# Reducing Third Parties in the Network through Client-Side Intelligence

**Georgios Kontaxis**

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

**COLUMBIA UNIVERSITY**

2018

# ABSTRACT

## Reducing Third Parties in the Network through Client-Side Intelligence

Georgios Kontaxis

The end-to-end argument describes the communication between a client and server using functionality that is located at the end points of a distributed system. From a security and privacy perspective, clients only need to trust the server they are trying to reach instead of intermediate system nodes and other third-party entities. Clients accessing the Internet today and more specifically the World Wide Web have to interact with a plethora of network entities for name resolution, traffic routing and content delivery. While individual communications with those entities may some times be end to end, from the user's perspective they are intermediaries the user has to trust in order to access the website behind a domain name. This complex interaction lacks transparency and control and expands the attack surface beyond the server clients are trying to reach directly. In this dissertation, we develop a set of novel design principles and architectures to reduce the number of third-party services and networks a client's traffic is exposed to when browsing the web. Our proposals bring additional intelligence to the client and can be adopted without changes to the third parties.

Websites can include content, such as images and iframes, located on third-party servers. Browsers loading an HTML page will contact these additional servers to satisfy external content dependencies. Such interaction has privacy implications because it includes context related to the user's browsing history. For example, the widespread adoption of "social plugins" enables the respective social networking services to track a growing part of its members' online activity. These plugins are commonly implemented as HTML iframes originating from the domain of the respective social network. They are embedded in sites

users might visit, for instance to read the news or do shopping. Facebook's Like button is an example of a social plugin. While one could prevent the browser from connecting to third-party servers, it would break existing functionality and thus be unlikely to be widely adopted. We propose a novel design for privacy-preserving social plugins that decouples the retrieval of user-specific content from the loading of third-party content. Our approach can be adopted by web browsers without the need for server-side changes. Our design has the benefit of avoiding the transmission of user-identifying information to the third-party server while preserving the original functionality of the plugins.

In addition, we propose an architecture which reduces the networks involved when routing traffic to a website. Users then have to trust fewer organizations with their traffic. Such trust is necessary today because for example we observe that only 30% of popular web servers offer HTTPS. At the same time there is evidence that network adversaries carry out active and passive attacks against users. We argue that if end-to-end security with a server is not available the next best thing is a secure link to a network that is close to the server and will act as a gateway. Our approach identifies network vantage points in the cloud, enables a client to establish secure tunnels to them and intelligently routes traffic based on its destination. The proliferation of infrastructure-as-a-service platforms makes it practical for users to benefit from the cloud. We determine that our architecture is practical because our proposed use of the cloud aligns with existing ways end-user devices leverage it today. Users control both endpoints of the tunnel and do not depend on the cooperation of individual websites. We are thus able to eliminate third-party networks for 20% of popular web servers, reduce network paths to 1 hop for an additional 20% and shorten the rest.

We hypothesize that user privacy on the web can be improved in terms of transparency and control by reducing the systems and services that are indirectly and automatically involved. We also hypothesize that such reduction can be achieved unilaterally through client-side initiatives and without affecting the operation of individual websites.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Distributed systems are better served when high level functionality is implemented at their endpoints rather than the middle [95]. From a security and privacy perspective, clients only need to trust the server they are trying to reach instead of intermediate system nodes and other third-party entities. Clients accessing the Internet today and more specifically the World Wide Web have to interact with a plethora of network entities for name resolution, traffic routing and content delivery. While individual communications with those entities may some times be end to end, from the user's perspective they are intermediaries the user has to trust in order to access the website behind a domain name. This complex interaction lacks transparency and control and expands the attack surface beyond the server clients are trying to reach directly. We draw the distinction between second and third parties where the former operate under the authority of the primary service users are trying to reach. For example, when a website delegates part of its content or functionality to a service operating under that website's domain the service is considered a second party and the users' trust towards the website extends to that service. Content delivery networks and same-site analytics services can be instances of second parties. On the other hand, services under their own distinct domains with symbiotic relationships to the website are third parties and users need to trust them separately. Routing networks on the Internet and cross-site social or advertising services are instances of third parties.

A client communicating with a web server exchanges traffic over a series of distinctly owned networks spanning organizational and national borders. Unless the client agrees on an end-to-end encrypted transport protocol with the server, all intermediate networks are effectively part of the conversation between the two endpoints as they have the ability to monitor and alter the information exchanged. End-user Internet service providers (ISPs), including popular ones such as AT&T and Verizon, inject advertisements [93] and tracking headers [10] [4] in the unencrypted HTTP traffic of their customers. They also tamper with SMTP traffic [27] to disable its opportunistic encryption. Intelligence agencies eavesdrop [47] on unencrypted traffic to piggyback on HTTP cookies for the purpose of tracking individual users. In addition, they impersonate popular Internet services through man-on-the-side attacks on unencrypted network paths for user exploitation and surveillance. Finally, network adversaries [90] inject JavaScript code into unencrypted HTTP traffic in transit to launch denial of service attacks.

Secure protocols, such as HTTPS, make the information exchanged inaccessible by intermediate nodes. This effectively abstracts a multi-party traffic exchange to a two-party conversation. Recent efforts to make HTTPS more affordable [6] and easy to deploy [1] are steps towards the right direction. Unfortunately a significant portion of web traffic still traverses the network without any security. In this dissertation we evaluate the security of 10,000 popular websites and find that only 30% support HTTPS. Translating HTTPS support to user impact is an open challenge given different browsing patterns among users and the subjective nature of private content served from individual sites. We therefore focus on service configurations rather than the amount of traffic or distinct users to understand how developers are building and maintaining the ecosystem. A similar study [91] focusing on connections finds that 50% are done over HTTPS. We examine HTTPS-capable sites and discover that 53% let their visitors default to plain-text HTTP. Overall only 56 of the 10,000 sites fully protect their users through a combination of HTTPS and HSTS preloading. Even when HTTPS is available implementation vulnerabilities threaten its security. Recent attacks such as FREAK [60], POODLE [85], Heartbleed [54], the work of AlFardan et al. [58] and BEAST [64] have impacted hundreds of thousands of Internet services. To make matters worse many of these services remain vulnerable months after the disclosure of the attack and some of them will never be patched. In the case of FREAK, 45% of the affected servers remained vulnerable 9 months post disclosure [50].

The functionality offered by modern websites introduces additional privacy concerns that cannot be addressed by the use of encryption between clients and servers. Websites often include content, such as images and iframes, located on third-party servers. Browsers loading an HTML page will contact these additional servers to satisfy external content dependencies. Such interaction has privacy implications because it includes context related to the user's browsing history. It also lacks transparency since the browser's address bar only displays the address of the website visited directly. "Social plugins" enable websites to offer personalized content by leveraging the social graph, and allow their visitors to seamlessly share, comment, and interact with their social circles [19]. These plugins are provided by services such as Facebook and are embedded by developers in the form of iframes in websites users might visit, for instance to read the news or do shopping. Indicatively, as of June 2012, more than two million websites had incorporated some of Facebook's social plugins, while more than 35% of the top 10,000 websites include Like buttons—a percentage three times higher than just one year before [53]. Roesner et al. [94] show that in 2012 30% of the 100 most popular domains embed Facebook plugins. Englehardt et al. [69] show that in 2016 35% of 1 million popular websites feature Facebook plugins. Their findings demonstrate that the presence of social plugins in popular websites has been consistent for almost a decade. [8] They also show that social and advertising networks are the only third parties with a large presence on the web. When visiting a web page, the browser will load such third-party content by connecting to Facebook and in the process will transmit the URL of the visited page along with any cookies available for the third-party domains. Facebook cookies uniquely identify a user by name and the ubiquity of these plugins enables the social network to learn a growing part of the user's browsing history. In recent years Facebook has started [32] taking advantage of social plugins to covertly profile its users. To make matters worse, users lack adequate control since browsers interact with third parties automatically when loading a page. Users could prevent the browser from connecting to these third-party servers. Such practice has already been studied in the context of advertising networks by the research community [71, 102] and millions of users have adopted browser add-ons that employ blacklists to prevent contact with such servers. [2] However social plugins are a unique case. Given that they bundle functionality with user tracking, preventing them from loading would break the functionality they offer and thus any such solution would be unlikely to be widely adopted by users.

## 1.1 Hypothesis

Given current studies [69, 94] advertising and social networking services are the primary third party content providers privy to the user's browsing history. Given the existing mitigations [71,102] against online advertisers we hypothesize that by addressing the privacy concerns of social plugins user browsing history won't leak to any third party found in more than 10% of popular websites. Subsequently for third parties to reconstruct the view social networks have of the user's browsing activity they would have to collude among themselves. Additionally, domain name resolution, traffic routing and content delivery are the three essential services users have to rely on to browse the web. We hypothesize that by protecting vulnerable traffic, such as plaintext TCP, from routing intermediaries users will only need to trust DNS services and the website offering the content they are trying to access. We also hypothesize that such reduction can be achieved unilaterally through client-side initiatives and without requiring changes to the third-party services or networks. We draw the distinction between second and third parties where the former are delegates of the website's content or functionality and operate under its authority. Content delivery networks and same-site analytics services can be instances of second parties. The origin of same-site analytics services is the website's domain in contrast to social plugins which use the origin of the social networking service and can extend their user tracking state across sites. Similarly, a website's domain name may resolve to the IP addresses of a content delivery network which may also carry a TLS certificate for that website.

## 1.2 Thesis Statement

This thesis argues that the third parties involved when clients access the web can be effectively reduced by moving their functionality to the endpoints. Specifically clients can leverage infrastructure-as-a-service platforms to deploy a secure network overlay that enables them to reach a web server in a way that reduces the networks involved. We consider that by routing vulnerable traffic through the network overlay and therefore not exposing it to the respective transit networks we are effectively removing them from the path. Ideally the edges of the network overlay are the client and the web server and thus all network intermediaries are effectively removed. Additionally we argue that clients have the technical capability to locally implement the content personalization offered by third-party social plugins thereby eliminating the communication with the social network when loading a web page. By eliminating communication with the social network when loading a page the user's browsing history is not leaked.

## 1.3 Contributions

- We identify the clustering of Internet services inside cloud providers and propose strategically establishing encrypted tunnels to their networks to avoid exposing plain-text traffic to the Internet.
- We define the following security-oriented metrics for routing traffic through our architecture; (a) Number of autonomous systems (AS) plain-text traffic must traverse to reach an Internet service. Ideally zero because the service is in the same network as a tunnel exit. (b) Involvement of a particular trusted or untrusted AS.
- We implement and evaluate Topology-aware Network Tunnels (TNT) as IP routing software. We address challenges in network measurements and system integration.
- We propose a novel design for privacy-preserving social plugins that i) prevents the social network from tracking its members' web activity, and ii) provides the same functionality as existing plugins with no compromises in content personalization.
- We implement and evaluate SafeButton, a Firefox extension that provides privacy-preserving versions of Facebook's social plugins.
- We describe in detail a pure JavaScript implementation of our design that can be offered by existing social networking services as a transparent service to their members.

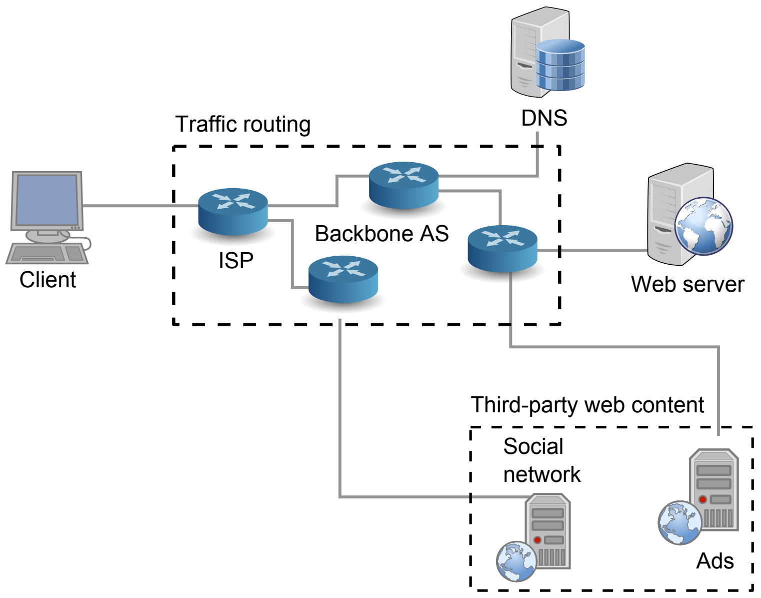

Figure 1.1: A plethora of third-party entities are involved when users visit a website. From a privacy perspective, they have to trust networks and services beyond their intended destination.

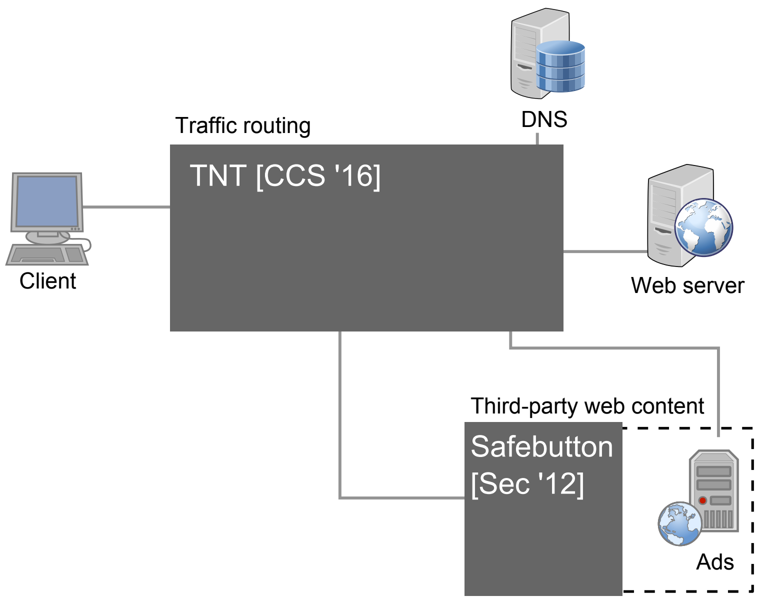

Figure 1.2: This dissertation proposes a privacy-preserving design for personalized third-party content as well as a secure transport overlay that effectively reduces the trusted networks involved.

# Chapter 2

# Related Work

## 2.1 Preventing Third-party Tracking on the Web

The privacy implications of third-party content when users browse the web have been identified early on [20, 21, 75]. Subsequent studies have tried to quantify the pervasiveness of tracking in general or among special groups of third parties including social networking services. Mayer et al. [83] highlight the threats against user privacy by the cross-site tracking capabilities of third-party web services. The authors detail a plethora of tracking technologies used by the embedded pages of advertisement, analytics, and social networking services. Their work demonstrates the high level of sophistication in web tracking technologies. Krishnamurthy et al. [79] study privacy leaks in online social networking services. They identify the presence of embedded content from third-party domains in the interactions of a user with the social network itself and stress that the combination with personal information inside a social network could pose a significant threat to user privacy. Roesner et al. [94] study the tracking ecosystem of third-party web services and discuss current defenses, including third-party cookie blocking. They identify cases where tracking services actively try to evade such restrictions by bringing themselves in a first party position, e.g., by spawning pop-up windows. Moreover, the authors present cases in which services are treated as first parties when visited directly and intentionally by the users, and at the same time appear embedded as third parties in websites, as is the case with social networking services and their social plugins. Overall, they conclude that current restrictions imposed

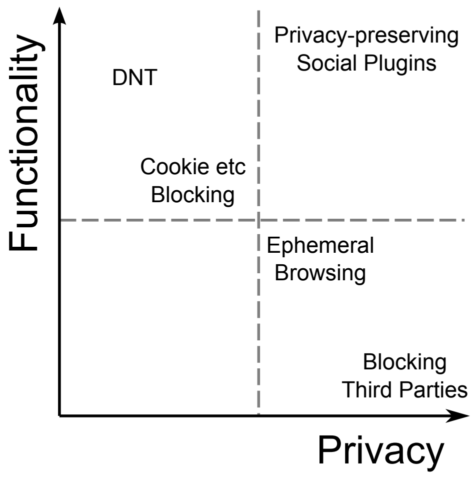

Figure 2.1: The privacy-functionality tradeoff when addressing third-party tracking.

by browsers against third-party tracking are not foolproof, and at the same time find more than 500 tracking services, some with the capability to capture more than 20% of a user's browsing behavior. Libert [80] carries out a large-scale study of the HTTP requests made when visiting a website and finds that in the majority of cases third-party services are involved. The author highlights that users lack visibility into the involvement of those third parties, points out that few large companies are responsible for most of the services and calls for additional scrutiny of their practices. Interestingly the most popular third parties are analytics and social networking services, which bundle user-facing functionality with invisible activity tracking.

From a privacy perspective, when third-party content providers do not have the ability to track users on the web, they are no longer involved in the user's browsing activity. Existing work tries to achieve that in three ways: 1) communicating the user's decision to opt out of tracking, 2) preventing the browser from sending user-identifying information to third-parties and 3) preventing the browser from connecting to them.

The W3C Platform for Privacy Preferences Project (P3P) [44] enables websites to express their privacy policy in a standardized format that is easy to parse by web browsers that later are supposed to communicate this information to users. Unfortunately it has not gained support due to the complexity involved in developers expressing their privacy policy and the effort required from users in understanding it. Moreover there is currently no way to force the adoption of P3P or enforce a website's P3P policy. Most web browsers no longer support P3P.

Do Not Track [7] is a browser technology which enables users to signal, via an HTTP header, that they do not wish to be tracked by websites they do not explicitly visit. Unfortunately there are no guarantees that such a request will be honored by the receiving site or not.

Given the limitations of policy-based privacy initiatives, existing research focuses on technical measures that attempt to identify and prevent tracking. Web browsers provide the option of preventing third parties from storing client-side state using HTTP cookies. Unfortunately there is no consistent behavior across browsers with some only preventing third parties from writing cookies. Third parties can work around such weak isolation by temporarily becoming first parties using a pop-up window or redirecting the entire page to their domain and back. Furthermore cookies are not the only method for storing client-side state which is why Jackson et al. [75] propose extending the same-origin isolation to the content cache, DNS cache and browsing history.

The plethora of ways to explicitly or implicitly store client-side state [76] has lead to an arms race. In response, a holistic approach to managing client-side state describes containerized browser instances either specific to a particular site [61] or in the form of an ephemeral browsing session [56]. In private or "incognito" windows the browser does not preserve any client-side state in an attempt to prevent long-term user tracking as opposed to addressing individual state-keeping mechanisms. Because of the rigid nature of this browsing mode it is an opt-in feature in all browsers that implement it.

In certain cases third parties are able to track users by merely being present on the pages they visit and without the need to set cookies or other state. Information available to them through the JavaScript API can result in a uniquely identifying set of values, a fingerprint, that is based not only on the software [67] but also the hardware [55, 86].

Tracking protection offered by Firefox [33] along with privacy-oriented browser extensions such as EFF's Privacy Badger [9] use a block list of known third-party trackers to prevent the browser from connecting to the respective domains. While stopping traffic exchange with tracking domains addresses even the most resilient forms of fingerprint-based tracking it creates a tradeoff between privacy and functionality.

Social networking services and other third-party providers of personalized content combine functionality users enjoy with the tracking of their browsing activity. Services such as Facebook that users visit both directly and by interacting with their content on other pages present challenges to the first and third-party isolation logic discussed above. Moreover they point out the usability issues with ephemeral browsing sessions when users have to log in to Facebook every time. Finally they highlight the need for privacy-preserving technologies that do not negatively impact functionality.

There has been significant work towards user control over the data provided to social networking services. The privacy implications of third-party applications installed in a user's profile within the social networking service are similar to the service being embedded in web pages. Existing research [68, 70, 99] tries to provide user control through a combination of encrypted data uploaded to the service for which the user has the key and information flow tracking to regulate access. Facecloak [82] shields a user's personal information from a social network and any third-party interaction by providing fake information to the social network and storing actual sensitive information in an encrypted form on a separate server. The authors in FlyByNight [81] propose the use of public key cryptography among friends in a social networking service so as to protect their information from a curious social provider and potential data leaks.

When social networking services are embedded in web pages they provide personalized content using information about the page they are embedded in and the user's social data. Privacy-preserving personalized content has been researched in the context of advertising. Recent work has focused on how to support personalized advertisements without revealing the user's personal information to the providing party. Adnostic [102] offers targeted advertising while preserving the user's privacy by having the web browser profile the user, through the monitoring of his browsing history, and inferring his interests. It then downloads diverse content from the advertising server and selects which part of it to display to the user. Similarly, RePriv [71] enables the browser to mine a user's web behavior to infer guidelines for content personalization, which are ultimately communicated to interested sites. Content personalized in the context of social networking services is a unique challenge.

Privacy research focusing on advertising prevents a website from building a profile for the user. In this thesis we focus on the nature of social networking services where the user already has a profile because of their direct interaction with the service. We then identify the need to decouple the identification step the user undergoes, to access their already existing social profile, from their subsequent requests for content personalization on pages the social networking service is embedded in.

Existing attempts to preserve user privacy are aligned with the general approach described above where browsers refrain from connecting to social networking services embedded as third parties. A series of browser add-ons exist [15,35] that block social plugins from the web pages a user visits by removing them or preventing them from loading, in a manner similar to what Adblock [2] does for advertisements. ShareMeNot [40,94] is a Firefox add-on that strips user cookies from a series of HTTP requests that the web browser issues to load social plugins. As a result, no user-identifying information is sent to the social networking service until the user explicitly interacts with the social plugin. The downside of this approach is that users are deprived of any personalized information offered by the plugin, e.g., the number and names of any of their friends that might have already interacted with a page. In other words, users view these social plugins as if they were logged out from the respective social network (or browsing in "incognito" mode).

In this dissertation we argue that any privacy mitigation should not come at the cost of the functionality third parties offer. Therefore we focus on providing the full content personalization of existing social plugins while protecting user privacy.

## 2.2 Preventing Third-party Tracking on the Network

Two hosts exchanging traffic over the Internet have to rely on a plethora of intermediate nodes and networks to forward their data. When the data is readable by anyone, intermediate nodes can observe and modify the data exchanged. Both client and server have to trust all intermediate parties as they are effectively communicating with all of them at the application level. Reis et al. [93] downloaded known HTML pages over HTTP and detected ISPs injecting advertisements in the page. Weaver et al. [104] using a client-server testbed

| Architecture | Content Security | Anonymity |
|---|---|---|
| TNT | Complete | No |
| Tor | Partial | Yes |
| VPN | Partial | No |

Table 2.1: Our proposal encrypts the entire path between the client and the destination network by optimizing tunnel placement and traffic routing. The same logic in Tor would compromise its anonymity properties.

found out that 14% of the measurements conducted showed evidence of a middlebox processing the traffic. They identified such proxies in all four major U.S. mobile carriers. Using the same testbed they discovered [5] that network operators, including a major U.S. ISP, inject erroneous DNS replies for non-existent domains to redirect users to pages featuring advertisements. Durumeric et al. [65] observed ISPs tampering with SMTP traffic to disable its opportunistic encryption. Nakibly et al. [87] identified content injection that cannot be attributed to edge ISPs and is more likely taking place at the Internet's backbone.

End-to-end encrypted protocols such as TLS [45] reduce the capability of intermediate parties to simple traffic forwarding and effectively offer a two-party interaction. When end-to-end encryption is not available alternative approaches attempt to mitigate the privacy implications.

To limit the exposure of their plain-text traffic some users connect to Virtual Private Network (VPN) servers offering encrypted tunnels between the client's device and some fixed point in the Internet beyond which traffic is unencrypted. While such services protect from a local network attacker, exposure to network adversaries might even increase as opposed to a direct route without the VPN service. Since VPN gateways are not optimized to be close to web servers user traffic might traverse more autonomous systems or even cross national borders, e.g., from a US gateway to a European server. Even VPNs with diverse gateways employ them without considering the destination of traffic.

Tor [63] is an anonymity network where traffic is encapsulated in layers of encryption and usually travels between three nodes before the original TCP/UDP packet exits to the

Internet. By design Tor attempts no correlation between the exit of an encrypted circuit and the destination of traffic. Sherr et al. [97] propose the introduction of performance metrics in anonymous routing systems to affect circuit selection. Even though their work focuses within the anonymity network one could propose extending it so that exit nodes consider the destination of traffic. That would compromise the anonymity Tor offers. Even if someone were to give up anonymity we argue that the Tor network cannot carry user traffic close to Internet services. Using the Tor network status protocol we studied 1010 exit nodes with the highest bandwidth consensus weight and found that only 1% of them was located within major cloud networks. Meek [49] and domain fronting in general set up HTTPS tunnels to the cloud and CDNs and use them as gateways to masquerade the client's connection to a blocked website or Tor bridge. The placement of gateways ignores the location of the website and prefers networks an adversary is unlikely to unblock. As a result, Tor serves a different purpose than this dissertation and the two architectures complement each other. Table 2.1 summarizes their differences. Compared to the work of Sherr et al. we offer a complete implementation, evaluate it end-to-end, and introduce security metrics for routing decisions.

Overlay networks have been used in the past to recover from link failures in the underlying infrastructure and achieve better end-to-end performance. Savage et al. [96] propose Detour, an overlay where its members periodically exchange performance metrics such as RTT and packet loss and base their routing decisions on them thereby bypassing the Internet's native algorithms. LASTor [57] is a similar idea for Tor. Andersen et al. [59] in RON use a similar architecture to quickly route around link failures. Both designs limit their scope to members of the overlay and cannot be used for availability or performance guarantees for the rest of the Internet. Participating nodes evaluate each other on a regular basis which is something that does not scale well to the number of Internet services. Gummadi et. al [74] use one-hop source routing (SOSR) to recover from link failures in well-connected parts of the Internet. They maintain an overlay of virtual routers and clients use them as proxies to probe and connect to arbitrary network destinations when their default route is unable to deliver traffic. Our work differs from SOSR in two fundamental ways. In SOSR the placement of routers is selected at random since this is more likely to offer alternative links to most network destinations. In contrast, this dissertation optimizes the placement of TNT routers on the edges of the Internet, inside cloud networks where Internet services form clusters. We also make routing decisions so as to carry encrypted traffic as close to the destination as possible while SOSR prioritizes finding any available path around a failed link. In terms of implementation we solve engineering challenges such as routing updates in the presence of active client flows while SOSR reroutes traffic that is already failing. LASTor is a modified Tor client which uses a static AS-level map of the Internet to predict network paths and avoid circuits with the same AS at its edges. Astoria [88] follows a similar approach. In this dissertation we carry out data plane measurements to reliably construct the network path to a destination and reevaluate the path over time to account for routing changes. Moreover both LASTor and Astoria focus on diversifying the ASes involved in the edges of a Tor path. We minimize the length of the Internet path from a client to a server which is a different objective, optionally avoiding specific ASes.

# Chapter 3

# Privacy-Preserving Social Plugins

## 3.1 Overview

The design of privacy-preserving social plugins is driven by two key requirements: i) provide identical functionality to existing social plugins in terms of content personalization and user interaction, and ii) avoid the transmission of user-identifying information to the social networking service before any user interaction takes place. The first requirement is necessary for ensuring that users receive the full experience of social plugins, as currently offered by the major social networking services. (SNS) Existing solutions against user tracking do not provide support for content personalization, and thus are unlikely to be embraced by SNSes and content providers. The second requirement is necessary for preventing SNSes from receiving user-identifying information whenever users merely view a page and do not interact with a social plugin.

We consider as user-identifying information any piece of information that can be used to directly associate a social plugin instance with a user profile on the SNS, such as a cookie containing a unique user identifier. The IP address of a device or a browser fingerprint can also be considered personally identifying information, and could be used by a shady provider for user tracking. However, the accuracy of such signals cannot be compared to the ability of directly associating a visit to a page with the actual person that visits the page, due to factors that introduce uncertainty [92] such as DHCP churn, NAT, proxies, multiple users using the same browser, and other aspects that obscure the association of a device with the actual person behind it. Users can mitigate the effect of these signals to their privacy by browsing through an anonymous communication network [63], and ensuring that their browser has a non-unique fingerprint [67].

When viewed in conjunction, the two requirements seem contradicting. Content personalization presumes knowledge of the person for whom the content will be personalized. Nevertheless, the approach we propose satisfies both requirements, and enables a social plugin instance to render personalized content without revealing any user-identifying information to the SNS.

## 3.2 Design

We examine the nature of current plugins and define it as the intersection of the user's social data, public data from the social graph and the content of the website the plugins are embedded in. We define user data as any data contributed by the user to the social network as well as additional data available to the user, for example, when a friend uploads content on the social graph that is accessible by the user. Specifically we focus on social plugins offered by Facebook and verify that their operation aligns with the above definition. We empirically confirm that social plugins offered by other popular social networks, specifically Twitter and Google+, also follow this design. We present our analysis on Facebook plugins in Section 3.4.

Social plugins present the user with two different types of content: private information, such as the names and pictures of friends who like a page, and public information, such as the total number of "likes." The main idea behind our approach is to maintain a local copy of all private information that can possibly be needed for rendering any personalized content for a particular user, and query the social networking service only for public information that can be requested anonymously.

This approach satisfies our first requirement, since all the required private information for synthesizing and presenting personalized content is still available to the social plugin locally, while any missing public information can be fetched on demand. User interaction is not hindered in any way, as user actions are handled in the same way as in existing social plugins. Our second requirement is also satisfied because all communication of a privacy-preserving social plugin with the SNS for loading its content does not include any user-identifying information. Only public information about the page might be requested, which can be retrieved anonymously.

The whole process is coordinated by the Social Plugin Agent, which runs in the context of the browser and has three main tasks: i) upon first run, gathers all private data that might be needed through the user's profile and social circle, and stores it in a local DataStore, ii) periodically, synchronizes the DataStore with the information available online by adding or deleting any new or stale entries, and iii) whenever a social plugin is encountered, synthesizes and presents the appropriate content by combining private, personalized information from the local DataStore and public, non-personalized information through the SNS. Maintaining a local copy of the user's social information is a continuous process, and takes place transparently in the background. Once all necessary information has been mirrored during the bootstrapping phase, the DataStore is kept up to date periodically.

Using the Facebook Like button as a example, the private information that must be stored locally for its privacy-preserving version should suffice for properly rendering any possible instance of its personalized content for any third-party page the user might encounter. This can be achieved by storing locally all the "likes" that all of the user's friends have ever made, as well as the names and thumbnail pictures of the user's friends. Note that all the above information is available through the profile history of the user's friends, which is always accessible while the user is logged in.

Although keeping all this state locally might seem daunting at first, as we demonstrate in Sec. 3.4.2, the required space for storing all the necessary private information for privacy-preserving versions of all Facebook's existing social plugins is just 5.4MB for the typical case of a user with 190 friends, and 145MB for an extreme case of a user with 5,000 friends. No information that is not accessible under the user's credentials is ever needed, and daily synchronization typically requires the transmission of a few kilobytes of data. We consider the need to distribute viral content an atypical case and defer work towards an adaptive synchronization policy to the future. From a design perspective our proposal could model the user's behavior by performing state synchronization as frequently as necessary while

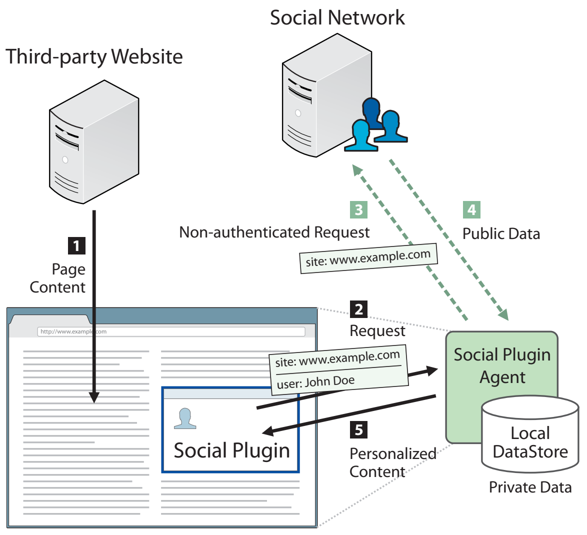

Figure 3.1: The loading phase of privacy-preserving social plugins. When a social plugin is encountered (1), the Social Plugin Agent intervenes between the plugin and the SNS (2). The agent requests (3) and receives (4) only publicly accessible content, e.g., the page's total number of "likes," without revealing any user-identifying information to the SNS. The agent then combines this data with personalized information that is maintained locally, and presents the unified content to the user (5).

prioritizing data from friends the user is more likely to interact with.

Continuing with the Like button as an example, Fig. 3.1 illustrates the process of rendering its privacy-preserving version. Upon visiting a third-party page, the Social Plugin Agent requests from the SNS the total number of "likes" for that particular page, without providing any user-identifying information (step 3). In parallel, it looks up the URL of the page in the DataStore and retrieves the names and pictures of the friends that have liked the page (if any). Once the total number of "likes" arrives (step 4), it is combined with the local information and the unified personalized content is presented to the user (5).

Further optimizations are possible to avoid querying for non-personalized content at load time. Depending on the plugin and the kind of information it provides, public information for frequently visited pages can be cached, while public information for highly popular pages can be prefetched. For example, information such as the total number of "likes" for a page the user visits several times within a day can be updated only once per day to

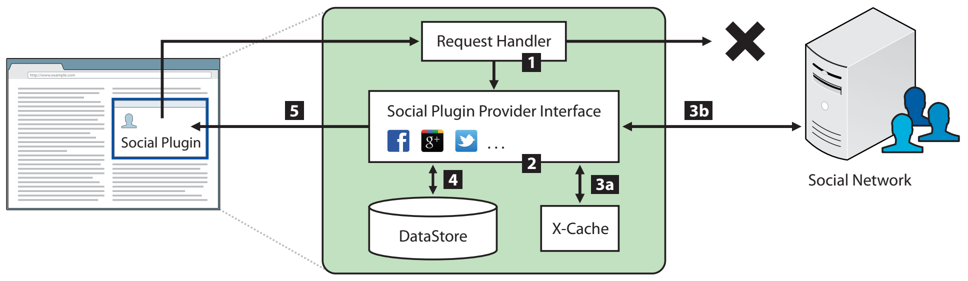

Figure 3.2: Overall architecture of SafeButton. A Request Handler (1) intercepts the HTTP requests of social plugins. Privacy-preserving implementations of the supported plugins (2) combine public remote data (3b), which can be cached in the X-Cache for improving network performance (3a), and private data from the user's social circle, which is maintained locally in the DataStore (4), and deliver the same personalized content (5) as the original plugins.

give an approximation of the page's popularity among people the user is not friends with. This allows the Social Plugin Agent to occasionally serve the Like button using solely local information. Similarly, the SNS could regularly push to the agent the total number of "likes" for the top 10K most "liked" pages. In both cases, the elimination of any network communication on every cache hit not only reduces the rendering time, but also protects the user's browsing pattern even further.

## 3.3 Implementation

To explore the feasibility of our approach we have implemented SafeButton, an add-on for Firefox (version 7.0.1) that provides privacy-preserving versions of existing social plugins. SafeButton is written in JavaScript and XUL [31], and relies on the XPCOM interfaces of Firefox to interact with the internals of the browser. Figure 3.2 provides an overview of SafeButton's main components, which are described below. A detailed description of how the components are put together to handle a Like button is provided at the end of this section.

**Request Handler** The main task of the Request Handler is to intercept the HTTP requests of a social plugin at load time, and hand off the event to an appropriate callback handler function. The requests are intercepted using a set of filters based on signatures that capture the target URL of each plugin. These signatures are received from the Social Plugin Provider Interface, along with the callback handlers that should be invoked whenever a filter is triggered. The Request Handler provides as an argument to these callbacks a reference to the DOM of the page that contains the social plugin that triggered the filter.

We have implemented the Request Handler by registering an observer for HTTP requests (`httpon-modify-request` notification) using XPCOM's `nsIObserverService`. This allows the inspection code to lie inline in the HTTP request creation process, and either intercept and modify requests (e.g., by stripping HTTP cookies or other sensitive headers), or drop them entirely when necessary.

**Social Plugin Provider Interface** The Social Plugin Provider Interface serves as an abstraction between the Request Handler and different Provider Modules that support the social plugins offered by different social networking services. This extensible design enables more networks and plugins to be supported in the future. In the current version of SafeButton, we have implemented a Provider Module for the social plugins offered by Facebook. We take advantage of the Graph API [17] to download the user's private social information that needs to be stored locally, and access any other public content on demand. Other social networks, for example Twitter and Google+, that offer similar plugins also provide API for developer's to access user data and interact with the service. Note that our proposal does not depend on the presence of an API as our design leverages data already available to users when they interact with the service. We should stress that, although an option, we do not employ any kind of scraping to extract information available to users through Facebook's web interface. We argue that any effort from a social network to terminate all avenues for users to access their own data would reduce transparency and counter any claims it has made in favor of user privacy.

A Provider Module for a SNS consists of: i) the signatures that will be used by the Request Handler for intercepting the HTTP requests of the platform's social plugins, ii) the callback handler functions that implement the core functionality of each social plugin based on local and remote social information, and iii) the necessary logic for initializing the

DataStore and keeping it up to date with the information that is available online.

Each callback function implements the core functionality for rendering a particular social plugin. Its main task is to retrieve the appropriate private social data from the DataStore, request any missing public data from the SNS (without revealing any user-identifying information), and compile the two into the personalized content that will be displayed. The function then updates the DOM of the web page through the page reference that was passed by the Request Handler.

**DataStore** The DataStore keeps locally all the private social data that might be required for rendering personalized versions of any of the supported social plugins. All information is organized in a SQLite database that is stored in the browser's profile folder for the user that has installed SafeButton. Upon first invocation, SafeButton begins the process of prefetching the necessary data. This process takes place in the background, and relies on the detection of browser idle time and event scheduling to operate opportunistically without interfering with the user's browsing activity.

In our implementation for Facebook, data retrieval begins with information about the user's friends, including each friend's name, thumbnail picture, and unique identifier in Facebook's social graph. Then, for each friend, SafeButton retrieves events of social activity such as the pages that a friend has liked or shared, starting with the oldest available event and moving onward. In case the download process is interrupted, e.g., if the users turns off the computer, it continues from where it left off the next time the browser is started.

Updating the DataStore is an incremental process that takes place periodically. Fortunately, the current version of the Graph API offers support for incremental updates. As we need to query for any new activity using a separate request for each friend (a Graph API function for multiple user updates would be welcome), we do so gracefully for each friend every two hours, or, if the browser is not idle, in the next idle period. We have empirically found the above interval to strike a good balance between the timeliness of the locally stored information and the incurred network overhead. In our future work, we plan to employ a more elaborate approach based on an adaptive update interval with exponential backoff so that a separate adaptive update interval can be maintained for different friend groups according to the frequency of their updates.

Note that we also need to address the consistency of the locally stored data with the corresponding data that is available online. For instance, friends may "like" a page and later on "unlike" it, thereby deleting this activity from their profile. Unfortunately, the Graph API currently does not offer support for retrieving any kind of removal events. Nevertheless, SafeButton periodically fetches the entire set of activities for each friend (at a much slower pace than the incremental updates), and removes any stale entries from the DataStore.

**X-Cache** The X-Cache holds frequently used public information and meta-information, such as the total number of "likes" for a page or the mapping between page URLs and objects in the Facebook graph. A hit in the X-Cache means that no request towards the social networking service is necessary for rendering a social plugin. This improves significantly the time it takes for the rendering process to complete, and at the same time does not reveal the IP address of the user to the SNS.

**Use Case: Facebook Like Button** Here we enrich the running case of the Facebook Like button from Sec. 3.2 with the technical details of the behavior of SafeButton's components, as shown by the relevant steps in Fig. 3.2.

Upon visiting a web page with an embedded Like button in the form of an iframe, the browser will issue an HTTP request towards Facebook to load and subsequently render the contents of that iframe. The Request Handler intercepts this request and attempts to match its URL against the set of signatures of the supported social plugins, which will trigger a match for the regular expression `http[s]?:\/\/www\.facebook\.com\/plugins\/like\.php`. Subsequently, the handler invokes the callback associated with this signature and passes as an argument the plugin's URL and a reference to the DOM of the page that contains the social plugin (step 1).

The first action of the callback function is to query X-Cache for any cached non-personalized information about the button and the page it is referring to. This includes the mapping between the page's URL and its ID in the Facebook graph, along with the global count of users who have "liked" the page (step 3a). In case of a miss, a request made through the Graph API retrieves that information (step 3b). The request is stripped from any Facebook cookies that the browser unavoidably appends to it. The response is then added to X-Cache for future reference. After retrieving the global count of users, the names (and if the developer has chosen so, the thumbnail pictures) of the user's friends that have liked the page are retrieved from the LocalStore (step 4).

Finally, the reference to the DOM of the embedding page (passed by the handler in step 1), is used to update the iframe where the original Like button would have been with exactly the same content (step 5).

## 3.4 Experimental Evaluation

### 3.4.1 Supported Facebook Plugins

In this section we discuss the social plugins offered by Facebook and evaluate the extend to which SafeButton can support them in respect to two requirements: i) user privacy, and ii) support for personalized content. Table 3.1 lists the nine social plugins currently offered by Facebook. For each plugin, we provide a brief categorization of its "view" functionality, i.e., the content presented to the user according to whether it is based on public (non-personalized) or private (personalized) information, as well as its "on-click" functionality, i.e., the type of action that a user can take.

Although SafeButton interferes with the "view" functionality of existing social plugins, it does not affect their "on-click" functionality, allowing users to interact normally as with the original plugins. As shown in Table 3.2, SafeButton currently provides complete support for seven out of the nine social plugins currently offered by Facebook.

The Like button and its variation, the Like Box, are fully functional; the count, names, and pictures of the user's friends are retrieved from the DataStore, while the total number of "likes" is requested on demand anonymously. The Recommendations plugin presents a list of recommendations for pages from the same site, with those made by friends appearing first. Recommendations from the user's friends are stored locally, so SafeButton can render those that are relevant to the visited site on top. The list is then completed with public recommendations by others, which are retrieved on demand. Similarly to the Like button, Facepile presents pictures of friends who have liked a page, and that information is already

| Facebook Social Plugin | Public Content | Personalized Content | User Action | 2012 | 2017 |
|---|---|---|---|---|---|
| Like Button | Total number of people that have liked the page | Names and pictures of friends that have liked the page | Like page | ✓ | ✓ |
| Send Button | - | - | Send content/page URL | ✓ | ✓ |
| Comments | List of user comments | Friends' comments appear on top | Post comment | ✓ | ✓ |
| Activity Feed | List of user activities (likes, comments, shared pages) | Friends' activities appear on top | - | ✓ | ✗ |
| Recommendations | List of user recommendations (likes) | Friends' recommendations appear on top | - | ✓ | ✗ |
| Like Box | Total number of people that have liked the Facebook Page, names and pictures of some of them, list of recent posts from the Page | Names and pictures of friends that have liked the page are shown first | Like page | ✓ | ✗ |
| Registration | - | User's name, picture, birthday, gender, location, email (prefilled in registration form) | Register | ✓ | ✗ |
| Facepile | - | Names and pictures of friends that have liked the page | - | ✓ | ✗ |
| Live Stream | User messages | - | Post message | ✓ | ✗ |
| Embedded content | Embed public posts, videos, etc. from Facebook in other websites | - | - | ✗ | ✓ |
| Follow Button | Total number of people that are following that person | Names and pictures of friends that are following that person | Subscribe to public updates | ✗ | ✓ |
| Quote | - | - | Select text on the website and share it on Facebook | ✗ | ✓ |
| Save | - | - | Save an external link in Facebook | ✗ | ✓ |
| Send | - | - | Send an external link to a friend in Facebook | ✗ | ✓ |
| Share | - | - | Post an external link and its preview on a friend's profile in Facebook | ✗ | ✓ |

Table 3.1: Public vs. Personalized content in Facebook's social plugins [19].

| Facebook Social Plugin | Exposed information during loading: Original | Exposed information during loading: SafeButton | Personalized Content with SafeButton |
|---|---|---|---|
| Like Button | IP addr. + cookies | IP addr. | Complete |
| Send Button | IP addr. + cookies | None | Complete |
| Comments | IP addr. + cookies | IP addr. | Partial[1] |
| Activity Feed | IP addr. + cookies | IP addr. | Partial[2] |
| Recommendations | IP addr. + cookies | IP addr. | Complete |
| Like Box | IP addr. + cookies | IP addr. | Complete |
| Registration | IP addr. + cookies | None | Complete |
| Facepile | IP addr. + cookies | IP addr. | Complete |
| Live Stream | IP addr. + cookies | IP addr. | Complete |

[1] *When all comments are loaded at once, all personalized content is complete. In case they are loaded in a paginated form, some of the friends' comments (if any) might not be shown in the first page.*

[2] *Some of the friends' comments (if any) might be omitted (access to comments is currently not supported by Facebook's APIs).*

Table 3.2: For 7 out of the 9 Facebook social plugins, SafeButton provides exactly the same personalized content without exposing any user-identifying information.

present in the DataStore. The Send, Register, and Login buttons do not present any kind of dynamic information, and thus can be rendered instantly without issuing any network request.

Similarly to the Recommendations plugin, content personalization in the Comments plugin consists of giving priority to comments made by friends. SafeButton retrieves the non-personalized version of the plugin, and reorders the received comments so that friends' comments are placed on top. When all comments for a page are fetched at once, the personalized information presented by SafeButton is fully consistent with the original version of the plugin. However, when comments are presented in a paginated form, only the first sub-page is loaded. The current version of the Graph API does not support the retrieval of comments (e.g., in contrast to "likes"), and thus in case friends' comments appear deeper than the first sub-page, SafeButton will not show them on top (a workaround would be to download all subsequent comment sub-pages, but for popular pages this would result in a prohibitive amount of data).

The Activity Feed plugin is essentially a wrapper for showing a mix of "likes" and comments by friends, and thus again SafeButton's output lacks any friends' comments. Note that our implementation is based solely on the functionality provided by the Graph API [17], and we refrain from scraping web content for any missing information. Ideally, future extensions of the Graph API will allow SafeButton to fully support the personalized content of all plugins. We discuss this and other missing functionality that would facilitate SafeButton in Sec. 3.6.

In recent years Facebook has retired plugins such as the Activity Feed and added new ones all of which can be fully supported by SafeButton as they are aligned with the existing plugins in the way they operate and the data they utilize to do so. For example the Follow button is another iteration of the Like button. The majority of the new plugins operate on public non-personalized content that is either presented to the user or is brought into the social network following some user action such as sharing the content. Retrieving non-personalized content without revealing the user's identity to Facebook is supported by SafeButton with the caveat of revealing the user's IP address. As discussed in Section 3.6 additional privacy mechanisms to anonymize the IP address can be combined with SafeButton.

### 3.4.2 Space Requirements

To explore the local space requirements of SafeButton, we gathered a data set that simulates the friends a user may have. Starting with a set of friends from the authors' Facebook profiles, we crawled the social graph and identified about 300,000 profiles with relaxed privacy settings that allow unrestricted access to all profile information, including the pages that person has liked or shared in the past. From these profiles, we randomly selected a set of 5,000—the maximum number of friends a person can have on Facebook [14].

To quantify the space needed for storing the required data from a user's social circle, we initialized SafeButton using the above 5,000 profiles. In detail, SafeButton prefetches the names, IDs, and photos of all friends, and the URLs of all pages they have liked or shared. Although we have employed a slow-paced data retrieval process (5sec delay between consecutive requests), the entire process for all 5,000 friends took less than 10 hours. For typical users with a few hundred friends, bootstrapping completes in less than a hour. As

| Data | 190 Friends | 5,000 Friends |
|---|---:|---:|
| Names, IDs of Friends | 10.5KB | 204.8KB |
| Photos of Friends | 463.4KB | 11.8MB |
| Likes of Friends | 4.6MB | 126.7MB |
| Shares of Friends | 318.4KB | 7.0MB |
| Total | 5.4MB | 145.7MB |
| Average (per friend) | 29.2KB | 29.7KB |

Table 3.3: Storage space requirements for the average case of 190 friends and the edge case of 5,000 friends.

already mentioned, users are free to use the browser during that time or shut it down and resume the process later.

Table 3.3 shows a breakdown of the consumed space for the average case of a user with 190 friends [103] and the extreme case of a user with 5,000 friends, which totals 5.4MB and 145.7MB, respectively. Evidently, consumed space is dominated by "likes," an observation consistent with the prevailing popularity of the Like button compared to the other social plugins. To gain a better understanding of storage requirements for different users, Fig. 3.3 shows the consumed space as a function of the number of friends, which as expected increases linearly.

We should note that the above results are specific for the particular data set, and the storage space might increase for users with more "verbose" friends. Furthermore, the profile history of current members will only continue to grow as time passes by, and the storage space for older users in the future will probably be larger. Nevertheless, these results are indicative for the overall magnitude of SafeButton's storage requirements, which can be considered reasonable even for current smartphones, while the storage space of future devices can only be expected to increase.

To further investigate the distribution of "likes," the factor that dominates local space, we plot in Fig. 3.4 the CDF of the number of "likes" of each user in our data set. The median user has 122 "likes," while there are some users with much heavier interaction: about 10% of the users have more than 504 "likes." The total number of "likes" was 1,110,000,

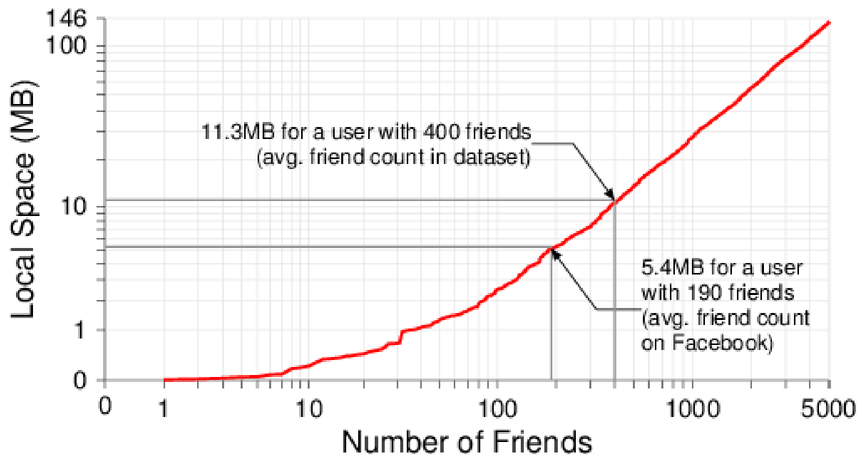

Figure 3.3: Local space consumption for the required information from a user's social circle as a function of the number of friends. For the average case of a user with 190 friends, SafeButton needs just 5.4MB.

i.e., 222 per user on average. This number falls within the same order of magnitude as previously reported statistics, which suggest that there are about 381,861 "likes" per minute on Facebook [48]. With a total population of about 901 million active users [16] at the time of the original measurement in 2012, this results in about 217 "likes" per user per year. These results indicate that our data set is not particularly biased towards excessively active or inactive profiles. We repeat the experiment in 2017 for a random population of users using the authors' profile as a starting point. Because of the high churn of online social connections as well as people deactivating their profiles there is little overlap with the population tested in 2012. Even though the two data sets cannot be compared directly data shows that the use of social plugins and consequently SafeButton's storage requirements remain similar.

Besides the storage of social data, SafeButton maintains the X-Cache for quick access to frequently used non-personalized data about a social plugin. To get an estimate about its size requirements, we visited the home pages of the top 1,000 websites according to `alexa.com` that contained at least one Facebook social plugin. About 82.4% of the identified plugins corresponded to a Like Button or Like Box, 14% to Facebook Connect, 3% to Recommendations, 0.5% to Send Button, and 0.1% to Facepile and Activity Box. After

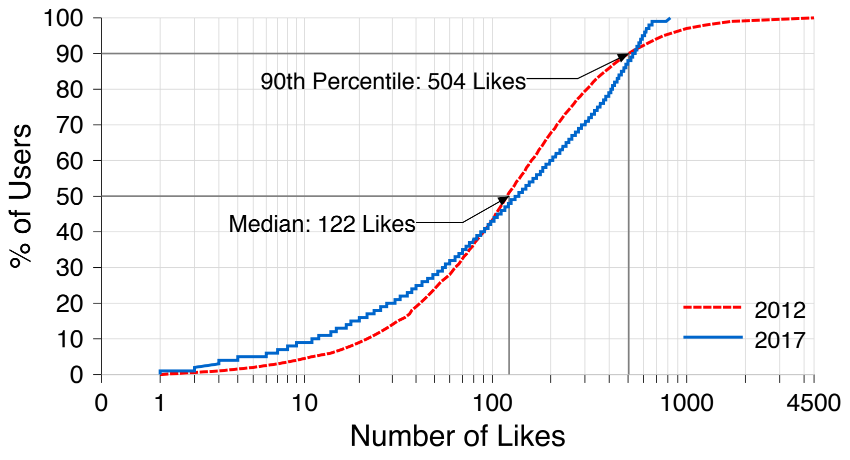

Figure 3.4: CDF of the number of "likes" of each user.

visiting all above sites, X-Cache grew to no more than 850KB, for more than 2,500 entries.

### 3.4.3 Speed

In this experiment, we explore the rendering time of social plugins with and without SafeButton. Specifically, we measured the time from the moment the HTTP request for loading the iframe of a Like button is sent by the browser, until its content is fully rendered in the browser window. To do so, we instrumented Firefox with measurement code triggered by `http-on-modify-request` notifications [28] and `pageshow` events [29]. We chose to measure the rendering time for the iframe instead of the entire page to eliminate measurement variations due to other remote elements in the page. This is consistent with the way a browser renders a page, since iframes are loaded in parallel with the rest of its elements.

We consider the following three scenarios: i) Firefox rendering a Like button unobstructed, and Firefox with SafeButton rendering a Like button when there is ii) an X-Cache miss or iii) an X-Cache hit. For the original Like button, we used a hot browser cache to cancel out loading times for any required external elements, such as CSS and JavaScipt files. Using SafeButton, visiting a newly or infrequently accessed web page will result in a miss in the X-Cache. For a Like button, this means that besides looking up the relevant information in the local DataStore, SafeButton must (anonymously) query Facebook to retrieve the total number of "likes." For frequently accessed pages, such personalized information will likely already exist in the X-Cache, and thus SafeButton does not place any network request at all.

Using a set of the first 100 among the top websites according to `alexa.com` that contain a Like button, we measured the loading time of the Like button's iframe for each site (each measurement was repeated 1,000 times). Figure 3.5 shows the median loading time across all sites for each scenario, as well as its breakdown according to the events that take place during loading. The rendering time for the original Like button is 351ms, most of which is spent for communication with Facebook. In particular, it takes 130ms from the moment the browser issues the request for the iframe until the first byte of the response is received, and another 204ms for the completion of the transfer. In contrast, SafeButton is much faster, as it needs 127ms for rendering the Like button in case of an X-Cache miss (2.8 times faster than the original), and just 24ms in case of an X-Cache hit (14.6 times faster), due to the absence of any network communication.

The difference in the response times for the network requests placed by the original Like button and SafeButton in case of an X-Cache miss can be attributed to the different API used and amount of data returned in each case. SafeButton uses the Graph API to retrieve just the total number of "likes," which is returned as a raw ASCII value that is just a few bytes long. In contrast, the original plugin communicates with a different endpoint from the side of Facebook, and fetches a full HTML page with embedded CSS and JavaScript content. While these two requests need a similar amount of time from the moment they are placed until the first response byte is received from the server, they differ by two orders of magnitude in terms of the time required to complete the transfer. Even if Facebook optimizes its own plugins in the future, we expect the rendering speed of SafeButton to be comparable in case of an X-Cache miss, and still much faster in case of an X-Cache hit.

### 3.4.4 Effectiveness

As presented in Sec. 3.2, we rely on a set of heuristics that match the target URL of each supported social plugin to intercept and treat them accordingly so as to protect the user's privacy. To evaluate the effectiveness and accuracy of our approach, we carried out

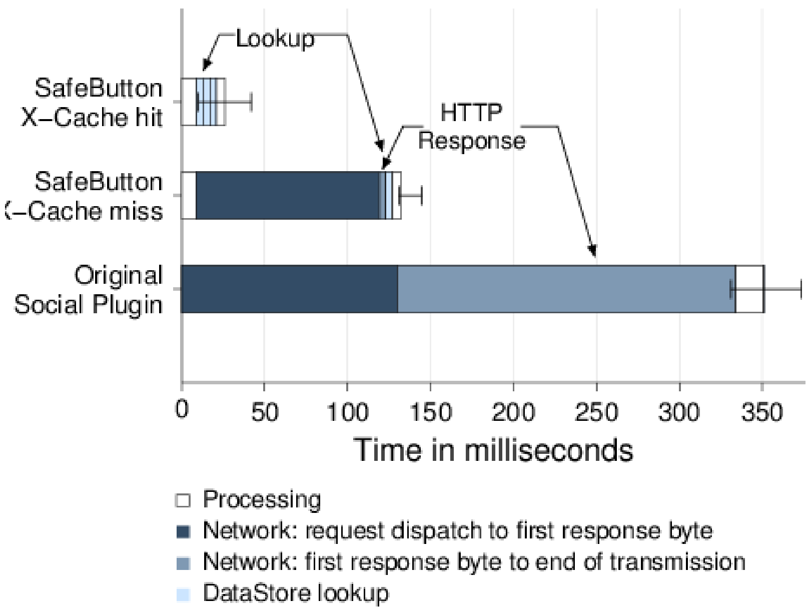

Figure 3.5: Loading time for a Like button with and without SafeButton. Even when the total number of "likes" is not available in the X-Cache, SafeButton is 2.8 times faster.

the following experiment. Using `tcpdump`, we captured a network trace of all outgoing communication of a test PC in our lab while surfing the web for a week through Firefox equipped with SafeButton. We then inspected the trace and found that no cookie was ever transmitted in any HTTP communication with `facebook.com` or any of its sub-domains.

This was a result of the following "fail-safe" approach. Besides the signatures of the supported social plugins, SafeButton inspects all communication with `facebook.com` and strips any cookies from requests initiated by third-party pages. Next, we performed the reverse experiment: using the same browser equipped with SafeButton, we surfed `www.facebook.com` and interacted with the site's functionality without any issues for a long period. Careful inspection of the log generated by SafeButton proved that no in-Facebook communication was hindered at any time.

### 3.4.5 Revisiting Social Plugins

We revisit social plugins in 2017 and find that both their nature and types of information stored in a user's social graph remain unchanged. The most prevalent data type is a numeric graph ID pointing to other objects. Objects can be simple items such as URLs or complex, such as user profiles, pointing to other objects. From the perspective of content personalization SafeButton still needs to store numeric IDs and URLs. We find that our storage requirements haven't changed.

While one might expect that the user population is 2017 has significantly greater storage needs because of additional data in their social profiles, from the perspective of personalizing external content this might be explained by Facebook's shift in strategy to bring content into Facebook rather than letting users interact with it on other websites. Indicative of this strategy is the fact that in the Facebook developer's page the Share button, which creates a copy of external content in Facebook, is now listed first, above the Like button. Further evidence to support this trend is the deprecation of plugins that exposed social data to external websites and the introduction of plugins that expose non-personalized data or bring external data into the social network. We argue that even though Facebook is steering users towards an in-network social experience SafeButton continues to benefit user privacy because Facebook in the last few years has openly discussed using social plugins as a way to profile users. In other words, even though it hasn't extended the personalized functionality of plugins it is increasingly depending on their presence to learn the browsing history of its users.

## 3.5 Privacy-Preserving Social Plugins as a Service: A Pure JavaScript Design

As many users are typically not aware of the privacy issues of social plugins, they are not likely to install any browser extension for their protection. For instance, NoScript [36], a Firefox add-on which blocks untrusted JavaScript code from being executed, has roughly just 2 million users based on 2017 data, and AdBlock [2], an add-on which prevents advertisement domains from loading as third parties in a web page, has about 14 million users. At the same time, Firefox has 500 million active users [34], which brings the adoption rate of the above security add-ons to 0.4% and 2.8%, respectively. For this reason, in this section we present a pure JavaScript implementation of privacy-preserving social plugins that could

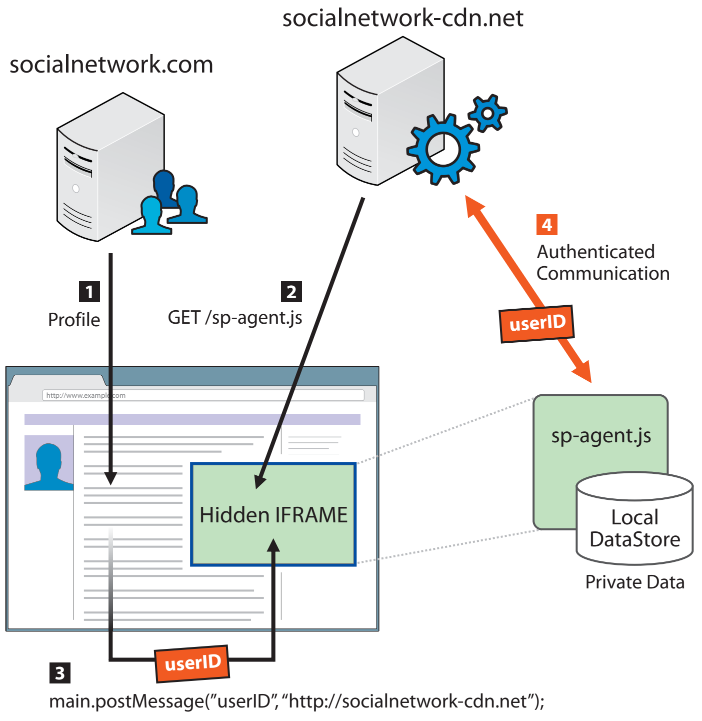

Figure 3.6: Privacy-preserving social plugins serviced by a SNS. Here: the loading of a social plugin in a third-party page. The code of the social plugin agent is always fetched from a secondary domain to avoid leaking cookies set by the primary domain of the SNS. The URL of the target page is passed via a fragment identifier so it is never transmitted to the SNS. The agent synthesizes and renders the personalized content of the social plugin.

be employed by social networking services themselves for the protection of their members. This design requires adoption and thus cooperation from social networks. By design, the same origin policy prevents unintended communication across domains. While a JavaScript implementation that can be hosted by websites embedding social plugins would obviate the need for users to manually install SafeButton, Facebook and other social networks will need to extend support so that it aligns with the same origin policy.

SafeButton has been motivated by the privacy concerns around the current implementation of social plugins. In 2010 Facebook claimed [38] that despite the leakage of browsing history it was not using plugins for tracking. The following proposal assumes an honest and not curious social network as Facebook claimed to be. Facebook has since reversed [13]

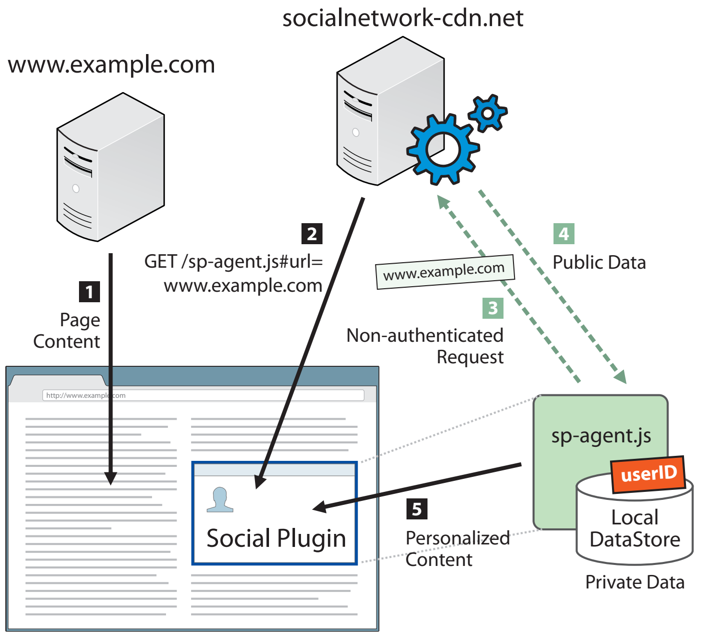

Figure 3.7: Privacy-preserving social plugins serviced by a SNS. Here: securely communicating the user's session identifier to the social plugin agent when logging into the SNS. Although the agent is hosted on a secondary domain, it receives and stores the identifier from the primary domain through the `postMessage` API, allowing it to place asynchronous authenticated requests for accessing the user's profile information.

its statements but still claims that users will be able to opt out from tracking. Our proposal still remains relevant as proof that privacy-preserving social plugins can be adopted by websites without the need for browser add-ons. In theory Facebook could offer our privacy-preserving version to users who opt out under their latest policy.

The use case would not be much different from now: web developers would still embed an iframe element that loads the social plugin from the SNS. However, instead of serving a traditional social plugin, the SNS serves a JavaScript implementation of a social plugin agent in respect to the design presented in Sec. 3.2. The agent then fetches personalized information from the browser's local storage, requests non-personalized information from the SNS, and renders the synthesized content according to the specified social plugin. The feasibility of the above design is supported by existing web technologies such as IndexedDB [26], which provide a JavaScript API for managing a local database, similar to the DataStore used in SafeButton.

The most challenging aspect of this implementation is to prevent the leakage of user-identifying information during the loading of a social plugin. If the iframe of the social plugin agent is hosted on the same (sub)domain as the SNS itself (e.g., `socialnetwork.com`), then the request for fetching its JavaScript code would also transmit the user's cookies for the SNS. At the same time, the agent would need to know the URL of the embedding page to personalize the social plugin's content. If the URL is passed as a parameter to that initial request, the situation is obviously as problematic as in current social plugins.

A solution would be to leave out the URL of the page from the request for loading the social plugin agent. However, there should be a way to communicate this information to the agent once its JavaScript code has been loaded by the browser. This can be achieved through a fragment identifier [52] in the URL from which the agent is loaded. Fragment identifiers come as the last part of a URL, and begin with a hash mark (#) character. According to the HTTP specification [24], fragment identifiers are never transmitted as part of a request to a server. Thus, during the loading of a social plugin in a third-party page, instead of passing an explicit parameter with the URL of the embedding page, as in `www.socialnetwork.com/sp-agent.js?url=<URL>`, it can be passed through a fragment identifier, as in `www.socialnetwork.com/sp-agent.js#<URL>`. The information about the URL of the visited page never leaves the browser, and remains accessible to the JavaScript code of the agent, which can then parse the hypertext reference of its container and extract the fragment identifier.

Unfortunately, this approach is still not secure in practice. The URL of the embedding page is usually also transmitted as part of the HTTP Referer [sic] header by most browsers. Therefore, even if we omit the target URL from the HTTP parameters of the request, the server will receive it anyway, allowing the SNS to correlate this information with the user's cookies that are transmitted as part of the same request.

To overcome this issue, the social plugin agent can be hosted on a secondary domain, different than the primary domain of the SNS, as also proposed by Do Not Track [7]. For instance, in this design the agent could be hosted under `socialnetwork-cdn.net` instead of `socialnetwork.com`, as shown in Fig. 3.6. This prevents the browser from appending the user's cookies whenever a social plugin is encountered (step 2), since its iframe will be served from a different domain than the one for which the cookies were set. The rest of the steps are analogous to Fig. 3.1.

Still, the social plugin agent must be able to issue authenticated requests towards the SNS for accessing the user's profile and retrieving the necessary private social information that must be maintained locally. This requires access to the user's cookies, and specifically to the identifier of the authenticated session that the user has with the SNS.

This challenge can be addressed through the `window.postMessage()` [30] JavaScript API which allows two different origins to communicate. When the user logs in on the SNS, the login page contains a hidden iframe loaded through HTTPS from the secondary domain on which the social plugin agent is hosted, as shown in Fig. 3.7 (step 2). The login page then communicates to the agent's iframe the session identifier of the user through `postMessage` (step 3). The iframe executes JavaScript code that stores locally the user identifier under its own domain, making it accessible to the plugin agent. The agent can then read the session identifier from its own local storage, and place authenticated requests towards the SNS for accessing the user's profile (step 4) and synchronizing the required information with the locally stored data. When the user explicitly logs out from the social networking site, the log out page follows a similar process to erase the identifier from the local storage of the agent.

In respect to supporting multiple users per browser instance and protecting the personal information stored locally, encryption can be employed to shield any sensitive information, such as the names or identifiers of a user's friends. In accordance with the communication of the session identifier described above, a user-specific cryptographic key can be communicated from the SNS to the social plugin agent. The plugin can then use this key to encrypt sensitive information locally. The key is kept only in memory. Each time the plugin agent loads, it spawns a child iframe towards the social networking site. The request for the child iframe will normally have the user's cookies appended. Finally, that child iframe, once loaded, can communicate via `postMessage` the encryption key back to the plugin agent.

## 3.6 Discussion

**Strict Mode of Operation** Although SafeButton does not send any cookies to the social networking service, it still needs to make non-authenticated requests towards the SNS to fetch public information for some social plugins (e.g., for Facebook plugins, the information shown in column "Public Content" in Table 3.1). These requests unavoidably expose the user's IP address to the SNS.

Some users might not feel comfortable with exposing their IP address to the SNS (even when no cookies are sent), as this information could be correlated by the SNS with other sources of information, and could eventually lead to the exposure of the users' true identity. For such privacy-savvy users, we consider a "paranoid" mode of operation in which SafeButton does not reveal the user's IP address to the social networking service when encountering a social plugin in a third-party page, by simply not retrieving any public information about the page. Unavoidably, some social plugins are then rendered using solely the locally available personalized information, e.g., for the Like button, the total number of "likes" for the page will be missing.

Alternatively, given the very low traffic incurred by SafeButton's non-authenticated queries to the SNS, these can be carried out transparently by SafeButton through an anonymous communication network such as Tor [63]. Given that social plugins are loaded in parallel with the rest of the page's elements, this would minimally affect the browsing experience (compared to browsing solely through Tor). Moreover, as users would first need to consume the page's content before attempting to use a social plugin they might not notice any delay in its rendering. We defer the evaluation of SafeButton utilizing Tor to the future.

**Potential Challenges with Future Social Plugins.** Although SafeButton currently supports all social plugins offered by Facebook, and our approach is extensible so as to handle the plugins of other social networking services, we consider two potential challenges with future plugins [78]. First, future personalization functionality could include social information from a user's second degree friends, i.e., the friends of his friends, or rely on the analysis of data from the entire user population of the social network. Second, this type of personalization could involve proprietary algorithms not available to the client at run-time.

Fundamentally SafeButton supports social plugins that leverage the intersection of user data with website content to enable personalized interaction with the website. We consider user data to be provided by the user to the social network or available to them. Examples of the latter is content uploaded by their friends that the user can access through their social profile. Future plugins that follow this data model will be supported by SafeButton. Even if future plugins rely on processing using proprietary algorithms or benefit from extended data sets they may do so offline while storing an intermediate or final product for real-time use by social plugins. In that case, as long as that product is part of the user's data SafeButton will be able to utilize it to support those plugins. We acknowledge that supporting future social plugins of a different nature than the above model is an open challenge and part of future work around SafeButton and privacy-preserving social plugins in general.

**Profile Management** As users may access the web via more than one devices, it reasonable to assume that they will require a practical way to use SafeButton in all of them. Although installing SafeButton on each browser should be enough, this will result in the synchronization of the locally stored information with the SNS for each instance separately. In our future work we will consider the use of cloud storage for keeping fully-encrypted copies of the local DataStore and X-Cache, and synchronizing them across all the user's browser instances in the same spirit as existing settings and bookmark synchronization features of popular browsers [22, 42].

Keeping a local copy of private information that is normally accessible only through the social networking service might be considered a security risk as it would be made readily available to an attacker that gains unauthorized access to the user's system. At that point though the attacker would already have access to the user's credentials (or could steal them by installing a keylogger on the compromised host) and could easily gather this information from the SNS anyway.

In any case, users could opt-in for keeping the DataStore encrypted, although this would require them to provide a password to SafeButton (similarly to the above mentioned settings synchronization features). For the pure JavaScript implementation though, as discussed in Section 3.5, the cryptographic key can be supplied by the SNS upon user login, making the process completely transparent to the user.

**Security in Multi-user Environments** We now consider the operation of SafeButton in a multi-user environment where more than one users share the same browser instance. In general, sharing the same browser instance is a bad security practice, because after users are done with a browsing session they may leave sensitive information behind, such as stored passwords, cookies, and browsing history. Ideally, users should maintain their own browser instance or accounts in the operating system.

SafeButton retrieves private information when users are logged in the SNS, and stores it locally even after they log out, as it would be inefficient to erase it every single time. Multiple users are supported by monitoring the current cookies for that domain of the SNS, and serving personalized content only for the user that is currently logged in. Local entries that belong to a user ID that does not match the one currently logged in are never returned. Obviously, users that share the same OS account can access each other's locally stored data, since they are contained in the same DataStore instance, unless they have opted in for keeping their data encrypted, as discussed earlier.

**Shortcomings of the Graph API** We have briefly mentioned some obstacles we have encountered, namely shortcomings in the developer API provided by Facebook, in respect to our objective of protecting the user's privacy while maintaining full functionality for the social plugins. We summarize these issues here and discuss how the social networks in general could support us.

*User Activity Updates through the API.* Currently the Facebook API [17] offers access to the social graph but there is no way to receive updates or "diffs" when something changes. For instance, we retrieve a friend's "likes" through the API, we are also able to fetch only new "likes" from a point forward, but are unable to receive notice when that friend "unlikes." A friend "activity" or "history" function could significantly aid our implementation in keeping an accurate local store.

*Accuracy of the Provided Information.* Sometimes, the API calls and documentation offered to developers differ slightly from the actual behavior of a plugin when it is offered by the SNS itself [18]. This creates a predicament for developers wishing to replicate the functionality.

*Support for All Social Information that is Otherwise Accessible.* We consider it reasonable for the API to provide access to information that is accessible via the social plugins offered by the SNS itself or via the profile pages of its users. For instance, there is no API call to access the comments of a specific user, although they appear in the user's profile page. Scrapping could retrieve them, but this practice is not ideal. Therefore, in our case, we have to resort to practices that result in reduced accuracy, such as anonymously retrieving a sample of the comments of a page and placing the comments of a user's friends at the top, if present in the sample. Retrieving the entire set of comments could be inefficient for pages with too many comments.

Alternatively, if Facebook did offer a more elaborate API around comment retrieval we could fetch the user IDs of all the commenters and subsequently specify a set of IDs to retrieve comments for. In that way we could hide the IDs of a user's friends among a group of $k$ strangers and request their comments for that page [101].

# Chapter 4

# Topology-Aware Network Tunnels

## 4.1 Overview

We present an architecture called Topology-aware Network Tunnels (TNT) which reduces insecure network paths to Internet services without their participation. An insecure path is a set of links over the Internet carrying traffic vulnerable to a network adversary. We use unencrypted and unauthenticated protocols such as HTTP or SMTP as our use case. Shorter insecure paths limit the exposure of the vulnerable, e.g., plain-text, traffic to passive and active network adversaries. At a high level, TNT establishes a network overlay of secure tunnels between the client and a set of vantage points. TNT evaluates the network path from each vantage point towards each packet's destination. It then selects the tunnel minimizing exposure to adversaries.

The TNT architecture addresses two key challenges: (1) optimize the placement of secure tunnels across the Internet and (2) determine the optimal tunnel to route each network packet through.

## 4.2 Threat Model

In our threat model the adversary can both passively monitor and actively alter network traffic at some point between a client and a server. This includes end-user ISPs as well as Internet backbone operators. Backbone networks, especially Tier 1 providers, are able to

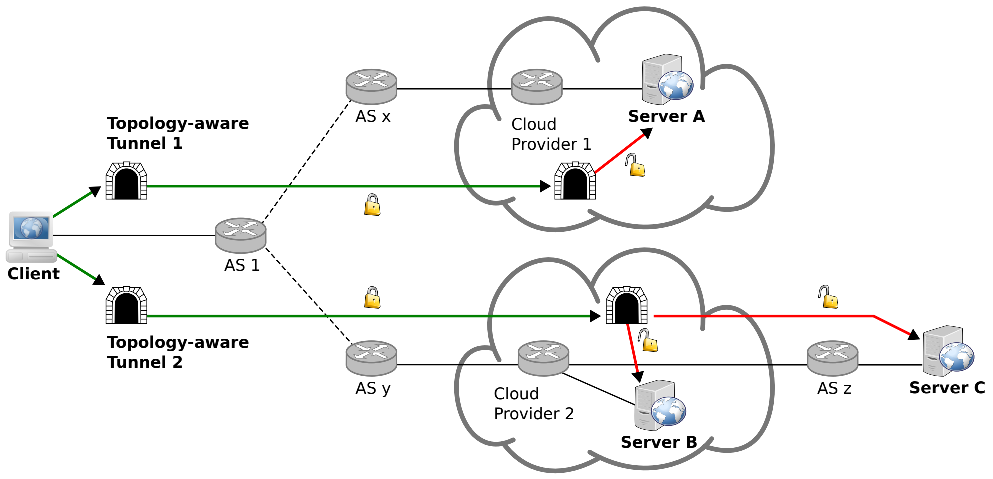

Figure 4.1: In the TNT architecture an overlay of secure topology-aware tunnels is established between the client and a set of network vantage points. The number and placement of secure tunnels is strategically selected to minimize the network distance packets need to travel outside the overlay to reach their destination. Individual network packets are intelligently routed through the tunnel exiting closest to their destination. Tunnel exits within the same network as the destination of a packet (Servers A, B) eliminate the exposure of traffic to network adversaries.

eavesdrop and tamper with traffic from multiple ISPs as it passes through.

Clients on a public network, e.g., WiFi hotspot, run TNT locally on their system. Alternatively, in a trusted private network such as a residential setup TNT can run on the home router. We also consider the networks hosting Internet services as trusted. Hosting providers have a clear incentive to keep their network secure from external threats and honor their agreement with customers. These threats include individuals or organizations passively eavesdropping on or actively manipulating traffic. Cloud networks where customer traffic may travel between data centers are assumed to secure their links. As a matter of fact Google has responded to evidence that intelligence agencies were eavesdropping [46] on its data centers by encrypting [37] the connections between them. Microsoft [43] has done the same. An adversary able to gain access to the trusted networks or systems of the client or the server is out of scope.

TNT creates encrypted tunnels between a client and key networks where Internet ser-

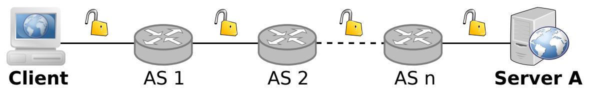

Figure 4.2: Example of a network path on the Internet. For insecure protocols, such as HTTP, data is exposed across the path to operators of the underlying infrastructure.

vices form clusters, namely in the cloud. As a result, adversaries not able to compromise the services or their hosting providers are presented with encrypted and authenticated traffic as opposed to plain text. This includes end-user ISPs and Internet backbone operators which are presently a threat because of their position in the network. It might seem that TNT exits create appealing targets where Internet traffic is funneled through a few networks. However, an adversary able to monitor tunnel exits is already able to monitor the networks hosting the servers and gains no advantage by the presence of TNT.

## 4.3 Design

### 4.3.1 Topology-Aware Network Overlay

The key intuition behind our proposal is that Internet services are clustered in few cloud and other infrastructure-as-a-service (IaaS) providers. Therefore we can optimize the number and placement of secure tunnels by collocating them with these infrastructure providers. This addresses the first challenge from above. That way we can shorten the insecure network path and essentially bring the client as close to these servers as possible, ideally within the same network. As a result, the traffic of insecure protocols will have minimal or zero exposure on the Internet. Apart from minimizing the overall path length, we also define metrics rewarding or penalizing the presence of a trusted or untrusted intermediate network in the path. The trustworthiness of a network is context specific so we focus on path length.

Figure 4.2 presents an example of a network path today. The set of links and routers a client's packets must traverse to reach a server is grouped into autonomous systems (ASes) and controlled by distinct organizations. Note that such path might span different countries or continents. This translates to potential passive and active attacks against the user's web browsing or e-mail.

Figure 4.1 presents the TNT architecture as an overlay on the existing Internet infrastructure. TNT has established secure tunnels between the client and two cloud networks that exhibit high clustering of Internet services. `Server A` is hosted by `Cloud Provider 1` and we can reach it through `Topology-aware Tunnel 1` without exposing plain-text traffic to the Internet. Packets towards `Server A` enter the tunnel before leaving the user's network and are encrypted and signed. Internet routers operated by `AS 1` and `AS x` observe an encrypted flow from the user to `Server A`. Without TNT these ASes have access to plain-text traffic. Packets exit the tunnel inside the trusted network of `Cloud Provider 1` and are authenticated and decrypted. Subsequently packets transit the cloud provider's network and reach `Server A` which is unaware of the process. To reach `Server C` without TNT the user's packets will travel in plain text through `AS 1`, `AS y` and `AS z`. With a TNT link to `Cloud Provider 2` they travel encrypted and signed through `AS 1` and `AS y`. `Server C` is an outlier not hosted in a cluster of Internet services. In this case TNT is able to minimize the length of the insecure network path so instead of 3 ASes only `AS z` will be able to observe plain-text traffic. Next we describe how the TNT router determines the optimal tunnel to route traffic through so as to minimize insecure network paths.

### 4.3.2 The TNT Router

The TNT router is a routing software suite managing topology-aware tunnels and directing traffic through them. It is located on the client's system or local network gateway, for instance a home router, and maintains topology-aware tunnels with remote networks based on the placement strategy described earlier. It has a network-mapping and a decision-making component. Given the available tunnels and a specific destination address the mapping component employs a set of probes to discover the network path between each tunnel's exit on the remote network and the destination. The discovery process involves active and passive network measurements described in section 4.6. The information is passed on to the decision-making component which evaluates it and assigns metrics on each tunnel based on its suitability to carry traffic to the specific destination. Based on the metric the TNT router directs outgoing traffic through the tunnel which minimizes its value. This satisfies the second challenge from the beginning of this section. To account for

```
Kernel IP routing table                                      (1)
Destination  Gateway      Genmask          M IF
0.0.0.0      192.168.0.1  0.0.0.0          0 eth0
192.168.0.0  0.0.0.0      255.255.255.0    0 eth0
```

```
Kernel IP routing table                                      (2)
Destination  Gateway      Genmask          M IF
0.0.0.0      10.0.0.1     0.0.0.0          0 tun0

a.m.z.n      192.168.0.1  255.255.255.255  0 eth0
a.z.u.r      192.168.0.1  255.255.255.255  0 eth0
192.168.0.0  0.0.0.0      255.255.255.0    0 eth0

10.0.0.1     0.0.0.0      255.255.255.255  0 tun0
10.0.1.1     0.0.0.0      255.255.255.255  0 tun1

a.b.c.d      10.0.1.1     255.255.255.255  1 tun1  (5)
```

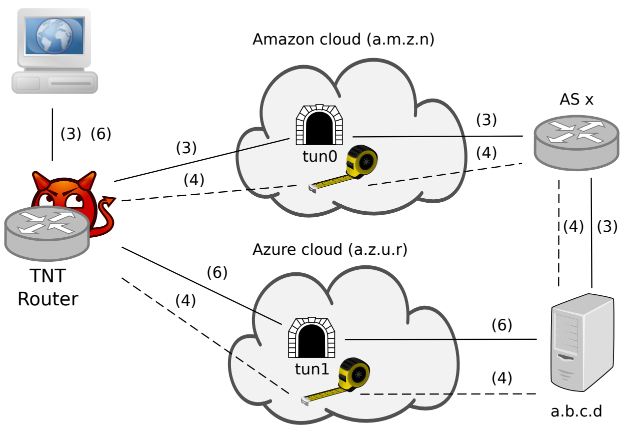

Figure 4.3: Operation of the TNT router when serving client requests towards network destination `a.b.c.d`. Initially it updates the system's routing table (1) to route all network packets through one of the tunnels by default (2). A client's request for which there is no explicit route will go through the default tunnel (3). Subsequently the TNT router will task the probes at each tunnel's exit with determining their distance from that destination so that future requests can be better routed (4). Following the path announcements from the probes, an explicit routing entry is created for that destination (5). The operating system will use that entry for future client requests (6).

the dynamicity of Internet routing the TNT router periodically reevaluates these metrics.

## 4.4 Understanding the Landscape of Web Services

We are motivated by the limited presence of encrypted and authenticated communication protocols on the Internet. We focus on HTTPS and 10,000 popular web services according to Alexa. In 2015 we found that only 30% offer HTTP over TLS. In practice 15% redirect to HTTPS. Just 4% of the sites have an HSTS policy that prevents an active network attacker from downgrading clients to plain HTTP. Repeated measurements the following years show an increase in services capable of HTTPS. With just half of popular web services using HTTPS, under the most optimistic interpretation of the data, our motivation remains relevant. Services available over TLS can also benefit from our proposal. As we are reducing the networks traffic is exposed to, we can mitigate cases of insecure TLS implementations or leaking of the SNI [51] when that is a privacy concern. At the same time we observe that popular web services are collocated in a small set of networks with 10 autonomous systems hosting 66% of the traffic generated by a web browser when visiting the home page of 10,000 popular web servers. We argue that if clients can reach these few networks securely they are able to connect to the hosted web services without exposing their plain-text HTTP traffic to the Internet.

### 4.4.1 HTTPS Adoption

To quantify the extent to which HTTPS has been adopted by Internet sites we evaluated 10,000 popular web domains according to Alexa. We focused on the `.com`, `.org` and `.net` top-level domains that resolved to US ASes. We verified the TLS certificates presented by these domains using the certificate authorities trusted by Mozilla. Table 4.1 presents our findings. Our HTTPS connection attempts were refused by 21.4% of the servers. Even worse, 21.5% redirected our HTTPS requests to HTTP. Additionally, 22.1% of the servers returned a TLS certificate which failed verification. Overall we failed to contact almost 70% over HTTPS.

The few sites supporting both HTTP and HTTPS need to make sure their visitors reach their secure endpoint. Search engine results and links from other sites might steer users towards the insecure HTTP. Also, if users omit the `https://` scheme when typing in the address bar, their browser will default to the insecure `http://`. Unfortunately for the majority of HTTPS-capable sites users will continue to visit them over HTTP. To make matters worse, an active network attacker can prevent the redirection to HTTPS from taking place by replacing `https://` URLs with `http://` in the server's responses in flight. Some ISPs are known to remove the STARTTLS string from SMTP responses, which serves a similar purpose for e-mail. The use of the `Strict-Transport-Security` HTTP header can mitigate this by instructing the user agent to place future requests exclusively over HTTPS. We evaluated the use of HSTS among servers redirecting visitors to HTTPS and found that only 25% return a valid policy. Overall out of 10,000 popular web servers we find that only 3,207 (32.1%) support HTTPS and just 420 (4.2%) support HTTPS with an HSTS policy. Note that just 56 domains are found in the hard-coded HSTS preload list of

| | | | 2015 | 2016 | 2017 |
|---|---|---|---|---|---|
| | HTTPS response | HTTPS? | % | % | % |
| 1 | Error (Conn. refused) | No | 21.4 | 19.8 | 16.0 |
| 2 | Error (Invalid cert.) | No | 22.1 | 19.2 | 10.8 |
| 3 | Error (HTTP 4xx 5xx) | No | 2.9 | 2.0 | 1.4 |
| 4 | HTTPS downgraded | No | 21.5 | 22.1 | 16.7 |
| | Total | No | 67.9 | 63.1 | 44.9 |
| 5 | OK | Yes | 17.0 | 16.6 | 15.1 |
| 6 | OK (HTTP upgraded) | Yes | 10.9 | 15.0 | 29.9 |
| 7 | OK (HSTS) | Yes | 4.2 | 5.3 | 10.1 |
| | Total | Yes | 32.1 | 36.9 | 55.1 |

Table 4.1: HTTPS capability of 10K popular domains. While the percentage of sites offering TLS has increased in recent years, users still access 45% of popular domains without encryption. Moreover, with the majority of HTTPS domains lacking an HSTS policy, users are vulnerable to TLS stripping attacks and may fall back to plain-text HTTP.

Chrome and Firefox.

Our most recent findings are consistent with a similar study [91] which highlights the lack of encryption in the web when moving away from few popular sites such as search engines and social networks. It also predicts that it will take more than 5 years for sites in the long tail of the web to adopt HTTPS in their majority. Moreover, the evaluation [39] of the TLS implementations of popular domains in 2017 found 13% to be inadequate. (Grade F) SSL 3.0 and TLS 1.0, which are considered insecure implementations, were found in 15% and 93% of domains respectively and a non-trivial number of domains were vulnerable to to known attacks such as DROWN, POODLE, CRIME and protocol downgrade.

HTTP/2 [25] is the latest version of the HTTP protocol. While web browsers, such as Chrome and Firefox, currently implement it only over TLS to motivate websites to adopt encryption the RFC does not make that practice mandatory. On the contrary, HTTP/2 over cleartext TCP (h2c) is described as a possible implementation. Even if web vendors refrain from supporting such an insecure option, HTTP/2 will not only have to become

| % | Autonomous System Name |
|---|---|
| 17.1 | Akamai Technologies, Inc. |
| 13.9 | Amazon.com, Inc. |
| 11.4 | CloudFlare, Inc. |
| 9.9 | Google Inc. |
| 3.7 | EdgeCast Networks, Inc. |
| 2.9 | SoftLayer Technologies Inc. |
| 2.1 | Fastly |
| 1.7 | Tinet SpA |
| 1.6 | Internap Network Services Corp. |
| 1.5 | Rackspace Hosting |
| 65.8 | Total |

Table 4.2: Top 10 most frequent ASes hosting sites.

widely adopted but domains will need to deprecate HTTP/1.0 and HTTP/1.1 currently offered over unencrypted connections. Based on the deployment history of TLS, the need to support older clients as well as the resources for updating the configuration of a web service will likely make deprecating insecure HTTP a long process.

### 4.4.2 Web Service Collocation

To study the geography of Internet services we mapped the websites from our data set to their respective ASes. A site may depend on more than one domain for resources such as scripts and images so we used a web browser to fully render the home page of each domain in our data set and recorded the destinations involved. We did not log HTTPS requests. We consider the home page of a domain to be the content received when visiting the exact domain or the standard `www` subdomain.

We visited the home pages of 9,944 domains from out data set. We excluded the 56 HTTPS-capable domains found in the HSTS preload list of Chrome. Ultimately we made 701,929 HTTP requests towards 34,893 unique domains to fully render the home pages. We subsequently resolved the domain names to their respective IP addresses and mapped them to ASes based on BGP prefix announcements collected by APNIC. Table 4.2 presents the top 10 most frequent ASes hosting the web servers involved in our 701,929 HTTP requests. The top 10 most frequent ASes host servers that receive 65.8% of all HTTP requests made.

Web servers hosted by Google present an interesting case. 22% of the HTTP requests made to Google servers target the `google-analytics.com` and 13% the `doubleclick.net` domain. As evidence has shown [47] passive network adversaries colluding with Internet backbone providers collect identifiers involved in requests to these domains to track users. It is also interesting that to reach Google our requests had to travel through two different tier 1 Internet backbone providers. The requests were made from a residential ISP and a university network in the US. In contrast, using our proposal (TNT) we can reach Google in a single network hop without exposing traffic to backbone providers.

To summarize, web services are clustered in few networks owned by cloud and other infrastructure-as-a-service (IaaS) providers. If end-to-end security with these services is not available, the next best thing is for users to establish a secure link to these networks and route traffic through it. Cloud providers make this approach practical as users can deploy their own virtual machines in the same networks.

## 4.5 Implementation

We have implemented the TNT router as an IPv4 routing software suite and tested it in Linux. An Internet-layer implementation is more flexible since it is transport and application-layer agnostic. While the core of the router operates at the IP layer, peripheral components implement high-level logic that enables traffic handling based on transport and application-layer heuristics. By default the router focuses on HTTP (TCP port 80) and SMTP (TCP port 25) traffic while all other traffic, including HTTPS, is routed as if TNT is not in place. The TNT router presents its tunnels as network interfaces to the operating system. The router determines the optimal tunnel to route outgoing traffic through and communicates its decision to the operating system. It does so by interacting with the underlying routing structures. Updating the operating systems routing structures affects how IP packets are transmitted through the available network interfaces. The operating system ultimately writes outgoing packets to the appropriate interface.

The TNT router reacts to outgoing traffic but instead of preventing its transmission until it makes a routing decision it applies its decision to future flows to the same destination. This way it does not disrupt the user's activity or impact network performance. While this means that the first flow to a new destination is not routed optimally in section 4.6 we show that the routing state quickly becomes optimal following the router's initialization phase. The routing state persists across system restarts.

Realizing the TNT router addresses the following challenges: (1) Be practical to deploy, use and maintain. (2) Reliably discover the network topology between tunnel exits and Internet destinations to make routing decisions. (3) Dynamically update the system's routing table without disrupting existing connections. Any naive routing update will reset connection-oriented protocols such as TCP.

### 4.5.1 Deployment

We use OpenVPN to establish TLS-based tunnels with virtual machines in the cloud. Tunnels appear as network interfaces to the operating system with a standard 1500-byte MTU. IP packets entering the tunnel are handled by OpenVPN which fragments them if necessary, encrypts them, appends its signature and sends them to the other end of the tunnel which reverses the steps. OpenVPN supports a variety of ciphersuites from OpenSSL. Our design is not specific to a tunneling technology or ciphersuites.

The deployment of the TNT overlay is automated including installing the TNT router locally and launching the necessary virtual machines in the cloud. We use a combination of Unix shell scripts and the command-line interfaces offered by cloud providers. At the moment we prompt the user for their cloud account credentials however we envisage a deployment process without any user involvement. We do not depend on specific cloud providers but deploying a virtual machine is a provider-specific process which we need to implement. We launch Linux virtual machines in the cloud and configure OpenVPN on both ends of each tunnel. Once the tunnels are established the deployment phase is complete and the TNT router begins running on the user's system without the need for further interaction. Signing up for a new cloud account can be streamlined as part of the deployment script. Users do not share their virtual machines with others but they do share the underlying physical hardware. Attacks from a collocated virtual instance are beyond the scope of our threat model. Cloud operators could offer TNT links to their network as a service so that users do not need their own virtual machines.

### 4.5.2 Operation

The TNT router has three components; (a) a TNT traffic selection program running on the client, (b) the TNT routing daemon also running on the client and (c) probes running on the remote end of each tunnel, which in our case are virtual machines in the cloud. Figure 4.3 depicts the operation of TNT. Initially the TNT daemon brings up the tunnels as distinct network interfaces. One of them at random is set to be the default interface meaning that all traffic TNT is configured to handle goes through it. The system's routing table is updated from step `1` to `2` in the figure. Traffic TNT is not configured to handle gets routed as if TNT is not in place, i.e., still gets routed based on step `1` in the figure. So far all tunnels but the default remain unused by the operating system since it has no reason to prefer them over the default. As a result the client's initial requests to the Internet host `a.b.c.d` will go over the default tunnel (step `3`). Delaying outgoing traffic until the TNT router calculates the metrics for a destination would impact performance. Setting up a default TNT route instead protects traffic from end-user ISPs without the need to wait for a routing decision. In section 4.6 we show that the router quickly makes an optimal decision applied to subsequent flows.

The TNT traffic selection program inspects outgoing traffic for the purpose of identifying destinations that the TNT router must handle. It uses libpcap and is able to identify traffic flows. To select traffic it uses BPF expressions and by default focuses on outgoing TCP flows to port 80 so as to select HTTP traffic. For each new flow matching the selection filter it extracts the destination IP address and queries the TNT routing daemon for an optimal route. If not found it tasks the forward probes with mapping their network path to the destination (step `4`). The forward probes subsequently communicate their findings to the routing daemon directly. To facilitate the network measurements the traffic selection program supplies not only the target IP address but additional context such as the transport protocol and destination port used. The forward probes listen on their end of the tunnel interface for control commands. They use active network measurements to discover the path to a destination on demand and announce it back to the router. We describe the measurement methodology in section 4.6.

The routing daemon interacts with the other components in two ways; it looks up IP addresses in the current routing table on behalf of the traffic selection program and evaluates network path vectors received from the forward probes. The traffic selection program queries the daemon with the IP addresses of destination in outgoing traffic flows. An IP address found in the routing table associated with a metric value means the optimal tunnel to reach that particular destination has been determined in the past. Otherwise the traffic selection program receives a negative response. The forward probes, tasked with discovering their network path to a destination, announce it back to the daemon. When evaluating the network path from a particular tunnel exit the daemon calculates a metric value based on the tunnel's suitability to carry traffic to that destination. We focus on path length as our metric and count the number of ASes involved. The daemon subsequently decides to route traffic through the tunnel which minimizes the metric and updates the operating system's routing structures to direct packets for that destination through a particular tunnel interface rather than the default. In Linux the daemon uses the Netlink[1] interface to access and alter the necessary operating system structures. In figure 4.3 the probes announce paths with distance 2 and 1 respectively for `a.b.c.d` so a decision is made to route the destination through `tun1`. Future requests for that host will go through `tun1`. This is done with an explicit entry in the operating system's routing table (step `5`). Note that the newly introduced route will only be applied to flows matching the context this route was generated. So a route generated because of an outgoing TCP port 80 flow will only be applied to flows to that port. Flows to 443 or some other port to the same destination will not be affected. Applying a route only to specific transport or application-layer flows is discussed later in this section.

The TNT router performs a series of optimizations to its routing table. To avoid stale routing entries it implements a decaying system where entries that have not been used

[1] http://lxr.linux.no/linux+v3.19/net/core/rtnetlink.c

for routing recently are pruned from the routing table. Similarly, for frequently used destinations it schedules lazy reassessments of the optimal path with exponential backoff to stay current with Internet routing changes. TNT routing entries actually describe entire AS prefixes rather than individual hosts. When the forward probes respond to a mapping query for a particular destination host they lookup and return the related BPG prefix the destination's AS is responsible for. This eliminates additional queries for addresses in the same prefix. Finally the router aggregates routes by grouping adjacent route prefixes to form shorter prefixes and reduce the number of entries in the table.

### 4.5.3 Transparent Routing Updates

Updating the routing table of a live end-user system to essentially implement multihoming is not a trivial task. Network routers dynamically change their routing table on a frequent basis without the same challenge because they simply forward IP packets without altering their header. However, packets exiting an interface in an end-user system adopt[2] that interface's IP address as their source. A routing update directing packets of an existing TCP connection through a different interface will change their source IP address. Packets with the new source IP address will be dropped or met with packets with the RST flag set since from the remote endpoint's perspective do not match any existing connection. Ultimately the TCP connection will close unexpectedly.

To ensure non-disruptive updates to the operating system's routing table we implement a transitioning process which guarantees the continuity of existing sessions in TCP as well as UDP and ICMP logical sessions. We utilize the support for multiple routing tables in the Linux kernel as well as the functionality offered by its Netfilter framework. The key idea is to split a routing update into two phases. Initially a new route is taken into consideration only for new connections while existing ones are routed as if the update never took place. This guarantees continuity. Eventually connections predating the update will terminate naturally and the system will reach a stable state where all connections use the updated

[2] While there are ways around that, upstream providers usually implement egress filtering to block outgoing packets with spoofed IP addresses. In TNT different tunnel interfaces are expected to exit in disjoint network prefixes.

| t | TNT Event | Routing State | Effective Routing Table by Connection Status[1] Existing[2] | New |
|---|---|---|---|---|
| 0 | | Stable | Main | Main |
| 1 | Start | Converging | Main | *TNT0* |
| 2 | | Stable | *TNT0* | TNT0 |
| 3 | Update route X | Converging | TNT0 | *TNT1* |
| 4 | | Stable | *TNT1* | TNT1 |
| 5 | Terminate | Converging | TNT1 | *Main* |
| 6 | | Stable | *Main* | Main |

[1] *Connection status is independent of the transport protocol used. It is logical, applies to TCP as well as UDP and ICMP and is based on timers and bi-directional IP packet exchange.*
[2] *Existing connections are in a logically-assured state. For TCP this is either the Related or Established state.*

Table 4.3: Updating the operating system's routing table so as not to disrupt existing TCP connections. The TNT router transitions the system from the current table to an updated version by cloning the table, modifying the new table and setting it as effective only for new connections.

route and the old one is deleted.

To implement this two-phase routing we clone the currently effective routing table into a new routing table and instruct the operating system to look up new connections in the new table while keep reusing the old one for existing connections. Initially we clone the default, main routing table into a new table TNT0. (Time `t0` in Table 4.3) The system transitions into a state where any already established TCP connections, as well as logically-assured UDP and ICMP connections, keep using the main routing table whereas the destinations of new connections are looked up in the TNT0 table. (`t1`) Eventually all connections predating the update (`t1`) will naturally terminate and the system will reach a state where all current and future connections will use table TNT0 exclusively. (`t2`) Subsequently any updates after time `t2` will clone TNT0 into TNT1, enter a converging state `t3` and eventually reach a stable state `t4`.

If we need to update the effective routing table while the system is still converging from a previous update we must allocate an additional table instead of recycling an existing one. For example a new update during time `t3` will cause the effective table TNT1 to be copied to a new table TNT2 which will then be updated and marked as the effective table. We cannot reuse table TNT0 at this time since it is still being used by connections predating the last routing update. We try to carry out updates in batches to avoid the need for more than two tables at a time. However traffic scenarios such as web browsing might cause bursts of updates that do not fit in a single batch. Under reasonable conditions Linux does not limit[3] us in the number of tables we can maintain. As soon as all connections associated with an old routing table are terminated that table becomes eligible for reuse in a future routing update. In section 4.6 we quantify the amount of routing tables necessary under realistic network activity. A routing cache would eliminate the need for the above technique. Since version 3.6 [84] the Linux kernel no longer supports such a cache for efficiency reasons. Windows 7 implements a routing cache but the same reasons might justify its removal from future versions.

To advise the operating system which routing table to use for each destination lookup we use kernel routing policies. Each routing table is associated with a mark and we use Netfilter's connection tracking to label new connections with the mark corresponding to the new table. Policies match the mark individual packets carry to specific routing tables.

### 4.5.4 Application-Specific Routing

As mentioned earlier both the operating system's core routing functions and the core of the TNT router make IP-based routing decisions. At the same time it makes sense to configure traffic routing preferences based on high-level context coming from the transport and application layer. For instance by default TNT must only handle IP packets belonging to TCP port 80 flows (HTTP) while HTTPS and any other traffic must not be affected. By default Linux uses a global routing table which affects all packets and is not suitable to our needs. To achieve the necessary flexibility we use routing policies which are combined with multiple

[3] Since version 2.6.19 the Linux kernel supports up to $2^{32}$ routing tables and efficiently addresses them using a hash table. Previous versions supported up to 255 routing tables.

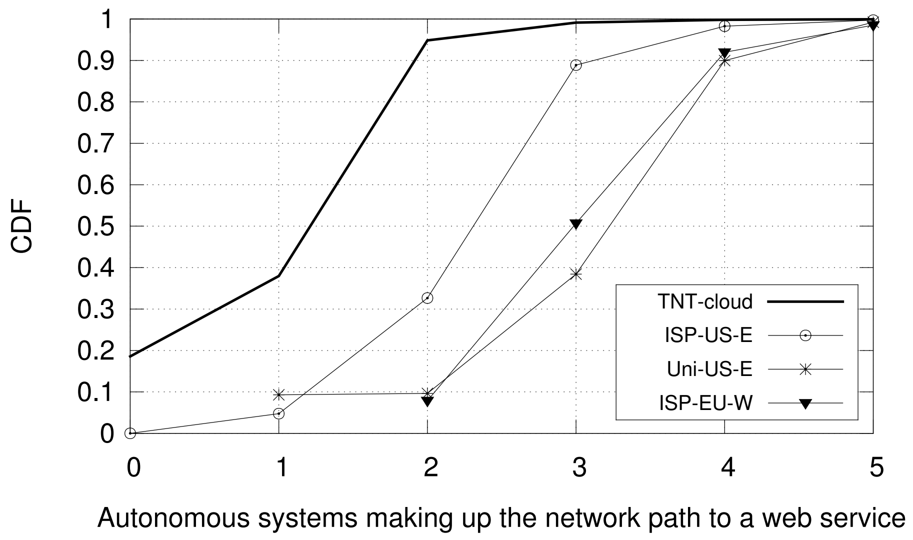

Figure 4.4: CDF of the number of ASes on the network path to each web service. TNT outperforms ISPs by exposing zero traffic to the Internet for 18.5% as well as achieving one-hop paths for an additional 19.5%.

routing tables and the Netfilter framework. Using the latter we mark specific connections or packets based on heuristics such as destination port. Marked packets subsequently are matched to specific routing policies leading to corresponding routing tables. For example we have a IP routing table that is only used for TCP flows to port 80. The system's default table is never modified and, unless we explicitly mark outgoing packets, traffic is routed as if TNT is not present.

## 4.6 Evaluation

### 4.6.1 Network Proximity

We quantify the exposure of plain-text traffic to adversaries by mapping the network paths to popular websites using a series of Internet vantage points. We then compare the results to a TNT deployment in the AWS and Azure cloud networks to evaluate the ability of TNT to minimize traffic exposure. For our measurements we used a total of 7 vantage points spread across the US and western Europe; 4 virtual machines in the Amazon Web Services (AWS)

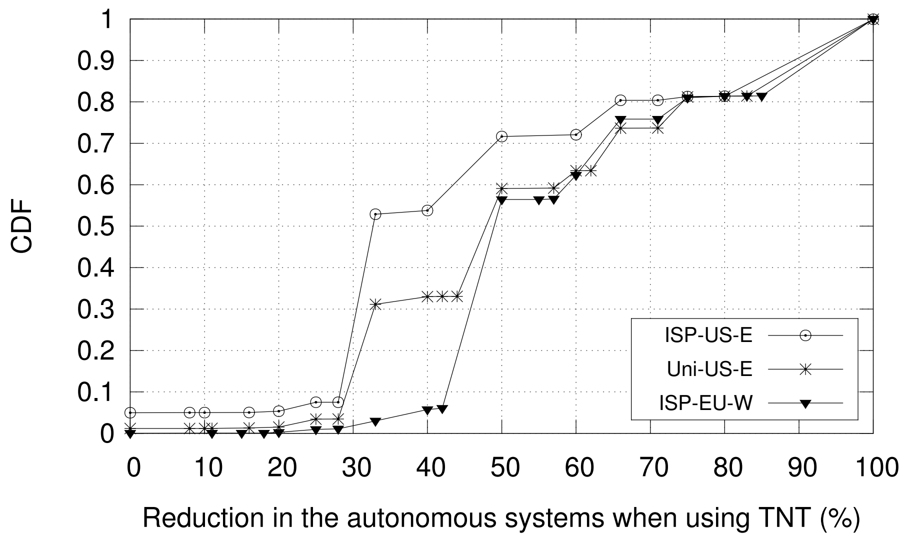

Figure 4.5: CDF of the reduction in ASes on a network path when using TNT as opposed to an ISP. TNT offers at least 33% in 70% of the cases.

and Microsoft Azure (Azure) cloud, 2 end-user lines in ISPs and access to a fast academic network. Our set of hosts was compiled by visiting the home page of 9,944 popular web domains according to Alexa with phantomJS, a Webkit-based, Javascript-capable headless web browser, and collecting HTTP requests. Our final list, including the initial domains, contains 34,893 unique domains resolved to 20,026 distinct IP addresses.

Our network mapping process correlates active network measurements with BGP routing views [72]. This is the same process followed by the forward probes of the TNT router to discover network paths. To measure the actual flow of packets between one of our vantage points and each web server in our data set we sent ICMP type 8 as well as TCP+SYN port 80 packets to the destination host and elicited ICMP type 11 responses packets from all intermediate network devices using varying TTL values in the header. Some network policies drop ICMP packets and firewalls at the destination might drop all packets but the ones the service is expecting. By using port 80 for web services we guarantee minimal disturbance for our measurements. In section 4.5 we discussed how forward probes of the TNT routing receive the same information so as to conduct their measurements with packets that are guaranteed to reach their destination undisturbed. Nevertheless some network policies silently drop TTL-expired packets so we only considered complete network paths for which we had identified all their hops ending up with data for 15,020 of the original 20,026 hosts. We subsequently resolved the IP addresses of each hop in the network paths to their respective AS using BGP prefix announcements collected by APNIC. Finally we verified the accuracy of our measurements by correlating the derived AS paths with BGP views [4] from looking glass platforms. Our methodology guarantees an accurate picture in the case of transit, i.e., customer-provider, relationships between ASes while leaving a margin of error in the case of peering agreements where BGP announcements are not available outside the participating ASes. In production the TNT router will ignore AS paths produced by active measurements that cannot be verified through BGP announcements.

Figure 4.4 presents the CDF of proximity, in terms of ASes involved, of each vantage point to the web hosts in our data set. We define this as our exposure metric, indicating the number of potential network adversaries, which the TNT architecture aims to eliminate or minimize. A distance of zero ASes in the figure translates to the server being in the same AS as our TNT link. Similarly a distance of one AS indicates a direct, peering relationship between our trusted AS and the AS of the server. One may observe that the TNT architecture outperforms end-user ISP and university networks by routing packets to 18.5% of destinations through ideal, adversary-free, paths. Note that TNT also outperforms the individual cloud networks we used. Zero hop network paths in the case of ISPs are attributed to CDNs hosted in their networks and occurs in less than 0% and 5% of the cases respectively. Since end-user ISPs are part of our threat model we do not consider these paths adversary-free. We have also calculated the average proximity of TNT to the home page of each domain as a whole, including all subresources. The results are consistent with figure 4.4. Similar results describe the proximity of TNT to advertisement networks. Additionally, TNT offers consistently shorter paths to almost all destinations tested. Figure 4.5 shows that we achieve at least 33% and 50% shorter paths in 70% and 40% of the cases.

From a privacy perspective one might be skeptical about funneling their traffic to a few large cloud providers. Note that this traffic is already transiting the public Internet in plain text. As TNT scales, traffic is distributed closer to its destination and user privacy

[4]`http://as-rank.caida.org`, `http://lg.he.net/`

improves. Nevertheless one could use TNT to only access destinations that are hosted in the clouds it maintains tunnels with, which is almost 20%.

### 4.6.2 Operating a TNT router

The TNT router is our implementation of topology-aware tunneling for clients to use. To quantify its impact on the end user's system or other network gateway we carried out a web browsing session that generated realistic network traffic patterns for the router to handle. We instrumented Firefox to visit in succession the home pages of 1,000 popular web domains according to Alexa. Firefox waited for each page to fully load before moving on to the next and we cleared its cache between sessions. HTTPS traffic was unaffected by TNT and was routed through the default interface. Plain-text HTTP was dominant as shown in section 4.4.

Our evaluation focuses on the impact the router has on the system's resources and is expressed in hits in the routing table, the number of concurrently active routing tables as described in section 4.5 and the number of entries found in the effective routing table over time. The first measure determines the amount of active network measurements necessary. The second and third measures determine the stress on the system's CPU and memory.

Initially we measured how quickly the operating system's routing table converged to its optimal configuration so that each IP packet is routed through the tunnel that exits closest to its destination. An IP packet with a destination address for which we do not yet have an explicit route is classified as a lookup miss and results in forward probes mapping and assessing the path to that destination. On the other hand, a destination address for which TNT knows the best way to reach it has an entry in the routing table and constitutes a lookup hit. Figure 4.6 presents the hit ratio over time. One may observe that approximately for the first 500 connections TNT has enough optimal routes to satisfy 50% of the destinations involved. This indicates a fast bootstrap phase. Over time the hit ratio increases and by the end of the browsing session we see that TNT was able to satisfy almost 80% of the destination lookups. For subsequent browsing sessions the hit ratio remains between 100% and 98%. The slight drop is attributed to sites with dynamic content.

The TNT router adds a network-specific route, associated with a metric, for every

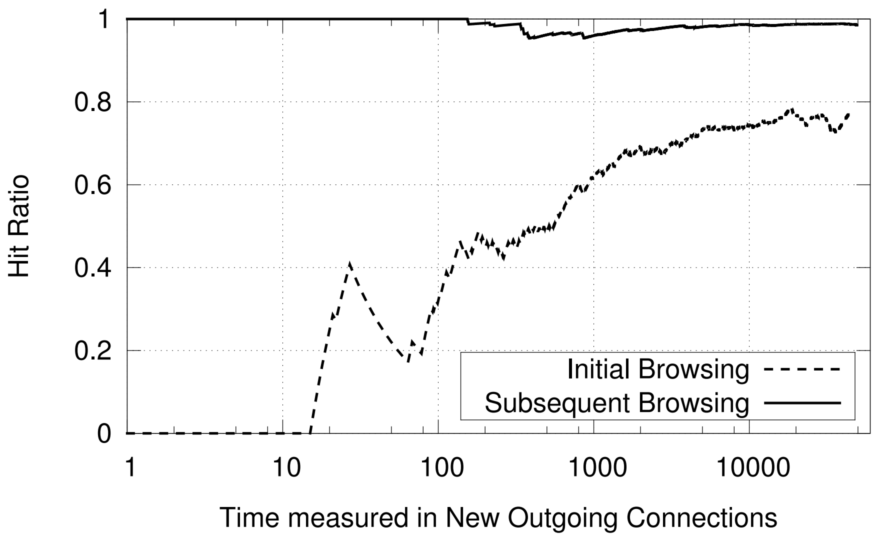

Figure 4.6: Ratio of optimal versus suboptimal TNT routing over time. Initially the ratio is low causing forward probes to map network paths. Later on it quickly rises indicating that few popular destinations have been mapped.

destination the distance to which has been determined by the forward probes. As the user visits more and more unique Internet destinations the routing table grows. Web browsing is a representative example of this scenario as it involves a plethora of different servers. Figure 4.7 presents the number of routing entries in the effective routing table over time. Note that the effective routing table is the one the operating system will use to look up the destinations of new connections. During the initial browsing session the size of the routing tables reaches approximately 1,400 entries. That might seem daunting compared to the initial 2 entries for most systems with a single network interface. However the implementation of the routing table (`fib_table`) in Linux is highly efficient[5]. It uses hash tables to lookup destinations in near constant time. The only way the number of entries impacts the system is in terms of memory consumption. However the way route information is stored in data structures is also efficient as it groups common parameters between routes to a single data structure (`fib_info`) that is shared by all routes. In practice the memory overhead for the number of routes TNT introduces is negligible even for embedded systems such as home routers.

[5] http://lxr.linux.no/linux+v3.19/include/net/ip_fib.h

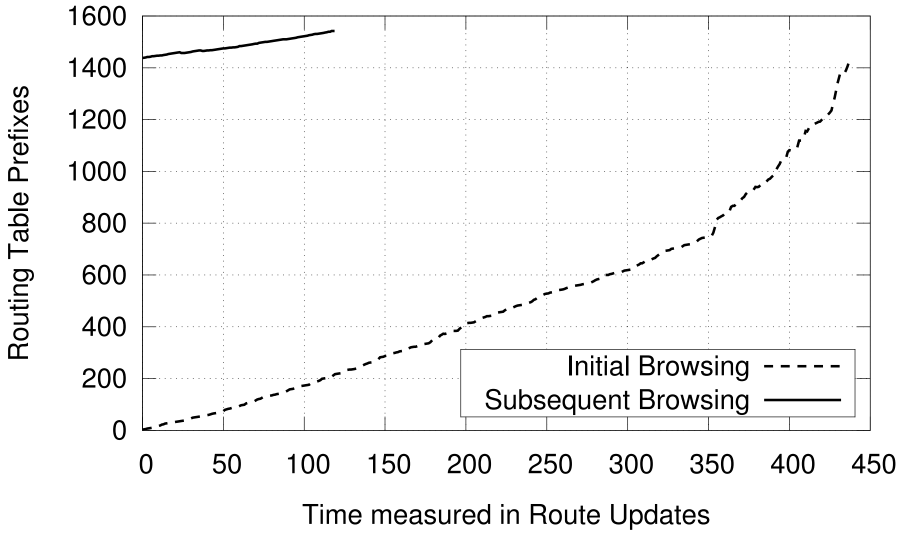

Figure 4.7: Number of entries in the system's effective routing table. The TNT router creates explicit entries per AS as part of its operation. In practice memory consumption is negligible and processing time near constant.

Note that we periodically expire routes that have not been recently involved in lookups.

In order to ensure a smooth transition when updating the routing table the TNT router uses auxiliary tables as described in section 4.5. Visiting a page causes multiple connections to be created in an asynchronous bursty manner which may result in an equally bursty set of routing table updates. Figure 4.8 shows how the system converges from multiple routing tables to a single one. Multiple tables are used only during routing updates which are infrequent. We argue that web browsing models the worst case scenario in terms of traffic patterns and so this figure sets an empirical upper bound on multiple routing policies and tables.

### 4.6.3 Web Browsing over TNT

To quantify the effect TNT has on the web browsing experience we studied the round-trip time (RTT) of packets towards the respective servers along with the overall load time for each page. Note that we measured RTT from the client's perspective. Her packets had to traverse a TNT link, reach the cloud network and then proceed to their final destination.

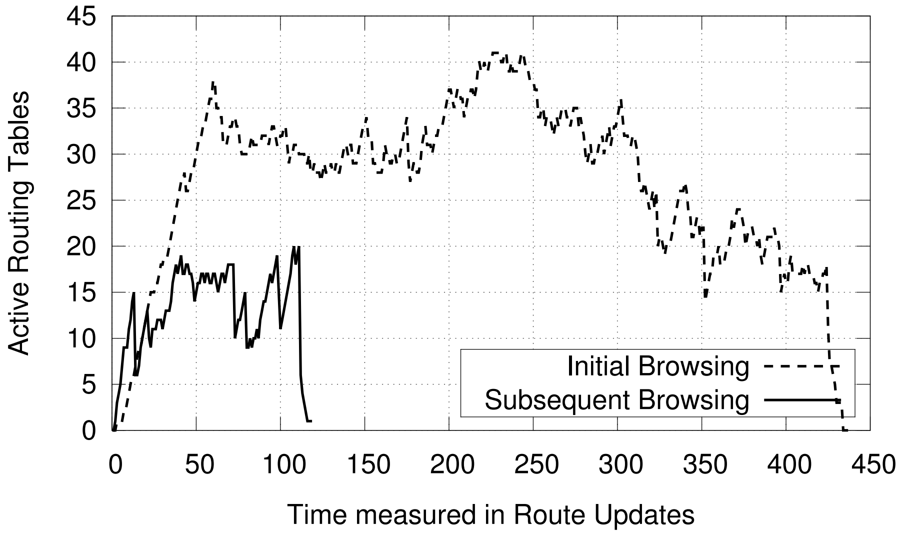

Figure 4.8: Number of routing tables concurrently active. Following an update, existing connections keep using the previous version of the table to avoid disruptions. In practice memory and processing overhead are negligible.

Our baseline was an academic network with a fast Internet connection in the east coast of the US. In terms of network latency, as figure 4.9 shows, TNT offers comparable times to our baseline. We sent TCP packets to destination port 80. In terms of page load time figure 4.10 shows consistent results between TNT and the baseline.

## 4.7 Security Discussion

For destinations hosted in the cloud a passive Internet adversary sees an end-to-end encrypted connection. Examples are servers A and B in figure 4.1. Such destinations are the primary use case for TNT so clients can reach them without exposing plain-text traffic to the Internet. Optionally, TNT may also leverage the cloud as a gateway to reach arbitrary Internet destinations over shorter unencrypted paths. Such example is server C in figure 4.1 where only the path between the cloud and the server is unencrypted. We have shown that in such cases TNT always creates shorter paths. There is however a tradeoff between reducing the number of ASes observing plain-text traffic and routing it through networks that may not have originally observed it. Especially ASes adjacent to the cloud may seem

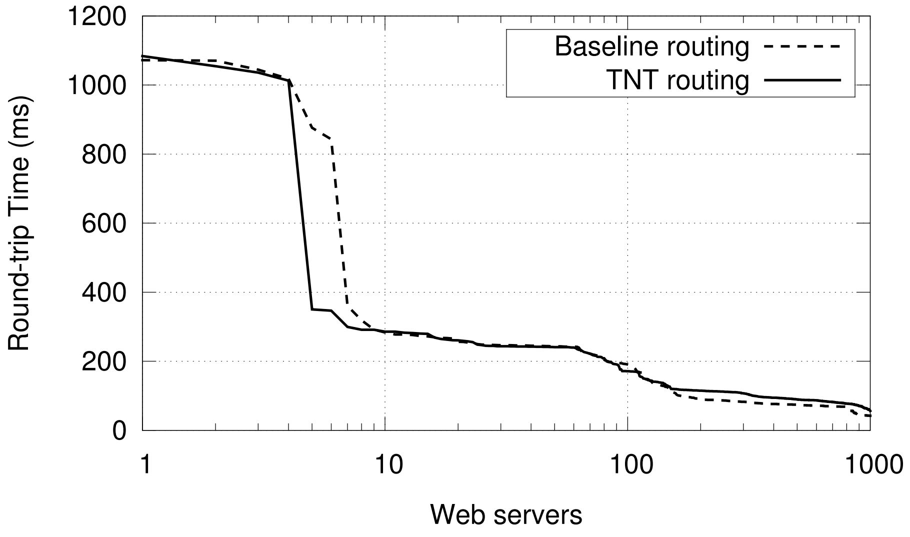

Figure 4.9: RTT of packets routed either through a fast academic network or a TNT link to AWS.

at an advantageous position to monitor the browsing behavior of TNT users. However we do not observe any notable deviation in the shape of the frequency distribution of ASes involved when TNT is present. Without TNT the most frequent AS is found in 22.5% of the paths and with TNT the most frequent AS is found in 19% of the paths. These are two different ASes and naturally, because of our routing decisions, some ASes see more and others less traffic. However, as far as users are concerned there is no single AS that can observe more of their browsing history than without TNT. This can be explained by cloud providers having multiple upstream providers for redundancy and load balancing reasons. In fact, the diversity of ASes involved actually increases.

When the cloud is used as a gateway by TNT it appears to be the source of clients' traffic at the IP level. This facilitates TNT routing. We do not attempt to hide or anonymize the source or destination of traffic. ASes observing encrypted client traffic entering the cloud and unencrypted traffic exiting the cloud can attribute cloud-originating traffic back to a particular client. An adversary can use the timing and size of packets to match encrypted flows between clients and the cloud to plain-text traffic between the cloud and external sites. Cover traffic and shaping techniques may obfuscate such heuristics. However, we argue that

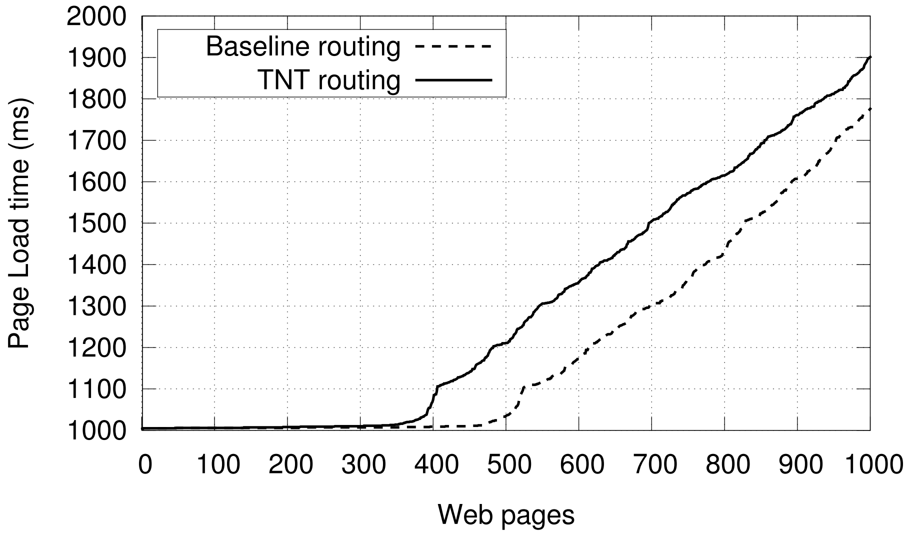

Figure 4.10: Load time of sessions routed either through a fast academic network or a TNT link to AWS.

the actual content of plain-text traffic carries a plethora of information that can identify users. For instance HTTP cookies, referrers and search terms are much more reliable in tracking users than the IP address of the device originating the traffic.

One might observe that the measurements in Section 4.4 identify content delivery networks (CDNs) as popular locations for hosting web services. While they may not be the true origin of web services, from the client's perspective these are the networks it needs to contact to fetch content. By reducing the exposure of traffic to those networks we are able to benefit the security of individual clients that may be targeted or profiled by a network adversary. It is possible that an adversary targets traffic between the CDN and its original server. We expect such traffic to be user-agnostic as the nature of CDNs describes optimizing the delivery of the same content to a large audience. Thus monitoring such traffic is unlikely to enable profiling an individual client and altering that traffic is more likely to be noticed because of the delivery scale.

An active adversary could try to either block our ability to map network paths or falsify the data we receive. It could also try to block TNT links. To make TNT network measurements resistant to blocking we tailor our probes to the packets a specific service is expecting to receive. For HTTP we transmit IP packets with a TCP header indicating destination port 80. An adversary blocking such packets would also stop actual user traffic towards a service. Fingerprinting traffic generated by TNT measurements is possible though. Instead of blocking our network measurements an adversary could tamper with the data we receive by spoofing responses from upstream routers. However in section 4.6 we describe how we correlate network paths resulting from data plane measurements with AS paths from BGP announcements. A measured path that is infeasible is not taken into consideration by TNT. Attacks against BGP are beyond the scope of this work and any solution is orthogonal. An active adversary situated between the client and the cloud could try to prevent TNT links from being established. Our threat model does not include censorship and failure to run TNT in a network should warn users about the operator's intentions.

Finally it might seem that TNT centralizes traffic flows within a few cloud networks which become appealing targets. However our threat model focuses on adversaries that are not powerful enough to attack the cloud but can carry out passive and active attacks today because of their location on the Internet. This includes ISPs and other infrastructure operators. Moreover the key idea behind TNT is to utilize cloud networks to reach destinations already hosted within them. Therefore adversaries powerful enough to attack the cloud gain no advantage from the presence of TNT. As an additional, entirely optional, use for TNT we propose routing traffic to Internet destinations outside the cloud so as to minimize the network path to them. While this makes such traffic available to adversaries capable of compromising cloud networks we argue that the benefit of shielding plain text traffic from every other adversary on the Internet presents an appealing tradeoff. At the same time the TNT architecture benefits from and encourages scaling to more cloud networks. We thus expect that individual clouds will see a decrease in the traffic going through.

## 4.8 Limitations

TNT maps network paths using IP-based measurements. It cannot identify hops operating below OSI layer 3 such as in the case of MPLS tunnels. As a result it will misrepresent the length of network paths featuring such traffic encapsulation. This is not a limitation of the TNT architecture but a constraint imposed by our implementation of network measurements. Gueye et al. [73] can approximate the geographical location of an IP host in the presence of loaded links. They use a set of known landmarks to compare the perceived end-to-end network delay to expected propagation time of the underlying physical links. Using the existing TNT distributed architecture we could approximate the location of network hops on the Internet and highlight hops that appear adjacent at layer 3 but are separated by a great physical distance such as in the case of an overseas or cross-country link.

TNT relies on active measurements which it correlates with BGP announcements passively collected from looking glass platforms. While BGP policies are complex and based on confidential agreements between networks we do not depend on the need to interpret them. Instead, our active measurements, such as traceroute, provide the actual path of packets based on those policies which we treat as a black box. By design the measurements approximate the actual client traffic that will follow so for instance we use TCP packets sent to destination port 80. We use BGP announcements to validate our measurements and complement them in the case of hops which are incompatible with our methodology, e.g., routers which do not return ICMP type 11 responses when the TTL of an IP packet expires on them. We also view BGP hijacking as an orthogonal problem. Hijacking routes combined with interfering with active measurements to hide network hops is a threat we consider out of scope.

TNT gives clients the network perspective of the tunnels they are using. This might affect how they experience systems that depend on the client's network location. For example, DNS replies for load balancing or geo-location purposes will be based off the tunnel's IP address instead of the client's. Because of the inherently inaccurate nature of IP address-based geo-location systems there are existing mitigations to improve the user experience. For example, search engines allow users to set their preferred language if they are misidentified as coming from a different country. The nature of the cloud itself is to expand with additional data centers close to services and their clients. Given this, we expect a gradual reduction of split network views between the client's original network and the vantage point offered by TNT.

## 4.9 Production

Modern clients are already taking advantage of the infrastructure-as-a-service platforms and the cloud specifically to expand their capabilities in terms of processing power, storage and network access. For example, users are able to sync data between devices, exceed their device's storage capacity by offloading excessive data to networks disks and render content using CPU as well as network resources available on remote machines [3]. We envision that TNT is the next step in accessing network services securely. Clients that are already using the cloud [23] to reduce their data usage when surfing the web can adopt TNT in a similar manner.

TNT will be transparent to end users who will not have to maintain it or be aware of its presence. Software updates can deliver TNT to their devices and home routers. Using any kind of resource of course carries a financial cost. How that financial cost is absorbed is an open question for many of the existing uses of the cloud. For example, most of Google services are not directly billed to end users. The cost of services deploying and maintaining TLS and other security features is handled by the Internet services themselves as the community is trying to promote awareness of good security practices and implicitly rewards services which apply them. In other words, the cost of TLS is part of doing business. There is the expectation that users will not make purchases from sites that do not offer TLS. Perhaps the next step is avoiding sites with poor TLS implementations or sites that the security community has identified as high risk in terms of traffic exposure. Cloud networks could also deploy TNT themselves to protect services that choose their networks for hosting instead of their competition. Such practices are not without precedent. [6]

Overall we think TNT is a tool which acknowledges and leverages the nature of modern networks. Given its security benefits we are excited to see in what shape or form the industry will deploy it in practice.

# Chapter 5

# Conclusion

## 5.1 Summary

In this dissertation we investigated the hypothesis that user privacy on the web can be improved by reducing the third parties involved. Towards this goal we presented a design for privacy-preserving social plugins. Our design effectively removes third parties that are embedded on websites by decoupling the loading of their content from the user's visit to a site. At the same time we are preserving their original functionality which in turn does not affect usability and acts as an incentive for the adoption of our proposal. Additionally we presented a network architecture which effectively reduces the networks involved when routing traffic to a website. By routing otherwise plain-text traffic over our encrypted overlay we are reducing the intermediate nodes clients and servers have to trust when exchanging traffic.

We also investigated the hypothesis that such privacy benefits could be accomplished unilaterally through client-side initiatives and without affecting the operation of individual websites. We demonstrated that browsers have the technical capability to offer locally the personalized content and functionality of social plugins and implemented our proposal without the need for cooperation from social networks. We also demonstrated that infrastructure-as-a-service has expanded the network access of clients. Specifically it has made it possible for them to deploy a secure overlay that allows them to reach a web server without the need to trust intermediate networks with their traffic.

## 5.2 Future Directions

As an increasing part of daily life is moving to the Internet, preserving user privacy becomes an issue of utmost importance. From third parties accumulating user data in exchange for personalized web content to more and more everyday devices being connected to the Internet, there are significant challenges to ensuring transparency and control for users. To this end we introduced novel designs for privacy-preserving content personalization in online social networking platforms as well as reducing third parties privy to the user's network activity. However there are interesting challenges ahead. The following research directions capture our vision towards user privacy in respect to the evolution of the Internet. We argue that user data should effectively reside on the client and that the client should interact only with the user's intended destination when explicitly sharing this data. Given the evidence presented in this thesis such requirements are aligned with the nature, function and evolution of the web.

### 5.2.1 Decentralizing User Data

There are open issues towards addressing the privacy concerns behind social networking services and other ubiquitous third parties around the web storing user data. The former receive user data in exchange for some functionality users enjoy while the latter amasses data about the users based on their web browsing activity. Existing research has focused on blocking the interaction between users and those parties, an approach appealing only to those users willing to trade functionality for privacy. Work that tries to prevent third parties from holding on to user data, based on whether such data retention is beneficial or expected from the user's perspective, ends up participating in an arms race with those services. For example Google and Facebook are both first-party services users visit and third-party trackers profiling the users' web browsing, thus making the separation non-trivial. At the same time third-party trackers are actively trying to detect and evade such black-listing technologies, e.g., third-party cookie blocking, ad-blocking software and private browsing windows.

We argue that future research should focus towards ways to empower users by giving them control over their data while defining transparent ways for data sharing. The existing monolithic system of web services interweaves the functionality they offer, e.g., an online social graph, with the user-generated content overlaid on top of it. In this thesis we have demonstrated a practical design principle for online social networks that addresses the simultaneous first and third-party nature of Facebook and prevents user tracking from widgets embedded in websites the user visits while preserving the functionality of those widgets including their personalized content. A key aspect of our design is avoiding direct interaction with Facebook when user-owned data can be placed on their devices and used to render personalized content locally. We have proven that user agents are capable of implementing such design and we argue that the next step is to focus on making clients the sole carriers of user-generated content. Towards this direction we should explore moving all user data away from such monolithic services in favor of peer-to-peer user interactions. A limitation of our current work is that if users choose to interact with social widgets, e.g., "Like" a page, that information is written back to Facebook and thus reveals their visit to a particular site. The fundamental principle of making user devices the only place their data lives gives users better control by default and promotes transparency as Facebook will have to make explicit data requests towards clients and justify them.

While one could argue that clients are unequipped to handle their data we have demonstrated the practicality of such approach by implementing our design. The next step is to leverage the cloud as a private syncing medium across devices, a privacy-preserving processing environment and a privacy-friendly facilitator of user-to-user data sharing. We argue that the growth of infrastructure-as-a-service technologies has extended the concept of a client beyond the physical confines of a device. Future research should revisit the privacy of user data given the evolution in the nature of clients that involves multiple user-owned devices, private off-device storage, processing and networking capabilities in the cloud.

### 5.2.2 Revisiting the Security Model of Client-Server Interactions

Given that TNT redefines a client's proximity to services in the cloud, it can benefit current work in Internet measurements and security as well as drive new research. At the same time Sherry et al. [98] describe how the cloud can offer middleboxes as a service. The cloud becomes the point of ingress and egress for enterprise networks and implements proxies, firewalls and load balancers. Going forward this makes the cloud the rendezvous point for clients interacting with servers over the Internet. As a result, new research can focus on intra-cloud traffic under a more simplified threat model and improved network conditions.

For example routing attacks [100] in the Internet's backbone are less likely inside a cloud network which is controlled by a single operator. TNT transforms reliable routing into two orthogonal problems; reliably routing to the cloud and reliably routing within the cloud. Factoring in the implicit trust placed by users to the cloud networks hosting the services they are trying to access creates an interesting research space that can yield practical solutions.

As the industry, represented by AT&T and Google, is trying to standardize middleboxes [11, 12], being able to reliably detect their presence is an interesting and crucial area of research. TNT assumes cloud networks are trusted and therefore does not consider attacks against plain text traffic between a TNT endpoint and its destination while in those networks. In practice, cloud providers may themselves employ middleboxes internally that process, cache or alter traffic. For example a legitimate IDS could cache the plain text user data it processes. We argue that the presence of such IDS should be transparent to users. While existing research [62,66,87,104] has identified middleboxes based on implementations that make their traffic distinguishable from actual clients, a motivated adversary can implement a middlebox that mimics the network stack of end-user systems. We argue that middlebox detection has to rely on fundamental aspects of their nature. Until now research towards this direction has been hindered by the volatile geography of the Internet's backbone. However, as the attack surface has shifted from the backbone to the cloud, existing security research can be revisited and new research may be enabled. For example, accurately determining the geographic location of a server can highlight an adversarial network tampering with traceroute. Because constraint-based geolocation [73, 89] uses delay measurements to estimate the location of Internet hosts it does not perform well when the targets are not close to the landmarks. Topology-based geolocation [77] leverages the network topology to improve on delay-based measurements but it introduces the additional challenge of accurately mapping the data plane. Volatile network latency and packet loss rates could become more reliable [74] in the smaller core of a cloud network. In other words, TNT narrows the search area from the backbone to cloud networks and IP geolocation appears to benefit from the nature of the cloud.

Going forward, detecting middleboxes in low-latency cloud networks may be possible by identifying the delay added by those middleboxes as a fundamental result of their processing. In contrast to a typical router, such devices use a high-level function to process network packets prior to forwarding them to their intended destination. Such processing, usually deep packet inspection, increases the round-trip time between the client and the server. Intelligently created packets the processing of which exacerbates the delay introduced by the middlebox but does not affect the operation of the legitimate the server can highlight the presence of a middlebox. If the client observes timing discrepancies between regular packets and tripwires it can conclude that a middlebox is interfering with the traffic. For instance, sslstrip [41] rewrites a server's HTML response to replace all `https://` links with the insecure `http://` scheme so it can monitor the client's future traffic. This may result in passwords and other sensitive information being transmitted in plain text. An sslstrip tripwire can be a pair of pages, one with a large number of `https://` links to be stripped and one without. Tripwires have to generate timing discrepancies above the natural network noise. Therefore they are most effective in low-latency networks involving few hops. Rethinking client-server interactions over the Internet in the context of the local network inside the cloud is an interesting direction.

# Bibliography

[1] A Certificate Authority to Encrypt the Entire Web. `https://www.eff.org/deeplinks/2014/11/certificate-authority-encrypt-entire-web`.

[2] AdBlock Plus. `http://adblockplus.org/`.

[3] Amazon Silk - Split Browser Architecture. `http://docs.aws.amazon.com/silk/latest/developerguide/split-arch.html`.

[4] AT&T charges $29 more for gigabit fiber that doesn't watch your web browsing. `http://arstechnica.com/business/2015/02/att-charges-29-more-for-gigabit-fiber-that-doesnt-watch-your-web-browsing/`.

[5] CERT: Multiple DNS implementations vulnerable to cache poisoning. `http://www.kb.cert.org/vuls/id/800113`.

[6] Cloudflare - Introducing Universal SSL. `https://blog.cloudflare.com/introducing-universal-ssl/`.

[7] Do Not Track - Universal Web Tracking Opt Out. `http://donottrack.us/`.

[8] EFF - An Open Letter to Facebook CEO Mark Zuckerberg. `https://www.eff.org/files/filenode/social_networks/openlettertofacebook.pdf`.

[9] EFF - Privacy Badger. `https://www.eff.org/privacybadger`.

[10] Electronic Frontier Foundation - Verizon Injecting Perma-Cookies to Track Mobile Customers, Bypassing Privacy Controls. `https://www.eff.org/deeplinks/2014/11/verizon-x-uidh`.

[11] Explicit Proxies for HTTP/2.0. https://tools.ietf.org/id/draft-rpeon-httpbis-exproxy-00.txt.

[12] Explicit Trusted Proxy in HTTP/2.0. https://tools.ietf.org/id/draft-loreto-httpbis-trusted-proxy20-01.txt.

[13] Facebook - A new way to control the ads you see on Facebook. https://www.facebook.com/notes/facebook-and-privacy/a-new-way-to-control-the-ads-you-see-on-facebook/926372204079329.

[14] Facebook - How many Pages can I like? https://www.facebook.com/help/?faq=116603848424794.

[15] Facebook Blocker. http://webgraph.com/resources/facebookblocker/.

[16] Facebook Fact Sheet. https://newsroom.fb.com/company-info/.

[17] Facebook Graph API. http://developers.facebook.com/docs/reference/api/.

[18] Facebook Like Button Count Inaccuracies. http://faso.com/fineartviews/21028/facebook-like-button-count-inaccuracies.

[19] Facebook Plugins. http://developers.facebook.com/docs/plugins/.

[20] Federal Trade Commission - DoubleClick Inc. Closing Letter. https://www.ftc.gov/sites/default/files/documents/closing_letters/doubleclick-inc./doubleclick.pdf.

[21] Federal Trade Commission - DoubleClick Inc. Complaint by the Electronic Privacy Information Center. https://epic.org/privacy/internet/ftc/DCLK_complaint.pdf.

[22] Firefox Sync. http://www.mozilla.org/en-US/mobile/sync/.

[23] Google - Data Saver Proxy. https://developer.chrome.com/multidevice/data-compression.

[24] Hypertext Transfer Protocol 1.1. https://www.ietf.org/rfc/rfc2616.txt.

[25] Hypertext Transfer Protocol Version 2 (HTTP/2). https://tools.ietf.org/html/rfc7540.

[26] Indexed Database API. `http://www.w3.org/TR/IndexedDB/`.

[27] ISPs Removing Their Customers' Email Encryption. `https://www.eff.org/deeplinks/2014/11/starttls-downgrade-attacks`.

[28] MDN - Intercepting Page Loads. `https://developer.mozilla.org/en/XUL_School/Intercepting_Page_Loads`.

[29] MDN - Pageshow Event. `https://developer.mozilla.org/en/using_firefox_1.5_caching#pageshow`.

[30] MDN - window.postMessage. `https://developer.mozilla.org/en/DOM/window.postMessage`.

[31] MDN - XML User Interface Language. `https://developer.mozilla.org/En/XUL`.

[32] MIT technology review - Facebook's like buttons will soon track your web browsing to target ads. `https://www.technologyreview.com/s/541351/facebooks-like-buttons-will-soon-track-your-web-browsing-to-target-ads/`.

[33] Mozilla - Tracking Protection. `https://developer.mozilla.org/en-US/Firefox/Privacy/Tracking_Protection`.

[34] Mozilla At a Glance. `http://blog.mozilla.org/press/ataglance/`.

[35] No Likie. `https://chrome.google.com/webstore/detail/pockodjapmojcccdpgfhkjldcnbhenjm`.

[36] NoScript. `https://addons.mozilla.org/en-US/firefox/addon/noscript/`.

[37] Official Gmail Blog - Staying at the forefront of email security and reliability. `http://gmailblog.blogspot.com/2014/03/staying-at-forefront-of-email-security.html`.

[38] ProPublica - It's Complicated: Facebook's History of Tracking You. `https://www.propublica.org/article/its-complicated-facebooks-history-of-tracking-you`.

[39] QUALYS SSL Labs - SSL Server Test. `https://www.ssllabs.com/ssltest/`.

[40] ShareMeNot. `http://sharemenot.cs.washington.edu/`.

[41] sslstrip. `http://www.thoughtcrime.org/software/sslstrip/`.

[42] The Chromium projects - Sync. `http://www.chromium.org/developers/design-documents/sync`.

[43] The Official Microsoft Blog - Protecting customer data from government snooping. `http://blogs.microsoft.com/blog/2013/12/04/protecting-customer-data-from-government-snooping/`.

[44] The Platform for Privacy Preferences Specification. `http://www.w3.org/TR/P3P/`.

[45] The Transport Layer Security (TLS) Protocol Version 1.2. `https://tools.ietf.org/html/rfc5246`.

[46] The Washington Post - NSA infiltrates links to Yahoo, Google data centers worldwide. `https://www.washingtonpost.com/world/national-security/nsa-infiltrates-links-to-yahoo-google-data-centers-worldwide-snowden-documents-say/2013/10/30/e51d661e-4166-11e3-8b74-d89d714ca4dd_story.html`.

[47] The Washington Post - NSA uses Google cookies to pinpoint targets for hacking. `https://www.washingtonpost.com/news/the-switch/wp/2013/12/10/nsa-uses-google-cookies-to-pinpoint-targets-for-hacking/`.

[48] Time Magazine - One Minute on Facebook. `http://www.time.com/time/video/player/0,32068,711054024001_2037229,00.html`.

[49] Tor Meek. `https://trac.torproject.org/projects/tor/wiki/doc/meek`.

[50] Tracking the FREAK Attack. `https://freakattack.com/`.

[51] Transport Layer Security (TLS) Extensions. `https://www.ietf.org/rfc/rfc3546.txt`.

[52] Uniform Resource Identifier. `http://www.ietf.org/rfc/rfc3986.txt`.

[53] Widgets Distribution. `http://trends.builtwith.com/widgets`.

[54] CVE-2014-0160, 2014. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160`.

[55] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014.

[56] G. Aggrawal, E. Bursztein, C. Jackson, and D. Boneh. An analysis of private browsing modes in modern browsers. In *Proceedings of 19th USENIX Security Symposium*. USENIX Association, 2010.

[57] M. Akhoondi, C. Yu, and H. V. Madhyastha. LASTor: A Low-Latency AS-Aware Tor Client. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012.

[58] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. On the Security of RC4 in TLS. In *Proceedings of the 22nd USENIX Conference on Security*. USENIX Association, 2013.

[59] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*. ACM, 2001.

[60] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, S. Zanella-Beguelin, J.-K. Zinzindohoue, and B. Beurdouche. FREAK: Factoring RSA Export Keys, 2015. `https://www.smacktls.com/#freak`.

[61] E. Y. Chen, J. Bau, C. Reis, A. Barth, and C. Jackson. App isolation: Get the security of multiple browsers with just one. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM, 2011.

[62] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet. Revealing middlebox interference with tracebox. In *Proceedings of the 2013 Conference on Internet Measurement Conference*. ACM, 2013.

[63] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*. USENIX Association, 2004.

[64] T. Duong and J. Rizzo. Here Come the XOR Ninjas, 2011. `https://bug665814.bugzilla.mozilla.org/attachment.cgi?id=540839`.

[65] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman. Neither Snow Nor Rain Nor MITM ... An Empirical Analysis of Email Delivery Security. In *Proceedings of the ACM Internet Measurement Conferene*. ACM, 2015.

[66] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson. The security impact of https interception. In *Proceedings of the 2017 Network and Distributed Systems Symposium*. Internet Society, 2017.

[67] P. Eckersley. How unique is your web browser? In *Proceedings of the 10th international conference on Privacy Enhancing Technologies*. Springer, 2010.

[68] M. Egele, A. Moser, C. Kruegel, and E. Kirda. PoX: Protecting users from malicious Facebook applications. In *Proceedings of the 9th Annual IEEE international conference on Pervasive Computing and Communications, Workshop Proceedings*. IEEE Computer Society, 2011.

[69] S. Englehardt and A. Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security*. ACM, 2016.

[70] A. Felt and D. Evans. Privacy protection for social networking platforms. In *Proceedings of the 2008 IEEE Workshop on Web 2.0 Security and Privacy*. IEEE Computer Society, 2008.

[71] M. Fredrikson and B. Livshits. RePriv: Re-envisioning in-browser privacy. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2011.

[72] L. Gao. On Inferring Autonomous System Relationships in the Internet. *IEEE/ACM Transactions of Networking*, 9(6):733–745, 2001.

[73] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of internet hosts. In *Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference*. ACM, 2004.

[74] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design & Implementation*. USENIX Association, 2004.

[75] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th international World Wide Web Conference*. ACM, 2006.

[76] S. Kamkar. Evercookie. `http://samy.pl/evercookie/`.

[77] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe. Towards IP geolocation using delay and topology measurements. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. ACM, 2006.

[78] A. Kobsa. Privacy-enhanced personalization. *Communications of the ACM*, 50:24–33, 2007.

[79] B. Krishnamurthy and C. E. Wills. Characterizing privacy in online social networks. In *Proceeedings of the 1st Workshop on Online Social Networks*. ACM, 2008.

[80] T. Libert. Exposing the Invisible Web: An Analysis of Third-Party HTTP Requests on 1 Million Websites. *International Journal of Communication*, 9:3544–3651, 2015.

[81] M. M. Lucas and N. Borisov. FlyByNight: mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM workshop on Privacy in the Electronic Society*. ACM, 2008.

[82] W. Luo, Q. Xie, and U. Hengartner. FaceCloak: An architecture for user privacy on social networking sites. In *Proceedings of the international conference on computational science and engineering*. IEEE Computer Society, 2009.

[83] J. R. Mayer and J. C. Mitchell. Third-Party Web Tracking: Policy and Technology. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012.

[84] D. S. Miller, 2012. `http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=89aef8921bfbac22f00e04f8450f6e447db13e42`.

[85] B. Moller, T. Duong, and K. Kotowicz. This POODLE Bites: Exploiting the SSL 3.0 Fallback, 2014. `https://www.openssl.org/~bodo/ssl-poodle.pdf`.

[86] K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Proceedings of W2SP 2012*. IEEE Computer Society, 2012.

[87] G. Nakibly, J. Schcolnik, and Y. Rubin. Website-targeted false content injection by network operators. In *Proceedings of the 25th USENIX Security Symposium*. USENIX Association, 2016.

[88] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira. Measuring and Mitigating AS-level Adversaries against Tor. In *Proceedigns of the Network and Distributed System Security Conference*. Internet Society, 2016.

[89] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2001.

[90] N. Perlroth. NYTimes - China Is Said to Use Powerful New Weapon to Censor Internet, 2015. `http://www.nytimes.com/2015/04/11/technology/china-is-said-to-use-powerful-new-weapon-to-censor-internet.html`.

[91] A. Porter Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz. Measuring HTTPS adoption on the web. In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, 2017.

[92] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. In *Proceedings*

*of the first workshop on Hot topics in understanding Botnets*. USENIX Association, 2007.

[93] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting In-flight Page Changes with Web Tripwires. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2008.

[94] F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012.

[95] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design, 1984.

[96] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: Informed Internet Routing and Transport. *IEEE Micro*, 19(1):50–59, 1999.

[97] M. Sherr, A. Mao, W. R. Marczak, W. Zhou, B. T. Loo, and M. Blaze. A3: An Extensible Platform for Application-Aware Anonymity. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, 2010.

[98] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. ACM, 2012.

[99] K. Singh, S. Bhola, and W. Lee. xbook: Redesigning privacy control in social networking platforms. In *Proceedings of the 18th USENIX Security Symposium*. USENIX Association, 2009.

[100] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. Raptor: Routing attacks on privacy in tor. In *Proceedings of the 24th USENIX Security Symposium*. USENIX Association, 2015.

[101] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2002.

[102] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings of the 17th Network and Distributed System Security Symposium*. Internet Society, 2010.

[103] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. `https://arxiv.org/abs/1111.4503`.

[104] N. Weaver, C. Kreibich, M. Dam, and V. Paxson. Here be web proxies. In *Proceedings of the 15th International Conference on Passive and Active Measurements*. Springer, 2014.