRER, E. G. Herbivore: A scalable and efficient protocol for anonymous communication. Cornell University Computing and Information Science TR2003-1890, 2003.
[9] HSIAO, H.-C., KIM, T. H.-J., PERRIG, A., YAMADA, A., NELSON, S. C., GRUTESER, M., AND MENG, W. LAP: Lightweight anonymity and privacy. In *IEEE Symposium on Security and Privacy* (2012).
[10] JANG, K., HAN, S., HAN, S., MOON, S., AND PARK, K. SSLShader: cheap SSL acceleration with commodity processors. In *NSDI* (2011).
[11] KONTAXIS, G., POLYCHRONAKIS, M., KEROMYTIS, A. D., AND MARKATOS, E. P. Privacy-preserving social plugins. In *USENIX Security* (2012).
[12] KOUNAVIS, M. E., KANG, X., GREWAL, K., ESZENYI, M., GUERON, S., AND DURHAM, D. Encrypting the internet. In *SIGCOMM* (2010).
[13] KRISHNAMURTHY, B., NARYSHKIN, K., AND WILLS, C. E. Privacy leakage vs. protection measures: the growing disconnect. In *WPES* (2011).
[14] KRISHNAMURTHY, B., AND WILLS, C. E. Generating a privacy footprint on the internet. In *IMC* (2006).
[15] KRISHNAMURTHY, B., AND WILLS, C. E. Privacy diffusion on the web: a longitudinal perspective. In *WWW* (2009).
[16] LIU, V., HAN, S., KRISHNAMURTHY, A., AND ANDERSON, T. Tor instead of IP. In *HotNets* (2011).
[17] MIKIANS, J., GYARMATI, L., ERRAMILLI, V., AND LAOUTARIS, N. Detecting price and search discrimination on the Internet. In *HotNets* (2012).
[18] NARTEN, T., DRAVES, R., AND KRISHNAN, S. Privacy extensions for stateless address autoconfiguration in IPv6. RFC 4941, 2007.
[19] NARTEN, T., HUSTON, G., AND ROBERTS, L. IPv6 address assignment to end sites. RFC 6177, 2011.
[20] NIKIFORAKIS, N., KAPRAVELOSY, A., JOOSEN, W., KRUEGELY, C., PIESSENS, F., AND VIGNAY, G. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE Symposium on Security and Privacy* (2013).
[21] PERRY, M. Torbutton design documentation. https://www.torproject.org/torbutton/en/design/index.html.en, 2011.
[22] RAGHAVAN, B., KOHNO, T., SNOEREN, A. C., AND WETHERALL, D. Enlisting ISPs to improve online privacy: IP address mixing by default. In *Symposium on Privacy Enhancing Technologies* (2009).
[23] ROESNER, F., KOHNO, T., AND WETHERALL, D. Detecting and defending against third-party tracking on the web. In *NSDI* (2012).
[24] SAVOLA, P. Mtu and fragmentation issues with in-the-network tunneling. RFC 4459, 2006.
[25] WALLS, R. J., CLARK, S. S., AND LEVINE, B. N. Functional privacy or why cookies are better with milk. In *HotSec* (2012).
[26] WILLS, C. E., AND TATAR, C. Understanding what they do with what they know. In *WPES* (2012).
[27] YEN, T.-F., XIE, Y., YU, F., YU, R. P., AND ABADI, M. Host fingerprinting and tracking on the web: Privacy and security implications. In *NDSS* (2012).