Privacy and Performance in Social Overlay Networks

Tomas Isdal

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2012

Reading Committee:

Thomas E. Anderson, Chair

Arvind Krishnamurthy, Chair

Tadayoshi Kohno

Program Authorized to Offer Degree:

Computer Science and Engineering

University of Washington

Abstract

Privacy and Performance in Social Overlay Networks

Tomas Isdal

Co-Chairs of the Supervisory Committee:
Professor Thomas E. Anderson
Computer Science and Engineering
Professor Arvind Krishnamurthy
Computer Science and Engineering

The core Internet protocols were not designed to protect the privacy of content or the identities of communicating users, thus making censorship and surveillance easy. The same problem persists at higher level protocols, popular peer-to-peer networks are trivial to monitor even for an adversary with limited resources. In fact, recent developments suggest that censorship and surveillance of Internet users is becoming more prevalent over time.

This dissertation describes the design, implementation, and evaluation of two systems that can bring us towards a future where censorship and surveillance is hard. *OneSwarm* is a privacy-preserving data sharing network designed to give users performance comparable to the peer-to-peer networks commonly used today but without revealing their behavior to third party surveillance. Unblock is an overlay network that leverages many of the same building blocks but is designed to help users circumvent censorship of Internet services. Common to both systems is the use of existing social trust between participants to thwart surveillance and censorship respectively. These systems are designed to run on today's Internet and require no changes to core Internet infrastructure or protocols. Measurements of the systems in the wild, and simulations of their behavior at scale, show that they protect user privacy and improve performance over existing alternatives.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

During my graduate career I have been fortunate to be able to work with an outstanding pair of advisors, Tom Anderson and Arvind Krishnamurthy. Tom has always encouraged me to aim for high impact work. He has an incredible eye for interesting problems and has guided me through the research process. Arvind is always wonderfully optimistic making it a joy to come in to the lab. He has an ability to find elegant solutions to problems, even those I have been stuck on for a long time. His capacity to, overnight, hack up a solution to an seemingly impossible problem is invaluable as a paper deadline approaches. I can not imagine a better combination of advisors.

I also need to thank my collaborators, most recently Raymond Cheng, Michael Piatek, and Will Scott. Without their contributions the work presented in this thesis would not have been possible. I would also like to thank other researchers I have co-authored papers with, including Colin Dixon, Harsha Madhyastha, and Arun Venkataramani.

During my time in graduate school I have benefitted from working in an lab full of talented graduate students. Ivan Beschastnikh, Michael Buettner, Lisa Glendenning, Daniel Halperin, John P. John, and Ethan Katz-Bassett have always been open to listen to research ideas or to be testers my (buggy) research software. They have provided valuable feedback. Being a part of a larger research group that shares ideas and result has allowed me to make progress much faster. As a result of that collaboration this thesis contains results, text, figures, and tables from a set of jointly authored publications [44, 45, 46, 58, 69, 70].

Last I would like to thank my family, especially my wife Deidre for her support during this process. I would not have been able to get through the ups and downs of graduate school without her encouragement and understanding.

Chapter 1
INTRODUCTION

The core Internet protocols were designed at a time when Internet communication primarily consistent of point-to-point conversations between large servers. Internet usage has changed dramatically since then. Billions of devices are connected to the Internet, ranging from super-computers to laptops to light-switches. The usage of the network has changed as well. Today's Internet users share text, voice, images, and video; sometimes data is shared with the entire world, sometimes with a few select friends, and sometimes data is shared in such a way that the original source or destination of the data would prefer to stay hidden both from each other and from observers in the network.

This shift in usage has made cloud services popular for communication and data sharing. Users upload their data to a centralized provider who then allows the user to share and access the data from anywhere. Centralized cloud providers handle everything from video and photo sharing to data collection from simple devices in the home that report water and electricity use. However, using the cloud has its downsides. For example: many popular providers collect, store, and share vast amounts of data about their users, raising privacy concerns. Even if we trust the cloud provider with our data, centralization makes censorship easier, a practical concern in many places around the globe.

Users unwilling to sign their data over to cloud providers can instead use decentralized peer-to-peer networks for their data sharing. Unfortunately, even peer-to-peer networks can compromise user privacy. Popular peer-to-peer systems not only leak the source of data but also which users that subsequently access it. The agents monitoring these networks range from researchers trying to better understand network properties [69], to content providers trawling for copyright infringement [31]. Either way, users are often unaware to the wealth of information being collected about themselves when they share or access content over the Internet, independent on whether they use a cloud provider or a peer-to-peer network.

The distributed and global reach of the Internet makes it useful not only for sharing personal data but also for news gathering and political organization. However, the free flow of information is not universally considered a good thing. Governments have shown a willingness to selectively restrict the content, source, and destination of communication to advance their social and economic agendas [42, 50]. Simultaneously, network equipment providers have seized the opportunity to profit from censorship by selling network interception and filtering devices [84]. Internet censorship is a reality for the majority of Internet users [64, 102].

Censorship is no longer just about disrupting activists in nations with oppressive governments. Rather, censorship describes the growing shift of nation-states exerting power in telecommunications networks to influence commonplace communication. Selectively cropping content from a person's daily news feed and their friend's status updates can significantly alter the way they perceive the world [10, 18]. By restricting the freedom of the press, governments can distort views, change facts, and affect public opinion. Internet censorship also allows nations to exert economic pressure [23] by blocking foreign competitors from entering markets and enabling new forms of protectionism in the multi-trillion dollar [8] Internet economy.

The prevalence of surveillance and censorship has created a demand for anonymizing and censorship defeating technologies. Proposals range from a complete redesigns of the core Internet protocols [56] to more modest commercial solutions involving one-hop proxies [6] and VPN connections [43] designed to hide the true originator of requests while tunneling traffic to a location where the desired content is not censored. These solutions have shortcomings, even simple changes to the Internet can take decades to gain foothold making a complete redesign unlikely to occur soon, and single hop solutions requires the user trust the provider to not log user requests.

Building an anonymizing and censorship-resistant network without replacing the existing network architecture requires an overlay built on top of the existing Internet. Tor [27] and Freenet [20] are example of existing overlays that are in use today. Tor offer anonymity by iteratively decrypting requests as they are forwarded through multiple overlay hops. When the request reach the last hop in the overlay the clear text is visible and is forwarded to the intended destination on the Internet. This process provides privacy by hiding the content and original source from the destination service as well as from monitoring agents in the network which makes censorship harder. Freenet provides privacy by separating the process of publishing and serving data. Publisher use an algorithm to select a few nodes that are responsible for storing a specific object. The publisher then push the content into the overlay through its neighboring nodes, this process is repeated over several hops until the intended storage node is reached.

Both Tor and Freenet makes it hard for an attacker to determine the source, destination, and content of messages, but as I show later in this dissertation they offer comparatively poor performance due to long overlay paths, oversubscription, and transport layer congestion. Good performance is critical for user adoption as even a small increase in latency cause users to abandon the service [39].

In this dissertation I present the design, implementation, and evaluation of two systems that are deployable on the Internet today. OneSwarm is a privacy-preserving data sharing network designed to give users performance comparable to the privacy-revealing peer-to-peer networks commonly used today but without revealing their behavior to third party surveillance. Unblock is a overlay network that share many of the same technologies with OneSwarm, but is designed to circumvent censorship of Internet services.

Common to both OneSwarm and Unblock is the use of existing social trust between participants to thwart surveillance and censorship respectively. Basing the overlay on links backed by social trust introduce a set of challenges. First there has to be a way to bootstrap new users, both OneSwarm and Unblock allows users to optionally add untrusted links until a sufficient number of trusted links are added. Second, overlay paths lengths can not be controlled by the system, instead they are bound to the social network formed by the participants. To overcome this challenge both systems make aggressive use of multi-path routing to, in the case of OneSwarm, maximize throughput, and in the case of Unblock, minimize latency. Together with new transport layer techniques both OneSwarm and Unblock improve performance over existing systems.

1.1 OneSwarm: Privacy preserving peer-to-peer data sharing

Most Internet users are both content consumers and content producers, with their data shared with others through centralized data sharing sites such as Facebook, YouTube, and Flickr. However, most popular web sites collect, store, and share vast amounts of personal information about their users, despite users finding such behavior objectionable [95].

Peer-to-peer systems potentially provide an alternative for achieving scalable file sharing without a trusted web site as mediator. However, the peer-to-peer systems available today offer users an unattractive choice between privacy and reasonable performance. On one side, protocols like BitTorrent are high performance and robust, but participants are easily monitored by anyone who cares to look [70, 71, 86]. On the other, anonymization networks (e.g., Tor and Freenet) emphasize privacy, but offer comparatively poor performance.

With OneSwarm, I explore a new design point in this tradeoff between privacy and performance. The goal is to provide make systematic monitoring much more challenging than before, while maintaining good enough performance that users turn on privacy by default for all of their data sharing. Central to the design is a notion of *flexible privacy*. OneSwarm does not adopt a universal guarantee regarding information exposure; each individual user is free to control the tradeoff between performance and privacy by managing trust in peers as well as sources of peers. User can bootstrap overlay connectivity by manually approving only trusted friends, automatically importing peers by piggy-backing on existing social networks, and/or chose to be bootstrapped by a matching service that provides a random set of untrusted peers.

Restricting sharing to only trusted contacts provides few avenues of attack for would-be monitoring agents, but it can be an obstacle to early adopters who by definition have no one in the system they trust. Alternatively, untrusted peers improve performance and availability by increasing redundancy, but widen the attack surface.

While stronger restrictions on user behavior permit stronger statements of system-wide security properties, deployment experience has shown that support for divergent, individual notions of privacy is essential for adoption. In the year since its release, OneSwarm has been downloaded hundreds of thousands of times, translated to more than half a dozen languages, and it is in active daily use by thousands of people world-wide.

1.2 Unblock: Blocking-resistant network services

Unblock is a blocking-resistant overlay network that allows users to reroute Internet traffic to avoid censorship, as well as to directly host blocking-resistant services. Users of the system explicitly connect to friends who they trust to conceal their identity. Friends form encrypted links with each other, forming a global social network. Unblock routes Internet traffic over these links to a region where the desired content is not censored. Routing, coupled with security mechanisms to prevent overlay disruption, hides overlay participants from the censor.

Unblock improves social routing performance by introducing shortcut links, untrusted connections that risk exposing a small set of users to an adversary in order to dramatically lower the median latency of the system as a whole. The system also enables each user to discover multiple paths to exit nodes for better path diversity and higher bandwidth. Users communicate using a hybrid transport protocol, consisting of a SSL control plane and an encrypted UDP data plane. The transport layer protocol use per-hop congestion control to minimize multi-hop queuing and to improve aggregate network utilization, and adaptive use of multiple paths to minimize latency or maximize throughput.

Measurements of the Unblock client show that its design provides better performance than competing solutions. Simulations of the system at scale find that shortcut links improve connectivity without introducing significant vulnerability or network hot spots.

1.3 Thesis and Contributions

The goal of this dissertation is to support the following thesis: *By building overlay networks based on social trust we can improve security and performance relative to existing solutions.*

The fundamental contributions presented in this dissertation include:

- **The design, implementation, and evaluation of a file distribution protocol for social network overlays.** I have designed, implemented, and evaluated OneSwarm, a file sharing protocol designed for data sharing in a social overlay network. The file distribution protocol gives the user complete control over their data, and allows them to share data only with friends and family, or with everyone, while still preserving their privacy. I have evaluated the protocol both in the wild, and in a simulator to evaluate the system at large scale. The results show that social network overlays can support a file sharing workload while both protecting user privacy and maintaining performance.

- **A software artifact, OneSwarm.** The OneSwarm client is used by thousands of active users each week for communication over a social network overlay. The software binary as well as related source code is available to the public.

- **The design, implementation, and evaluation of a censorship circumventing system based on social network overlays.** I have designed, implemented, and evaluated Unblock, a protocol designed to help user circumvent Internet censorship. Unblock allows users to access Internet services that otherwise would be unavailable due to censorship. Through measurements of the implementation and simulations of the system as large scale I show that social network overlays can get around a censor while at the same time improving performance relative to public overlays based on onion routing.

1.4 Organization

The remainder of this dissertation is organized as follows. Chapter 2 provides the reader with a background of the area as well as motivation. The contributions are described in Chapter 3 and Chapter 4. In Chapter 5 I discuss related work followed by conclusions and future work in Chapter 6.

Chapter 2

BACKGROUND AND MOTIVATION

In this chapter I introduce the terminology used in the remainder of this dissertation. I then the present various methods used by governments and private entities to monitoring and filtering Internet traffic, tools designed to aid users that wish to avoid it, and issues with those tools. Last I describe the methods I use to improve on them.

2.1 Surveillance and censorship

The core Internet protocols, such as TCP/IP, were not designed to protect the content and authenticity of communication or to hide the participants. Rather it assumed that it was sufficient to do that on top of TCP. However, merely the existence of communication between two endpoints can reveal information or lead to *censorship* where select communication is altered or blocked. Protecting against this was not a priority as in the early days of the Internet there was no infrastructure around to monitor and analyze the traffic. Things have changed since then. In Europe Internet *surveillance* is mandated by the Data Retention Directive. It requires phone companies and Internet Service Providers (ISPs) to store information about emails, text messages, phone calls, and search engine traffic for 24 months [67]. While the corresponding US law is less draconian, reports about warrantless wiretapping by the NSA in collusion with ISPs suggest that the extent of the surveillance is larger than what is officially stated [35]. The situation in non-democratic countries is even worse with monitoring and filtering all Internet traffic being commonplace [15, 13].

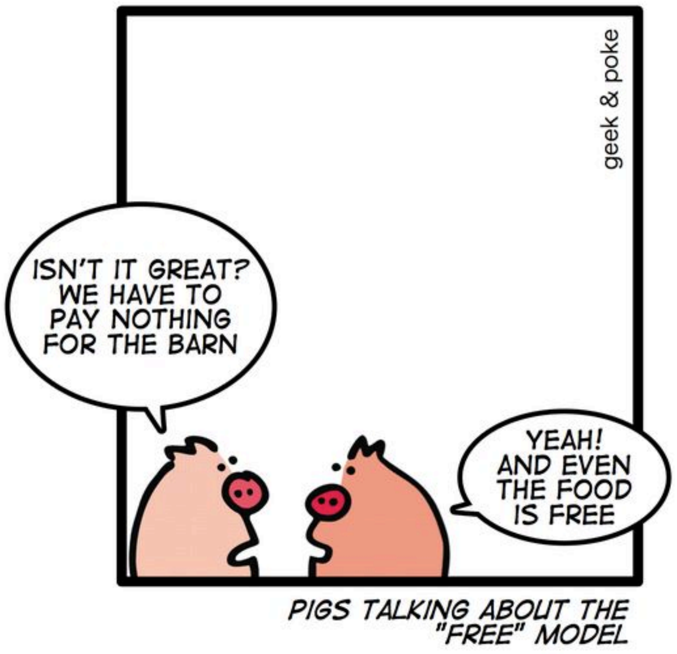
geek & poke
ISN'T IT GREAT? WE HAVE TO PAY NOTHING FOR THE BARN
YEAH! AND EVEN THE FOOD IS FREE
PIGS TALKING ABOUT THE "FREE" MODEL

Figure 2.1: The "free" model. Created by Oliver Widder [100]. License: CC BY-SA 3.0

2.2 Surveillance and censorship in the wild

Governments are not the only entities interested in monitoring and censoring Internet users. Private companies collect user data to improve their bottom line. In this section I will first look at how private companies track users. I will then look at how governments monitor Internet traffic with the goal of preventing users from accessing content deemed undesirable.

2.2.1 Web tracking

"If you're not paying for something, you're not the customer;
you're the product being sold." –Andrew Lewis

The business model of many Internet services is based on the "free" model; users do not need to pay anything to access the service, instead the service will show advertisements. The advertisements are provided by advertising networks. By serving ads on a large number of webpages it is possible for these networks to track user behavior across multiple services [79]. The services share information about users to increase their ad revenue, increasing the amount of data the ad networks knows about each user. With this tracking it is possible to, over time, build up a detailed user profile for each visitor. The profiles contain information such as demographics, geographical location, hobbies and interests, and items that the user might be interested in buying [11]. These profiles are then bundled up and sold [5].

The more a service knows about a person, the better it can target advertising. In Europe people have the right to request all information a service have collected on them. When requesting information about their profile from Facebook individuals have received dossiers hundreds of pages long, sometimes including information they had deleted [83].

2.2.2 Censorship

Most countries around the world enforce some type of Internet censorship [42, 102]. In this section I will give a brief introduction to the methods used. Figure 2.2 shows an overview of how a censor can block web requests.

DNS name filtering

DNS name filtering is one of the simplest forms of censorship. It requires no modification to the core Internet infrastructure and only simple policy changes to DNS servers. The censor only monitors DNS queries as they arrive at the DNS server, there is no need to all monitor traffic flowing through the network. In northern Europe DNS filtering is voluntary and implemented by the ISPs. The government supplies a blacklist of domain names and the ISPs configure their DNS servers to either drop requests for those domains, or to respond with a government site warning the user. The major criticism against this approach is that it is incomplete, in that it only contains the domains discovered by the government, ineffective in that it is easy to get around for determined users, blunt in that

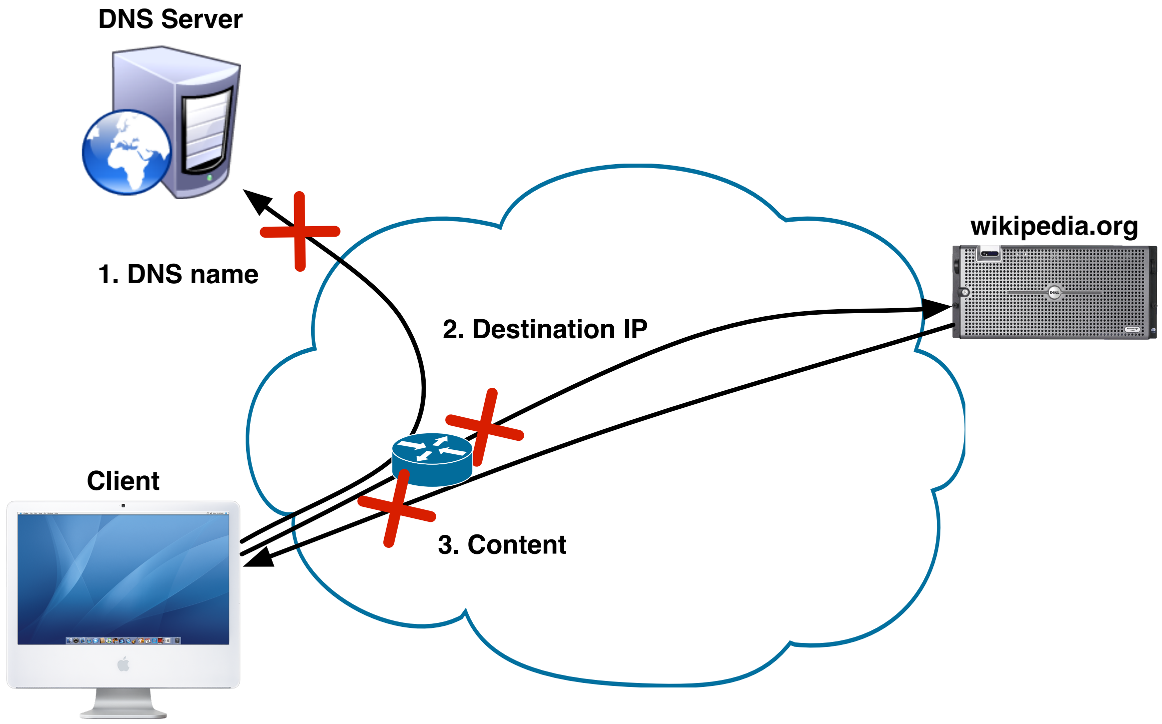

Figure 2.2: An overview of web request blocking. A censor can block requests based on domain name, destination IP address, or because of prohibited content in the request or response payload.

the filter blocks entire domains and not just the pages with prohibited content, that the filter grows to include innocuous sites, and that it gives power to governments to shut down political speech [88].

Getting around DNS filtering requires little skill and no additional software. Instead of instructing the operating system to use the filtering DNS server provided by the ISP, the user can connect to a public DNS server that does not implement the filter. The most popular public free DNS servers are run by Google (8.8.8.8) and OpenDNS (208.67.222.222). Because it is so easy to get around, DNS based filtering is at best preventing users from accidentally accessing prohibited content.

Destination IP filtering

Destination IP filtering requires more extensive changes to the Internet infrastructure than DNS based filtering. Filtering based on IP address makes circumvention more difficult as it has the ability to prevent access to peer-to-peer networks and to anti-censorship tools such as Tor. IP filtering is more costly than DNS filtering in that it requires the censor to monitor the packet headers of all network traffic. The filter can either be installed at the country border, with the advantage that only the few routers with cross country links need to implement the policy, or it can be pushed down into the network closer to the end user. Pushing the filter closer to end users means that domestic traffic can be filtered as well, but the downside is that a larger number of routers needs to be updated with the filtering policy [88].

Content filtering (deep packet inspection)

Deep packet inspection (DPI) means that the censor examines the entire packet payload as it passes through the network. From a censor's point of view DPI has a couple advantages. It can be dynamic: for example it can filter all traffic containing certain words no matter the source or protocol. It can be arbitrarily fine-grained, instead of blocking entire web sites it can block individual pages, parts of pages, or modify pages by removing prohibited content. End-to-end encryption such as TLS prevents inspections of packet payloads so in those cases the filtering policy can only be based packet headers and TLS handshaking information that is sent in the clear. Even with this limitation DPI has been used to fingerprint encrypted protocols, such as Tor, based on unique TLS handshake properties [26].

The main disadvantage with DPI is that it requires specialized hardware to run at high speed. To decrease hardware costs a censor can use a hybrid IP filter / DPI solution. Packets to network addresses previously flagged as suspicious are redirected through DPI, while the remaining network traffic is forwarded without inspection [88].

2.2.3 Case study: Australia - Wikileaks blocked for publishing list of blocked websites

Like many western countries, Australia requires ISPs to censor pages and domains containing child abuse. However, in Australia the list of censored pages has grown over time to include items not originally intended to be censored such as poker portals and pages about euthanasia [104] . In March of 2009, Wikileaks got hold of the block list and published a blog post about how censorship in Australia quietly expanded in scope. The post also contained the entire list of censored pages. Shortly after the blog post Australia added the wikileaks webpage to the block list [57] making the page unreachable to novice Internet users in Australia. However, after popular uproar the Australian government backtracked and in April 2010 it reexamined the censored pages and found that wikileaks did not contain any prohibited material. While sites with a large following can exert enough popular pressure to make the government backtrack this is rarely the case for small sites. Censoring the list of censored pages exaggerates the problem as site owners might not even be aware that their site is censored.

This example demonstrates the potential pitfalls of Internet censorship – once the infrastructure is in place it is easy quietly expand the scope. In addition, just as with DNS filtering, highly motivated and technically savvy individuals can get around the type of censorship deployed, something I will cover next.

2.3 Circumventing surveillance and censorship

The most popular method for protecting user data from surveillance is to use end-to-end encryption such as SSL/TLS, or HTTPS when used over the web. These techniques protect the data from monitoring in the middle of the network, ensure that the content is not tampered with, and that the other party indeed is who they say they are [1]. Importantly, they do not hide the identities communicating, both participants know who the other party is, and an attacker in the middle of the network knows who is communicating with whom.

[1] This is only true if the certificate chain is trustworthy.

One way to hide the fact that you are communicating with a particular host is to use an indirection service. Instead of sending traffic directly to the destination the traffic is sent to the service, the service then scrubs off the source information and forwards the traffic as if the service itself was the source effectively hiding the original source is hidden from the destination. An early indirection service was anonymizer.com but now there are a large number of commercial proxy and VPN solutions available. This solution is partial because the users must put all their trust in the anonymizing service; there is no way to ensure that the service is not saving information about user traffic. In some ways users are even worse of now, before the only information leaked was the IP address[2]. By paying the anonymizing service with credit card the user not only provides them with all her Internet traffic, but also with her name and address.

2.3.1 Tor

Tor aims to correct the shortcomings of the commercial anonymization services. Tor, short for The Onion Router, provides anonymity guarantees by way of onion routing [27]. Figure 2.3 shows an overview of the Tor ecosystem. Tor requires users to install a small client program and configure it as a SOCKS proxy. Upon startup the Tor client contacts a Tor directory server to receive a list of Tor relays and their corresponding IP addresses. The client then selects three relays based on criteria such as exit node policy, bandwidth, uptime, and geographic location [65, 30]. Once the relays are selected the client initiates a *circuit setup* where it in turn contacts each replica in a telescopic fashion. This occurs at each hop until the traffic reaches an exit node, which forwards the traffic to the final overlay-external destination. The initial circuit setup is using public key cryptography but once the circuit is established the client exchange a symmetric key with each replica to improve performance.

By encrypting all traffic to the Tor network the client can maintain anonymity even on a monitored Internet connection. An attacker looking at the traffic will only see encrypted

[2] While an IP usually can be mapped to a specific person, most countries require a court order to release this information.

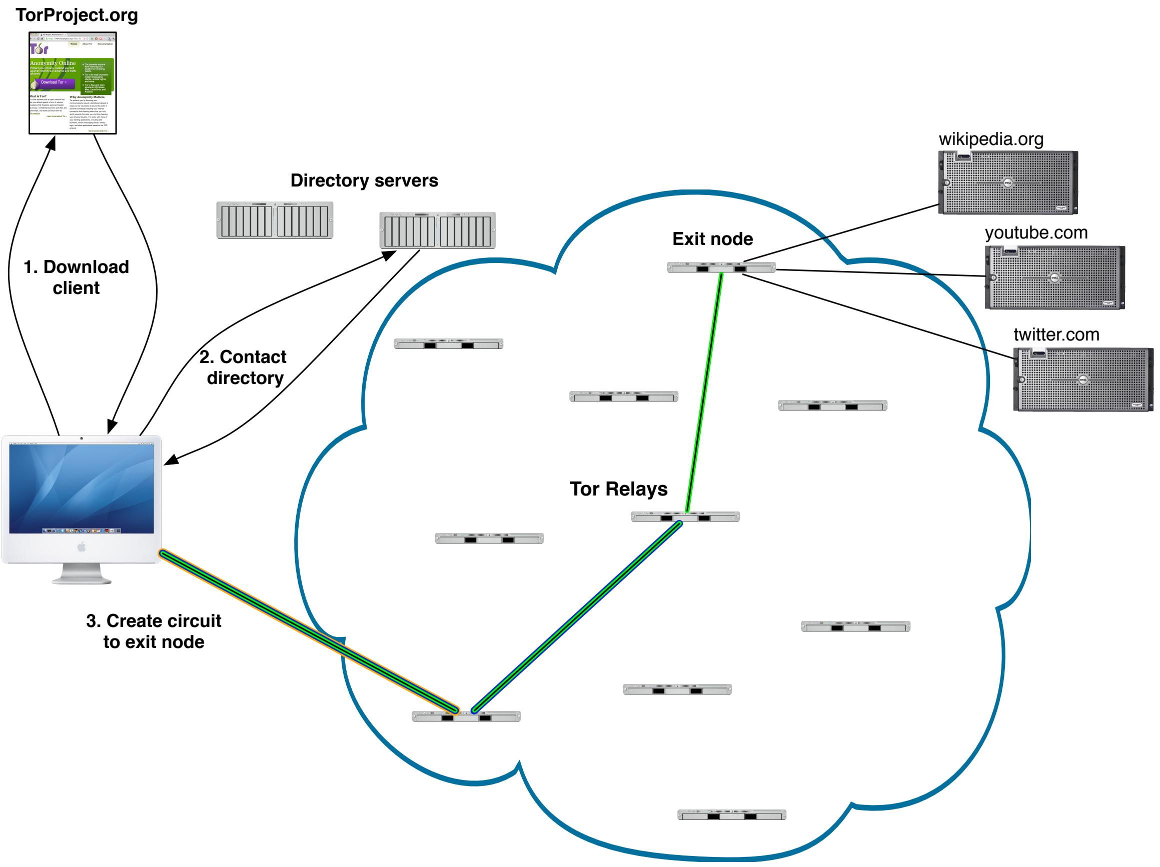

Figure 2.3: Overview of the Tor ecosystem. The user starts by downloading the Tor client. The client contacts a directory server to get a list of all Tor relays, it then selects a sequence of 3 relays with the traffic exiting the Tor network after the last hop. At each a layer of encryption is peeled off, the final destination is only visible to the exit node.

packets destined for the Tor network. In addition the relays only get partial information about the traffic. The first router knows that a users accessing the Tor network but the only information available is the next hop. The middle relay knows that traffic is flowing from a Tor relay to another Tor relay, but not the original source, destination, or content. The last relay can look at the final destination and the content, but have no way of knowing which user or IP address originated the request. From the service providers point of view the only information available is that a Tor relay is contacting it with a request for data.

Circumventing censorship with Tor

In addition to providing anonymity, Tor (described in Section 2.3.1), can by used to circumvent censorship. Since all requests made through Tor originate at the exit node, basic censorship techniques such as DNS filtering and DPI based keyword matching are ineffective against Tor users. To close the loophole governments block Tor using different techniques ranging in complexity.

Blocking torproject.org: The most basic blocking seen in the wild is to block the Tor home page, torproject.org. This makes it harder for new users to download the tool, but existing users are not impacted, once the user has the Tor client installed, she no longer has an dependency on the homepage. Users can share the client binary over email, thumb-drives, or by republishing the binary on their personal blogs; this type of blocking is ineffective except against novice users [26].

Blocking directory servers and relays: Instead of blocking access to the client binary, some governments prevent the clients from accessing the Tor directory [26]. The Tor developers responded by instructing all Tor relays to also provide access to the Directory allowing users with access to any relay to get access to the network. The next step made by censors was to periodically fetch the Tor directory and block all IPs in it. The Tor developers continued the arms race with the introduction of Tor bridges. Bridges are special relay nodes that are unlisted in the public directory, their location is instead communicated to users by other means. I investigate the effectiveness of Tor bridges in Section 4.1.1.

2.3.2 Freenet

Freenet [20] is a peer-to-peer network designed to support anonymous publication and distribution of information. Only data specifically shared within the Freenet overlay is accessible, this differs from Tor where client traffic can be forwarded to legacy Internet services. Freenet operates in two different modes, the OpenNet mode that is free for anyone to join, and the *dark mode* where all connections are between trusted social links. Using

Freenet in dark mode limits exposure to third parties. Only trusted friends and attackers with access to network traffic can see that a certain user is an active Freenet user.

Freenet's data storage model is similar to that of a DHT [87, 60]. Each user must reserve some local storage space for use by the system. Each node is responsible for storing data in a certain key-range. All data is addressed by key and the routing algorithm tries to locate the user in charge of a certain key. The same algorithm is used for both insertion and retrieval. Freenet has a caching mechanism to improve performance when fetching popular objects. After a successful content lookup the data is sent back hop-by-hop. Each node on the path will add the object to the short term cache before forwarding it to the next hop. If the node later sees a lookup for the same object it can respond with cached data instead of forwarding the query.

2.3.3 Peer-to-peer and BitTorrent

Users unwilling to trust a cloud service to handle their data sharing can user peer-to-peer networks. Peer-to-peer networks depart from the traditional client-server model used for most Internet services; instead of relying on a centralized server to handle all requests, peer-to-peer merge the client and server roles. Once a client receive a piece of data it becomes a source for that data taking the role of a server. Leveraging client resource makes peer-to-peer systems scalable. The resources of the original source no longer limits the total amount of resources available. Because of this property, a peer-to-peer system allows its users to share files without any need for expensive infrastructure. BitTorrent is the most popular peer-to-peer system in use today and as of the fall of 2011 BitTorrent accounts for 13.5% of total Internet traffic in North America [81].

BitTorrent overview

Native BitTorrent is not a global network but is made up of *swarms* where each swarm contains the peers interested in a particular data item. *Trackers* handle swarm membership, each tracker maintains a list of swarms registered with that specific tracker as well as the peers associated with each swarm. When a peer joins a swarm it will issue a query to the tracker requesting a random subset of the peers in a particular swarm. Upon receiving the tracker response, peers connect to the other swarm members and initiate the data transfer.

Data is split into blocks commonly in the 64KB-2MB size range. When a peer completes a block download it will verify the block hash against a list of block hashes published by the source in a .torrent file. This protects the swarm against data corruption due to hostile peers and faulty network equipment. Once a peer completes a block it becomes a source of that block to other peers.

Monitoring of BitTorrent BitTorrent is not designed to protect the privacy of its users, and it is simple to set up systems that monitor large numbers of users. As an example, during a research project designed to measure the impact of incentives in BitTorrent I could discover over 300,000 BitTorrent users in 48 hours [44, 58, 69]. With repeated queries to the tracker it is possible to crawl the membership of individual swarms and the IP addresses associated with each peer. While the primary interest in this specific case was to measure the Internet connection properties of the users it is trivial to modify the experiment to instead record the name of the files transferred in each swarm. In fact, more recent work performed by LeBlond et al. collected statistics of BitTorrent users covering 148 million IPs over a period of 103 days [53]. With that level of surveillance it is possible to also deduce the initial source of the data, in addition to the set of people downloading each item.

It is not only researchers who have discovered how easy it is to monitor file sharing networks. An example is *YouHaveDownloaded* [91], an online service that lists the files downloaded by a particular IP address. YouHaveDownloaded claims to monitor 20% of BitTorrent downloads using public trackers. The creator or the site, Suren Ter, says he started it as a joke to show to the public how easy it is to track people on the Internet.

Rights holders are also monitoring peer-to-peer networks for copyright infringement. By outsourcing copyright enforcement to firms specializing in peer-to-peer surveillance they can find all users sharing a specific data item leading to lawsuits with tens of thousands of defendants [31]. It is unknown if these firms only monitor the specific content they are instructed to track, or if they seek a more global view allowing them to proactively seek out clients based on observed usage statistics. Either way, monitoring of popular peer-to-peer systems is rampant on the Internet today giving users the undesirable choice between using a cloud service and thus being tracked by the cloud provider, or, using a peer-to-peer network and risk getting monitored by anyone.

2.3.4 Closed solutions

The Global Internet Freedom Consortium (GIFC) provides a number of closed-source solutions for censorship circumvention. The most popular GIFC tools are Garden, UltraSurf, DynaWeb, GPass, and FirePhoenix. Little public information is available about these proprietary anti-censorship tools, and they use techniques such as code obfuscation, code signing, wrapping, and even resort to using decoys. It is unclear however, whether these systems are robust to monitoring or censorship. Further, closed source solutions preclude community participation and scrutiny, and require the users to implicitly trust the developers to not leak sensitive or private information. A closer look at one of those systems, UltraSurf, has shown that it has architectural and implementation defects [7]. I leave these systems outside the scope of this thesis.

2.4 Social overlay networks

In this dissertation I explore how to design overlay networks where trust from the users existing social network is used to improve both the privacy guarantees and the performance of overlay network. I call these overlays *social overlay networks*. Basing the overlay topology on the social network of the users makes it possible for system to take the trust users have in each other into account. In the case of OneSwarm the system relies on trusted friends to not reveal the source of data when forwarding requests. In the case of Unblock users trust their friends to not expose their network address to the censor.

Limiting the topology to just the social network works well for users with a large number of friends in the system, but users with few connections suffer from poor performance in the best case, and complete disconnection from the overlay in the worst. Both OneSwarm and Unblock provide users with an optional mechanism for adding *untrusted* peers. Naturally, the addition of untrusted links worsen the security guarantees of the users that chose to use them as user cannot be certain the the untrusted peer is not malicious. Protocol behavior towards untrusted peers must therefore be different towards untrusted versus trusted peers. Both OneSwarm and Unblock behave differently towards untrusted peers to protect the user from attacks.

In practice the overlay contains a mix of trusted and untrusted links. The structure of the overlay means that there often exists a multiple possible paths between any two users. While the performance of an single path is poor and unreliable, both OneSwarm and Unblock implements search and routing algorithms and a transport layer capable of using multiple paths simultaneously. In the next chapter I describe how the reliance of social network links, the addition of untrusted links, and the use of multi-path impacts the design, security, and performance of OneSwarm.

Chapter 3

ONESWARM

This chapter describes the OneSwarm data sharing protocol, implementation, and evaluation. OneSwarm explores a new design point in the tradeoff between privacy and performance. It aims to provide much better performance than Tor and Freenet and much better privacy than BitTorrent. The goal is to provide good enough performance that users turn on privacy by default for all of their non-public data sharing.

Data objects shared via OneSwarm are located and transferred using disposable, temporary addresses and routed indirectly through an overlay mesh, providing resistance to the systematic monitoring of user behavior. Content lookup and transfer is congestion-aware and uses multiple overlay paths, providing good performance at reasonable overhead even for rare objects and diverse peer bandwidths.

Central to the design is a notion of *flexible privacy*. OneSwarm does not adopt a universal guarantee regarding information exposure; each individual user is free to control the tradeoff between performance and privacy by managing trust in peers as well as sources of peers. Deployment experience has shown that support for divergent, individual notions of privacy is essential for adoption. In the year since its release, OneSwarm has been downloaded hundreds of thousands of times, translated to more than half a dozen languages, and it is in active daily use by thousands of people world-wide.

The feedback and behavior of this community has guided the evolution of the protocol, driving its design towards increased user control, while nevertheless retaining resistance to systematic third-party monitoring.

In addition to its qualitative impact on design, OneSwarm's user community serves as the basis for the evaluation of the system. I report measurements of data transfers between instrumented clients running on PlanetLab communicating through the OneSwarm mesh. Despite the overhead of providing privacy, OneSwarm's performance is competitive with unanonymized BitTorrent. Furthermore, a novel lookup and transfer technique yield a median factor of five improvement in large file download times relative to Tor and a median factor of twelve improvement relative to Freenet.

Measurements of system properties of OneSwarm are limited by the need to protect the privacy of the users of the system. To gain insight into the behavior of the system at scale, I complement the PlanetLab measurements with a simulations study. For this, I use a trace of the object sharing patterns and social connectivity of more than 1 million users of last.fm, a popular music-focused website that aggregates the playback histories and social network of its users [52]. Trace replay shows that OneSwarm provides high availability, with 95% of satisfiable requests being fulfilled by the overlay during peak load.

3.1 Measurements

Currently, systems builders have limited data with which to evaluate new protocol designs layered on top of social networks. Without measurements of real social graphs and associated workloads, the relevant constraints and properties of the environment are unclear. Recently, significant progress has been made towards measuring and understanding the properties of online social networks [2, 61, 62]. But, these existing studies have focused primarily on graph properties. While essential, graph properties alone are not sufficient. To design and evaluate file sharing protocols built on top of a social graph it is important to understand the sharing behavior of users as well.

In this section, I report measurements of both a social network and a sharing workload. The popular music website last.fm[1] provides both. last.fm builds music related services on top of a database of user listening habits and a social network. These services include artist recommendations, Internet radio, popularity charts, interest groups, etc. To build the

[1]http://www.last.fm/

collective database, last.fm users run custom software that reports the playback history of each user's local music library. While last.fm does not provide any data-sharing features and it does not include user-generated content, the listening habits of users represents a portion of the potential workload for OneSwarm, e.g., for privacy preserving P2P distribution to users with a license to an entire playlist. Basing simulations on this workload allows me to take both the social network as well as the content interests of individual users into account instead of modeling the social network and the content interest independently.

Maintaining a list of friends is not required to use any of last.fm's core functionality. The social network is used to provide communication, notifications, and convenient access to the listening histories of friends. All friend links are symmetric and must be approved by both parties. Also, each user's list of friends and listening history is public.

In the remainder of this section I review my trace methodology and present an analysis of the trace as motivation for my design in the next section. I then return to the trace in Section 3.5 as part of the evaluation for OneSwarm.

3.1.1 Methodology

To measure both the social graph and each user's listening behavior, I used last.fm's public XML-RPC API.

Crawling the social graph: Given a user name, the last.fm API provides a list of that user's friends. I crawled the social graph using a breath first search starting from several manually chosen seed users. Because last.fm acceptable use policy imposes a rate limit on API calls, this process took one month to complete (August 25th – September 25th, 2008).

Crawling the sharing workload: The last.fm API provides a list of song identifiers and play counts for users at a per-week granularity. Along with social links, I collected a trace of these week-long histories for each crawled user for the two weeks prior to the start of our crawl. Although these week-long histories provide a coarse-grained view of usage per user, they do not translate directly to fine-grained trace replay, i.e., at the granularity of minutes. last.fm provides fine-grained information with song identifiers and timestamps as an RSS feed for each user, but monitoring these RSS feeds for all last.fm users exceeds the crawl rate-limit. Instead, I sampled the RSS-feed of 1,000 random users from the crawl and the model of short-term activity uses this trace.

3.1.2 Social network

The crawl discovered 1,768,197 users and 6,325,306 social links. Most users that had social links were in a single large connected component. Because last.fm does not provide a count of all active users, I estimate coverage by sampling users and computing the fraction of these that were observed during the crawl. last.fm provides lists of users per country, and my samples were drawn randomly from the set of all users providing country information.[2] I sampled 8,081 such users of which 4,263 occur in the crawl (53%). Of the remaining users, 92% have no social links. The remaining 8% of users are grouped into small, disconnected clusters. These results suggest that the crawl covers the largest connected component in the social network and that the overwhelming majority of remaining users have no social links.

Degree distribution: Figure 3.1 shows the complementary cumulative distribution function (CCDF) of degrees for all users observed in our trace. The crawl reveals that the majority of users have very low degree. 30% of users have just one social link, the median degree is 3, and 81% of users have 10 or fewer friends in last.fm. This is in many ways the worst case for OneSwarm: reaching the majority of fringe users requires longer average path lengths. Also shown is a best-fit power law distribution ($\alpha = 1.51$) obtained using the maximum likelihood method [21]. The Kolmogorov-Smirnov goodness-of-fit metric for the fit is 0.137. Unlike other social networks, the last.fm degree distribution does not strongly follow a power-law. This is likely to be the case for the OneSwarm social network as well. It is unlikely that any single user will have a trust relationship with millions.

[2]Although I could *sample* such users by screen scraping web pages, *enumerating* all users in this manner violates the last.fm API acceptable use policy.

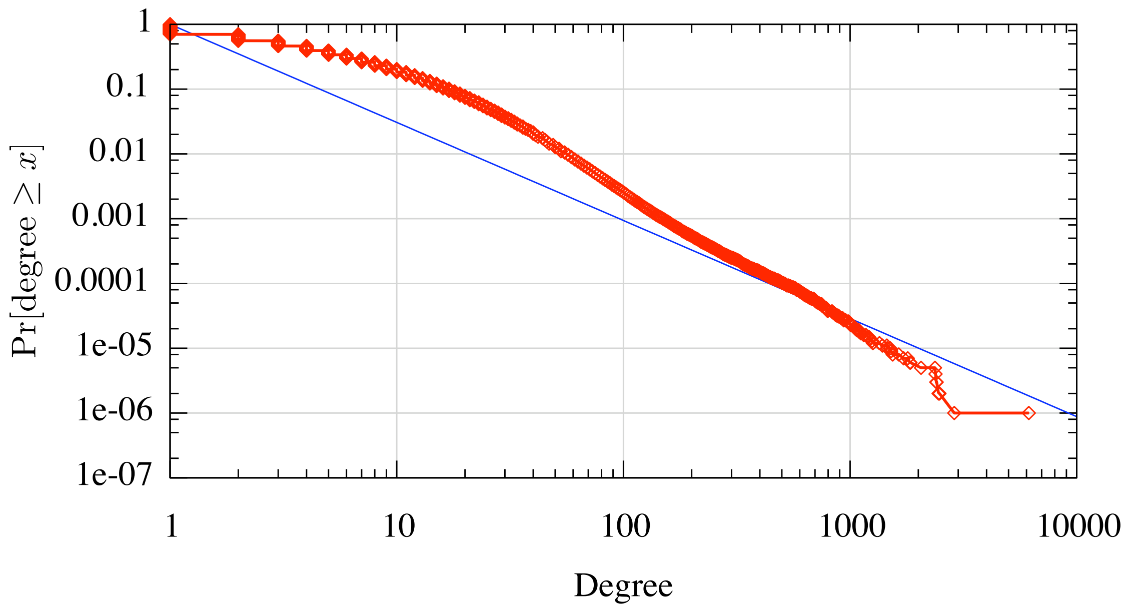

Figure 3.1: Complementary cumulative distribution (CCDF) of degrees for all users in the last.fm trace. A best-fit power law distribution is shown ($\alpha = 1.51$) for comparison.

There is no requirement for users to add their social network to last.fm, most functionality is available even when no friends are specified. This is different than the model in OneSwarm, users where users get improved performance by adding their friends in the system. Because of this I believe the last.fm social graph to be a conservative estimate of how the OneSwarm social network will look at scale.

3.1.3 Resilient core

Social networks tend to have a highly connected core of nodes. For protocols built on social networks, this may hinder both performance and robustness. When available, core nodes may become bottlenecks. When unavailable, path lengths increase, raising overhead and reducing capacity, and some nodes become completely disconnected.

For my purpose of layering a data sharing system on a social network, understanding the structure of the core is crucial for system design. If most paths *necessarily* transit the core, these nodes will need to carefully manage the sharing of their scarce resources. But, if

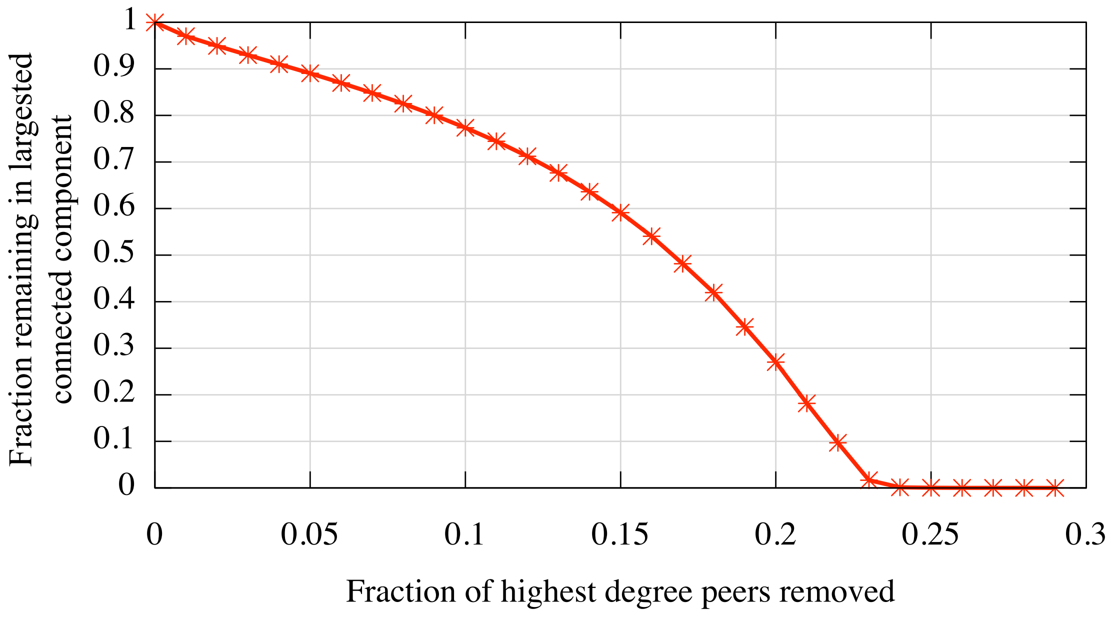

Figure 3.2: The fraction of nodes in the largest connected component of the last.fm social graph (y-axis) as an increasing fraction of high degree nodes are removed (x-axis).

significant path redundancy exists, core nodes can (and should) be avoided during periods of congestion.

To understand which of these effects dominate the following analysis was performed. After removing a fraction of the highest degree nodes from the graph, the resulting connectivity is computed. This removal is then repeated for an increasing fraction of nodes. The results are summarized in Figure 3.2. Connectivity degrades slowly, suggesting the existence of redundant paths around any highly connected nodes. This data differs somewhat from previous studies of online social networks [62]. For example, Mislove et al. showed that in the Flickr social network the connected component fractured completely after the removal of 10% of the highest degree nodes; in contrast, the last.fm social graph fractured after removing 24% of the highest degree nodes. I speculate that this difference is due to last.fm lacking publish/subscribe support for extremely popular nodes; lacking these nodes, the last.fm graph is already split into a connected component and many isolated users. President Obama may (as of this printing) have millions of “friends”, but he is unlikely to mediate file sharing requests for each of them. At the very least, the data indicates there may be more resilience in social graphs than previously thought. I caution that the results may not generalize beyond this data set.

Synthesizing these results, I observe that limited path redundancy is expected for those users with extremely low degree. But, for the set of nodes with even modest connectivity, redundant paths exist, even after targeted removal of high degree nodes. From the perspective of building OneSwarm, these results call for an adaptive design. High load on core nodes should be detected and alternate paths used. But, in circumstances where such paths are the only option, resource sharing must be effective.

3.1.4 Path properties

The average shortest path between users in the last.fm social graph is 7.1, and the diameter is 14.[3] Paths between last.fm users are longer than those reported of other social networks, e.g., Mislove et al. report average path lengths between 4 and 6 for popular social networks [62]. I attribute this difference to the absence of very high degree nodes in the last.fm data set and to the relative prevalence of low degree nodes; both factors increase path length.

Longer path lengths present a challenge for multi-hop overlay forwarding; any single path is likely to contain some node with limited capacity, and each path is only as fast as its slowest link. However, I lack the ability to measure the bandwidth of each last.fm user. Instead, I synthesize this data by assigning each user in the last.fm social graph a bandwidth capacity drawn randomly from a bandwidth distribution of typical peer-to-peer users. The details of these bandwidth measurements are out of scope for this work. More information can be found in my report on the subject [44].

Figure 3.3 compares 50,000 randomly selected {source, receiver} pairs in terms of utilization of sender's capacity, for various transfer disciplines. This data shows the potential

[3]Because computing all shortest paths in such a large graph is not computationally feasible, these results are based on a sample of 50,000 randomly selected user pairs.

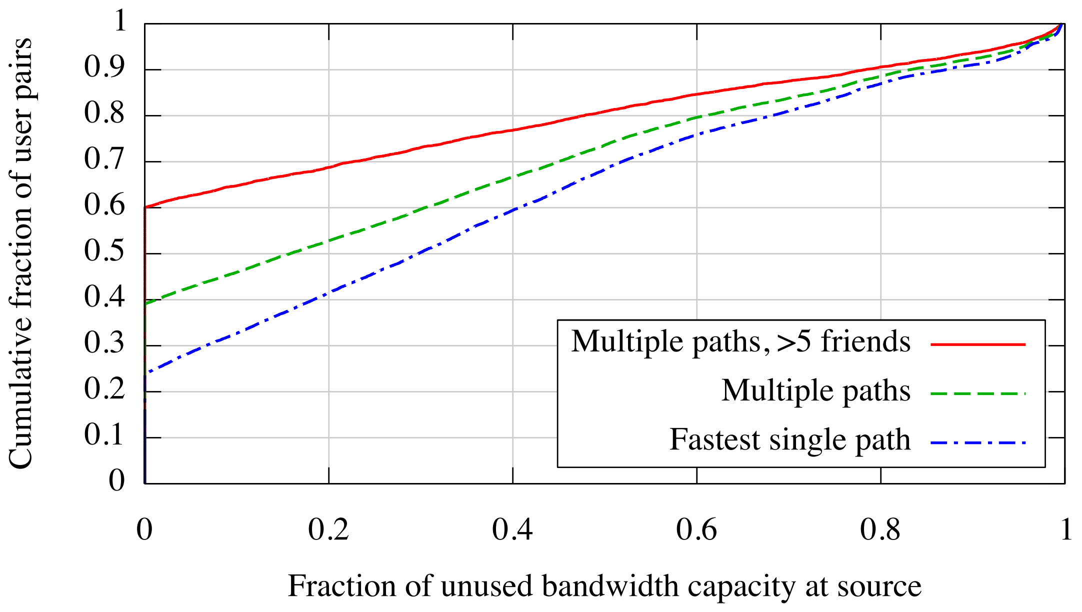

Figure 3.3: Unused client bandwidth for transfers involving either the fastest single path, multiple paths, or multiple paths for the subset of clients with more than five friends. Using the fastest single path, just 24% of user pairs saturated the sender's capacity. This increases to 39% when using multiple paths. For users with more than 5 friends 60% of senders are fully utilized.

for improvement from using multiple paths. Even assuming we could find the fastest single path, just 24% of user pairs saturated the sender's capacity. This increases to 39% when using multiple paths. With multiple paths, performance is limited by the large fraction (nearly 30%) of last.fm users with only a single friend. The most significant increase in performance comes from combining multiple paths *and* multiple friends. In this case, 60% of senders are fully utilized. To achieve full utilization, we need multiple paths, multiple friends, and multiple sources.

3.1.5 Listening habits

This section reports measurements of the listening behavior of last.fm users. I focus on the workload properties most relevant to the design of OneSwarm. These are: 1) the popularity of objects, 2) the variation in demand among users, and 3) the total and peak demand.

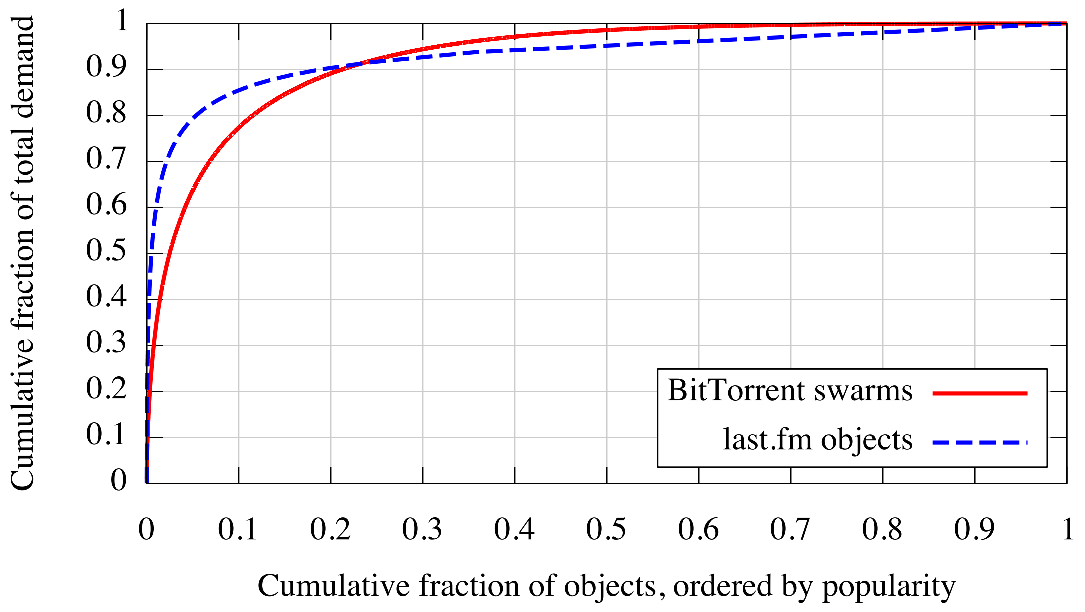

Figure 3.4: Cumulative distribution of object demand in the last.fm and BitTorrent workloads.

I discuss each of these in turn.

Object popularity

For file sharing systems layered on social networks, path lengths depend on both the connectivity of users and the object popularity. Even if paths between users are typically lengthy, paths to popular objects may be short because of replication. I first consider object popularity in terms of requests per object.

Although the majority of objects are unpopular, as expected, popular objects account for the majority of total demand. Figure 3.4 shows the cumulative fraction of total system demand attributed to objects ordered by decreasing popularity. I include an identical accounting of demand in the BitTorrent P2P file sharing system produced by Piatek et al. for comparison [72]. Demand is skewed in both BitTorrent and last.fm but the distributions differ. Unpopular objects contribute significantly more to total demand in last.fm than in BitTorrent. Songs listened to by three or fewer unique users account for 10% of total demand. Also, popular last.fm objects account for a larger fraction of total demand than do popular BitTorrent objects. The top 5% of objects account for 79% of total demand in last.fm and 63% in BitTorrent.

The comparatively large fraction of total demand attributable to unpopular objects may stem from last.fm's approach to data collection. Existing P2P workload measurements are influenced by the properties of the distribution system. For example, if accessing unpopular objects is slow or impossible in a particular P2P network, an object request trace is likely to underrepresent the true demand for those objects. Since last.fm simply records user behavior when interacting with their own libraries, it does not exhibit this bias.

Assuming that content in OneSwarm exhibit a similar popularity distribution to content requests in last.fm and BitTorrent this data has the following implications for the design. 1) The skew in object popularity means that many requests will be for popular objects with plentiful replicas; locating these will not require a thorough search of the entire overlay, presenting an opportunity for optimization to reduce overhead. 2) But, to locate less unpopular objects, OneSwarm should be able to conduct a thorough search if needed.

Demand per user

Figure 3.4 show demand from the perspective of objects. Next I turn to demand per user. For last.fm, demand per user is the distribution of songs played, shown in Figure 3.5. Demand varies by orders of magnitude; some user histories include 10s of songs while others include 1000s. This type of skew in demand is typical of object request workloads. While one might expect heavy users of last.fm to also have many friends, the length of play history and the number of friends are only weakly correlated ($\rho = 0.14$). From the perspective of file sharing, this implies that a significant fraction of requests will come from users with only limited connectivity.

The measurements in Figure 3.5 describe only active users, i.e., those that listen to at least one song. Surprisingly, these users are in the minority; 52% of measured last.fm users

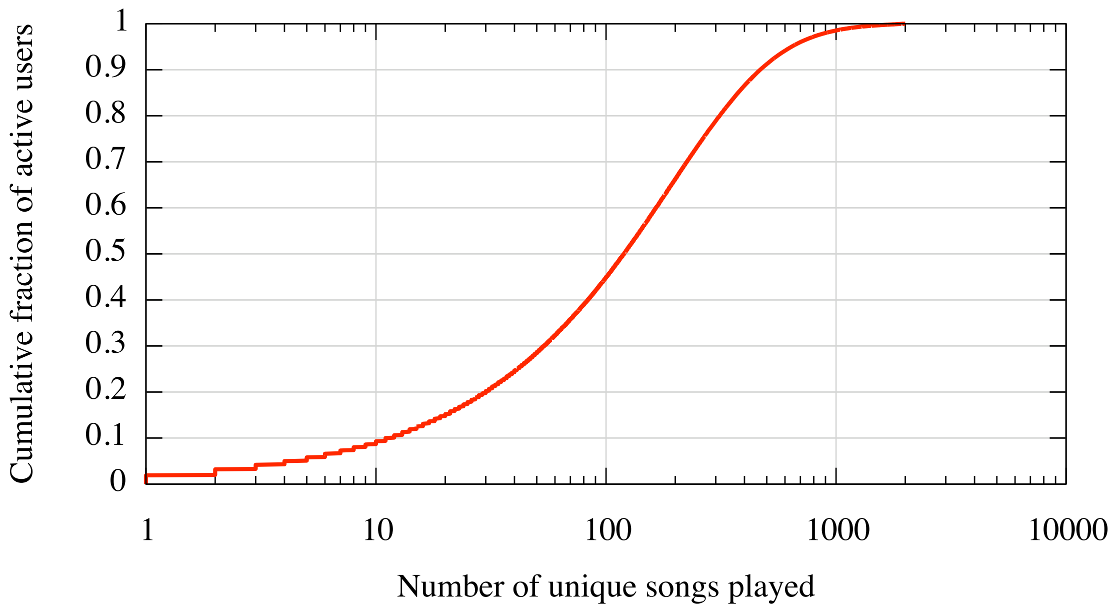

Figure 3.5: The cumulative fraction of users (y-axis) playing a given number of unique songs (x-axis) or fewer in our two week trace.

did not listen to any songs during our two week trace. If users only keep their client running while actively using the system protocol designers building on social networks should expect a large fraction of the social links to be unavailable even over lengthy time scales. Over shorter time scales, the last.fm usage exhibits a typical diurnal pattern with peak activity of 7.3% of users and a typical daily minimum of 2%, obtained using our fine-grained measurements of the listening behavior of 1,000 users.

Total demand

Over the two weeks of our activity trace, I observed 799,953 users that listened to at least one song with 156,295,286 total songs played. Of these, 15,120,192 were unique song requests per user. Multiplying this value by the average song length in bits (weighted by popularity) gives an estimate for the total demand. Assuming an audio bitrate of 128 Kbits/s, total demand for measured last.fm users over two weeks is 44.6 TB.

Our measurements suggest that, at least for a music sharing workload, multihop overlay forwarding is practical given current broadband capacities. Distributing 44.6 TB in two weeks requires just 4.2 MB of data per user per day. Even when forwarded over multiple hops, this meager amount of traffic is still well under the gigabytes of total capacity of even a modest 1 Mbit home broadband connection. Even when forwarding data over multiple hops the amount of bandwidth needed is significantly less than the bandwidth available, suggesting that a social network based overlay will be able to support this traffic pattern without problem. Further, because our trace accounts for only two weeks of usage, I overestimate the steady-state demand of the last.fm workload. The number of unique songs added by the second week of our trace was roughly half the unique songs discovered during the first week.

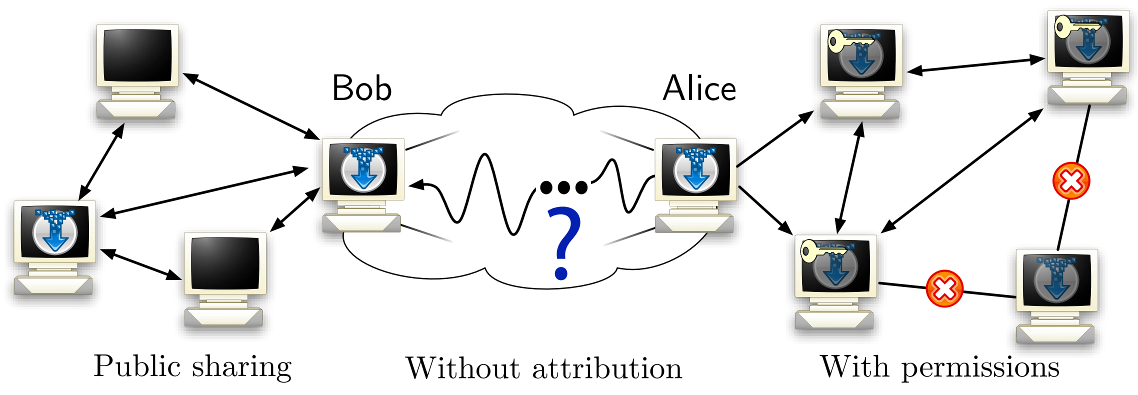

Figure 3.6: An example of the range of data sharing scenarios supported by OneSwarm. Bob downloads public data using OneSwarm's backwards compatibility with existing BitTorrent implementations, and makes the downloaded file available to other OneSwarm users. Alice downloads the file from Bob without attribution using OneSwarm's privacy-preserving overlay, but she is then free to advertise the data to friends. Advertisements include a cryptographic capability, which allows only permitted friends to observe the file at Alice.

3.2 Data sharing with OneSwarm

Figure 3.6 illustrates the range of privacy preserving options supported by OneSwarm. In this example, suppose users Alice and Bob both want to download a left-leaning political podcast. Suppose further that Bob does not consider his political views to be sensitive information, but Alice would prefer that her political views not be made public; instead, she might want to share the podcast with just a few like-minded friends.

OneSwarm supports all of these levels of privacy *within the context of a single swarm*. Bob downloads the podcast from a *public* set of existing BitTorrent and OneSwarm peers. During the download, Bob also acts as a replica for sharing *without attribution* using an overlay consisting of OneSwarm peers only. This overlay obscure the identities of a path's source and destination. Alice is one such destination, and she downloads the podcast using only anonymizing paths to preserve her privacy from third-party monitoring. But, she is free to advertise the file explicitly to friends who may also be interested in the content.

Each case shown in Figure 3.6 imposes a different tradeoff between privacy and efficiency. Publicly distributed data is not private, and direct transfers between a large set of replicas yield efficient distribution. Sharing data with permissions limits access and hence distribution capacity. Finally, data shared without attribution is accessible by anyone, but the set of users sharing the data is obscured, which increases overhead. To summarize:

- **Public distribution:** All data sharing need not be private. This is the case for which existing P2P systems excel, and OneSwarm draws on this strength by serving as a fully backwards compatible BitTorrent client. This helps bootstrap content into OneSwarm's privacy preserving overlay; data originally obtained using legacy protocols can be easily shared using any other mode. Sharing recorded course lecture videos is an example of this type of distribution.

- **With permissions:** Persistent identities allow OneSwarm users to define per-file permissions. In this case, *access* to files is restricted (rather than attribution of source or destination). In OneSwarm, capabilities restrict access to protected files, allowing all permitted users to recognize one another and engage in swarming downloads for scalability.[4] For example, OneSwarm can be used to restrict the distribution of a photo archive to friends and family only.

- **Without attribution:** When sharing sensitive data, privacy depends on obscuring *attribution* of source and/or destination. Unlike data shared with permissions, which is directly advertised, data shared without attribution is located using privacy-preserving keyword search, and data transfers are relayed through an unknown number of intermediaries to obscure source and destination. This type of distribution is appropriate for sensitive material. Since it is up to the user to define what is sensitive, the same data object

[4]Of course, capabilities (or data itself) can be relayed to others once obtained, but OneSwarm's default behavior is to maintain restrictions on data shared with permissions unless explicitly overridden.

3.2.1 Workload constraints

The workload measurements in the previous section provided information about the challenges that the protocol must handle. To summarize the findings:

- *Skewed object popularity motivates popularity-aware search:* The object popularity in last.fm is heavily skewed; the top 5% of objects account for 79% of total demand. Even so, rarely requested objects comprise a significant portion of the overall demand. To support this workload, the mechanism used for finding content must be able to efficiently find popular content while still being able to locate unpopular objects.

- *Long paths motivates multipath downloads from a single source:* In last.fm, the average shortest path between any two users is 7.1. In an overlay with similar structure, the diversity of end-host bandwidth capacities means that any single path is likely to be slow, limited by its lowest-capacity and/or most congested link. To provide good performance, OneSwarm uses multiple paths per-source to transfer data.

- *A resilient core improves availability but requires adaptation to congestion:* last.fm has significant path diversity and a very resilient core. But, the popularity of a minority of well-connected users suggests that as the amount of traffic in the network increases, OneSwarm must be able to find alternate routes to avoid congested nodes.

- *Bootstrapping is crucial since many users have few trusted links:* As with many social networks, popularity is highly skewed in last.fm and the majority of users have few social links. In an overlay, this would reduce both performance and privacy: downloads are efficient only when there are multiple path options, and privacy can likewise be more easily compromised for users with very limited fanout. For such users to benefit from OneSwarm, the design includes mechanisms for both trusted and untrusted overlay links.

These constraints shape OneSwarm's control and data transfer protocols as well as how users manage and define trust relationships.

3.3 Protocol design

In this section I describe the OneSwarm protocol. Table 3.1 provides a road map. I first provide an overview before describing each mechanism in detail; I defer a detailed security analysis to the next section.

3.3.1 Overview

Broadly, the protocol supports two tasks: 1) defining and maintaining the overlay topology and 2) locating and transferring data objects. A key design insight is that good P2P data sharing performance results from being able to optimize over multiple options for each data transfer. Thus I explicitly designed OneSwarm to make it easy for users to configure a rich peering topology and then to use that topology efficiently for each transfer.

Topology

OneSwarm users define overlay links by exchanging public keys, which identify nodes in the mesh and bootstrap authenticated and encrypted direct connections between peers in the underlying IP network. Thus, hassle-free key distribution is essential for usability, and OneSwarm uses social graph import and community server mechanisms to make key distribution straightforward for users. A distributed hash table (DHT) serves as a name resolution service; each client maintains encrypted entries advertising their IP address and port to authorized peers.

OneSwarm peers are either *trusted* or *untrusted*.[5] Trusted peers reflect real-world relationships, e.g., friends and family, and object permissions are defined in terms of access

[5] In practice, trust can be defined on both a per-object and per-peer basis. I discuss trust at the granularity of peers for simplicity.

	Mechanism	Description	Purpose	Section
Topology	Social import	Automatic key exchange via email invitations, LAN discovery, or existing social network services.	Bootstrapping trusted mesh links	3.3.2
	Community servers	A publish/subscribe coordination server for clients. Used to bootstrap new users with a random set of peers or to manage group membership for private communities.	Bootstrapping untrusted links, group management	3.3.3
	Distributed hash table	Encrypted key/value storage service maintained by clients to map identities to current IP addresses and ports.	Name resolution for mesh IDs	3.3.4
Data transport	Congestion-aware search	Controlled flooding of search queries to locate data and construct forwarding paths without overwhelming the network or exposing endpoints.	Locating objects, discovering paths, avoiding hotspots	3.3.5
	Swarming data transport	Data is split into blocks, with active downloaders redistributing completed blocks. Transfers use multiple paths and multiple sources, if available.	Load balancing, efficiency	3.3.6
	Long-term history	Each client maintains transfer volumes for each peer, using these to prioritize service during periods of congestion.	Resource allocation, contribution incentives	3.3.7

Table 3.1: A summary of protocol mechanisms used by OneSwarm.

control lists of trusted identities. Untrusted peers are used only for data sharing without attribution, serving to bootstrap mesh connectivity for users with few trusted friends.

Supporting a mix of trusted and potentially untrusted peers provides greater performance than using only trusted peers and enhances privacy relative to using only untrusted peers. Moreover, my experience has shown it to be a practical necessity for user adoption. The initial implementation assumed mutual pairwise trust among directly connected peers in order to simplify the protocol and security analysis. But, this restriction was widely criticized (or ignored) by many early adopters, leading to a design supporting variable trust in peers. Untrusted peers are treated differently by the protocol; the timing and delivery of messages are randomized to frustrate statistical attacks.

In Section 3.3.5, I outline the random perturbations of the timing and delivery of protocol messages needed to support untrusted peers, delaying a more complete discussion of attacks and defenses until Section 3.4 to first provide a complete protocol description.

Transport

The mesh defined by the web of trust among users is used to locate and transfer data. The overall approach is inspired by the success of existing P2P swarming systems, e.g., BitTorrent, and I adopt existing swarming techniques wherever possible, with three adaptations to enhance privacy. First, instead of sharing all data publicly with distinct and dynamic sets of peers, each OneSwarm client restricts direct communication to a small number of persistent contacts, which provide indirect connectivity to the rest of the mesh. Second, instead of centralizing information about which peers have which data objects, e.g., at a tracker as in BitTorrent, OneSwarm peers locate distant data sources by flooding object lookups through the overlay. Third, instead of sources sending data directly to receivers, data transfers occur over the reverse search path in the mesh, obscuring the identities of sender and receiver when sharing data without attribution.

Flooding lookup and indirect transfers increase the overhead of OneSwarm relative to existing protocols such as Freenet, potentially creating capacity constraints and/or bottlenecks. To cope with this, OneSwarm's search and data forwarding protocols are congestion-aware, automatically routing around overloaded intermediaries and allowing such nodes to shed load at will. To provide high performance in the face of overloaded or slow paths, OneSwarm transfers use multiple paths to each data source. To incentivize users to contribute capacity, each OneSwarm client maintains a history of traffic volumes provided by its peers, using this information to prioritize service during periods of congestion.

3.3.2 Linking trusted peers

Each OneSwarm user generates a 1024 bit RSA public/private key pair when installing the client, with the public key serving as its identity among its peers. OneSwarm identities are persistent, allowing two users that have exchanged keys to locate and connect to one another whenever both are online, even though their IP addresses might change. In existing social-sharing P2P designs [20, 73], key exchange is typically manual. I view manual exchange as a hindrance to adoption and include multiple methods for users to more easily distribute keys.

Between two OneSwarm users that share a real-world trust relationship, OneSwarm automates key exchange in three ways. First, as in UIA [34], the OneSwarm client discovers and exchanges keys with other OneSwarm users over the local area network. Second, I piggy-back on existing social networks, e.g., Google Talk, to distribute public keys automatically among friends. Third, users can email invitations. Invitations include a one-time use capability that authenticates the recipient during an initial connection, during which public key exchange occurs.

For all methods described above, users can choose whether to accept new overlay links. This allows users to maintain separate lists of OneSwarm contacts and contacts from other social services, while still avoiding the inconvenience of manually exchanging keys with friends out-of-band.

XMPP

XMPP is an open instant messaging standard and is the underlying protocol used for the Google Talk and LiveJournal instant message services. Each user adds friends to a list of contacts to exchange messages, transfer files, and receive status updates. The set of contact lists forms a social network among users. To piggyback on this network, OneSwarm speaks the messaging protocol and notifies a bot running at UW that the user's XMPP identity corresponds to a given OneSwarm identity. During this exchange, the bot notifies the user if any of their XMPP friends are also OneSwarm users. This is a two step process. First, each client send the bot its public key, a nickname, and a list of the SHA-1 hash of each of its XMPP friend IDs. Next, the bot checks an internal database to determine whether another OneSwarm user has registered a key corresponding to any ID hash in the list. If so, the bot verifies that the relationship is symmetric; i.e., both have registered the hashes of each other, and then returns the nickname and public key of the remote peer to the client registering its friend list. During the remote peer's next update, the situation is reversed, and both peers will have learned the each other's keys.

3.3.3 Managing groups and untrusted peers

Exchanging keys manually allows for fine-grained control, but in many circumstances explicitly authorizing every peer relationship is cumbersome and unnecessary. Further, OneSwarm is frequently used by communities of users with dynamic membership but mutual pairwise trust, e.g., a group of colleagues at the same institution. In such cases, users can benefit from an automated service that provides subscription to keys.

To support key management within a group, OneSwarm allows users to subscribe to one or more *community servers*. A community server maintains a list of registered users and provides authorized subscribers with a current set of public keys via a secure web connection. In effect, subscribers to a given community server delegate trust regarding a subset of their peers to the operator, who vets prospective members. These *private* community servers mediate key exchange among users with existing trust relationships.

In contrast with private community servers, *public* community servers have open membership, allowing new OneSwarm users to easily obtain a set of untrusted peers. Bootstrapping early adopters is a significant challenge for overlay networks based on pairwise trust. But, in the case of sharing without attribution, trusted peers are not required; privacy depends on the obfuscation provided by forwarding data through multiple unknown intermediaries. Untrusted peers are used only for this type of sharing and serve to bootstrap overlay connectivity for users with few trusted friends.

Registration itself is a three step process. First, the OneSwarm client provides its public key, which the server then verifies by issuing a challenge nonce value and verifying the incremented, encrypted response. Finally, the server uses consistent hashing of the public key to compute a subset of peers to return to the client.

Community server registration is designed to inhibit systematic crawling of the membership list of a public community server. Verifying keys with a challenge/response allows the server to limit the number of registrations by a single IP address, consistent hashing limits the information obtained from repeated membership queries, and each connection is established only when both nodes have obtained the identity and the location of the other node from the community server.[6]

Although an attacker with significant resources can evade these restrictions by creating many Sybil identities from distinct IPs, doing so is of limited value. The overlay topology is an amalgam of links from community servers, manual exchanges, email invitations, and other social networks; a crawl of community servers provides only a partial view, and more privacy conscious users need not subscribe to any community server whatsoever. I consider the effectiveness of attacks enabled by public community servers in more detail in Section 3.4.

[6]An alternate approach would be to obtain a random set of peers from a DHT, but a significant limitation is that current DHTs are not robust to Sybil creation from a single IP.

3.3.4 Identity and connectivity

To be able to locate and identify peers with changing network address OneSwarm creates a a long-term identity when first started. Long-term identities are linked to transient IP addresses and port numbers via a distributed hash table (DHT) maintained among all users. On startup, each client P inserts a copy of its current IP address and port into the DHT. This value is inserted multiple times—once for each peer.

DHT entries for a client P are encrypted with the public key of a given peer and signed by P. Each entry is indexed by a 20 byte randomly generated shared secret, which is agreed upon during the first successful connection between two peers.

Prior to the initial connection with a newly added friend, P temporarily advertises connectivity information at a special location: the SHA-1 hash of the concatenation of P's public key and the public key of the given friend. This location serves as the initial rendezvous point.

Inserting connectivity information individually for each peer enables fine-grained control over network address information. A simple alternative is indexing connectivity information by the public key of P alone. But, in that case, any user that learned P's public key could monitor P's IP location as long as P maintained its identity. By encrypting updates and publishing connectivity information for each peer individually, P can control and revoke each peer's access to its IP location updates.

In my implementation, ID $\rightarrow$ {IP, Port} mappings are stored in a Kademlia-based DHT using twenty-fold replication for fault tolerance [60]. This level of replication has been shown to provide high availability for DHTs running on end-hosts [32]. Each client's location in the DHT is independent of its identity and is determined by hashing the client's current IP address and DHT port.

3.3.5 Naming and locating data

OneSwarm peers connect to one another using secure sockets (SSLv3) bootstrapped by their RSA key pairs. When two peers connect, they exchange file list messages. file list messages are compressed XML including attributes describing the name, size, and other metadata for files for which a particular peer has permissions. (The node sends an empty list to each untrusted peer, or if it has nothing to share with a specific peer.)

Naming

Shared files (or groups of files) are named in OneSwarm using the 160 bit SHA-1 hash of their name and content. The low order 64 bits of this hash are used as an ID in search messages; these messages are flooded to discover potential data sources. For public data, users obtain content hashes 1) out-of-band, e.g., from an email or website, 2) from file list messages exchanged with peers, or 3) from keyword search in the overlay. I describe transfer negotiation via search since this subsumes the other cases.

Congestion aware search

OneSwarm search is designed to manage the tradeoff between overhead and performance by being *congestion aware*. Using the shortest path minimizes overhead, but risks poor performance if the shortest path is slow or overloaded. Given that highly connected users are more likely to appear in a path, this is a practical concern.

OneSwarm addresses this by managing the propagation of searches. Because the path taken by a search message determines the path of data transfer, the key idea is to forward searches along the shortest path possible (to limit overhead) subject to each intermediary's current load (to improve performance).

To discover shortest paths, OneSwarm relies on flooding. Keyword search messages include a randomly generated search ID and list of keywords. Unlike flooding search in other P2P file sharing networks, OneSwarm search messages do not include a time-to-live value since this information would allow intermediaries nearby the source or destination to easily reason about behavior. Instead, OneSwarm forwards searches to trusted peers provided the forwarder has idle capacity and the search has not been forwarded previously. By maintaining a set of rotating Bloom filters, an hour of search history can be remembered space-efficiently, ruling out the possibility of searches living forever in the overlay.

Among untrusted peers, forwarding is randomized to prevent collusion attacks. Instead of forwarding unmatched search messages to all peers, OneSwarm forwards searches to untrusted peers probabilistically. This inhibits colluding untrusted peers from inferring a data source by observing the lack of a forwarded search message. To prevent information leakage through repeated queries, the decision to forward a search is made randomly —but deterministically— so repeated queries for the same data will yield the same result. I explore the privacy implications of this in Section 3.4.5.

To avoid the propagation of every search to every client in the overlay, each client delays each search message for at least 150 milliseconds before forwarding it to peers. The search source (or any forwarder) may terminate the search, once enough data sources have been discovered, by sending a search cancel message to nodes to which they have sent or forwarded a search message. The search cancel message is forwarded along the same paths as the corresponding search message but without any forwarding delay, allowing cancel messages to quickly reach the search frontier. Previous studies have shown that most searches in P2P network are for files with many replicas [17] and as a result these popular searches will be canceled quickly, reducing overhead.[7]

In addition to the fixed forwarding delay for search cancellation, OneSwarm also delays messages based on the load at each intermediary. Where load is high, searches get delayed due to queueing. This causes searches to propagate over alternative paths, improving performance. When excess capacity exists, search messages will follow the shortest path, reducing transfer overhead.

[7]Search overhead could also be reduced by routing queries through super-nodes or performing random walks [17], but such optimizations either impact privacy or result in transfers over long paths.

This design trades global reachability for lower overhead. Specifically the design does not guarantee that each object can be found by all nodes at all times. When nodes experience congestions they probabilistically drop search messages to ensure that available bandwidth is used for data forwarding. Searches will flow over alternative, non congested, paths if they exists. If no alternate paths are available the search will fail. An evaluation of the efficiency and effectiveness of the search algorithm is provided in Section 3.5.2.

Path setup

If a node has the file that matches a search query, it does not forward the search. Instead, it responds with a search reply message. Among trusted peers, this response is immediate. But, receiving a search reply message in less than 150 ms (the default per-hop forwarding delay) would reveal the responder as a data source to potentially untrusted peers. To prevent this, the OneSwarm software delays search reply messages (and all protocol messages) sent to untrusted peers in order to emulate the delay of a longer path. This value is chosen randomly between 150-300 ms (i.e., 1–2 hops). As with forwarding of search messages, the delay value is persistent for a particular file and a particular peer to prevent information leakage from repeated queries.

Search reply messages include the search identifier, a list of content hashes for matching files, file metadata, and a *path identifier*. The path identifier allows clients to distinguish among multiple paths even if those paths partially overlap. I first describe how path IDs are computed and then how they are used to enable multi-path and multi-source downloading. Each peer maintains a randomly chosen *link ID* for each peer link and keeps this information private.[8] The data source sets the initial value of the path ID to the lower 32 bits of the first matching file's hash. Next, the search reply is sent (to each peer who forwarded the data request) with the SHA-1 hash of the initial value XOR'd with the link ID of the given peer. This process of updating the path ID is repeated at each overlay hop,

[8]Though randomly chosen, this value is fixed for the lifetime of the connection.

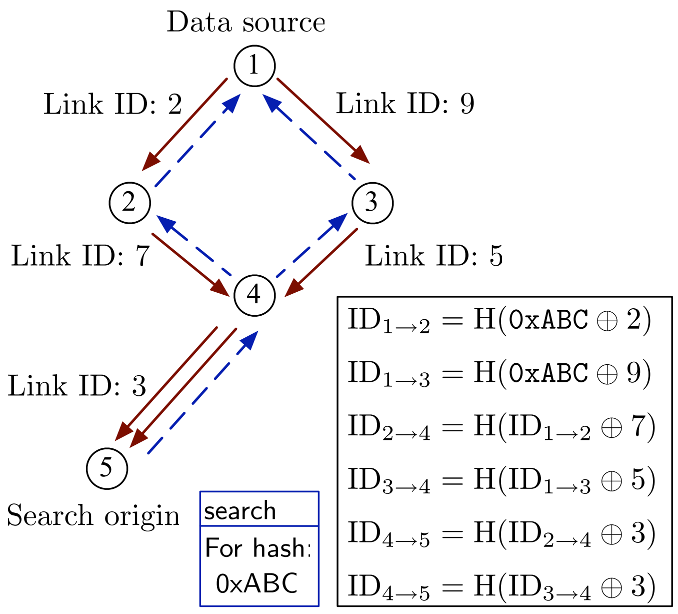
Data source
Link ID: 2
Link ID: 9
Link ID: 7
Link ID: 5
Link ID: 3
Search origin
search
For hash:
0xABC
$ID_{1\to2} = H(\texttt{0xABC} \oplus 2)$
$ID_{1\to3} = H(\texttt{0xABC} \oplus 9)$
$ID_{2\to4} = H(ID_{1\to2} \oplus 7)$
$ID_{3\to4} = H(ID_{1\to3} \oplus 5)$
$ID_{4\to5} = H(ID_{2\to4} \oplus 3)$
$ID_{4\to5} = H(ID_{3\to4} \oplus 3)$

Figure 3.7: An example of end-to-end path ID computation. Client 5 searches for peers with file ID `0xABC` and queries are forwarded along the dashed links. In this case 2 unique paths are found.

resulting in a unique ID for each path back to the sender. A simple example of path ID computation is shown in Figure 3.7. The ability to recognize unique paths allows the receiver to add new paths during the course of a download. Transfers can start as soon as one path is discovered. New searches can be launched to replace paths that fail, while still allowing the search originator to eliminate duplicate paths.

3.3.6 *Swarming data transfer*

A path identifier indexes routing tables at each overlay hop and effectively identifies a circuit from data source to receiver. Keep-alive messages refresh paths, which expire after thirty seconds of inactivity. OneSwarm uses the wire-level protocol from BitTorrent to transfer data [22]. But, rather than connecting directly to peers, OneSwarm tunnels BitTorrent traffic through overlay paths. Each overlay path is treated as a virtual BitTorrent peer, even those that terminate at the same endpoint. Of course, the receiver has no definitive way to know which paths terminate where. Rather than obtaining a list of peers from a centralized tracker, as in BitTorrent, OneSwarm discovers new paths by periodically flooding search messages for active downloads.

Basing OneSwarm's wire-level protocol on BitTorrent draws on BitTorrent's strengths. Swarming file downloads minimize redundant data transfers in the overlay. If multiple users are downloading a popular file, OneSwarm will discover and use paths to those new partial sources.

Like the unpredictable and heterogeneous end-hosts BitTorrent is designed for, multi-hop overlay paths have highly variable bandwidth and end-to-end latency. Scheduling block requests over unpredictable paths requires careful engineering to avoid wasting capacity or inducing lengthy data queues. I take advantage of parts of the BitTorrent protocol that allow multiple requests to be queued at the data source, effectively giving the host control over the end-to-end transfer window size. For example, if a path becomes congested, traffic will automatically be shifted to paths that do not traverse the congested link. If a forwarding node disconnects, the capacity of the data source is automatically shifted to other paths. As in BitTorrent, content integrity is protected by SHA-1 hashes of file blocks, allowing recipients to detect data corruption.

3.3.7 Incentives

Persistent identities and long-term relationships provide a rich foundation on which to implement different incentive strategies. Each OneSwarm client maintains transfer statistics for each peer including total data uploaded and downloaded, maximum transfer rates, control traffic volume, and uptime.

OneSwarm retains BitTorrent's default tit-for-tat policy for making servicing decisions among multiple virtual BitTorrent peers. This creates an incentive to contribute capacity while downloading, improving swarm performance. Persistent identities among directly connected peers provide an incentive to continue sharing data after downloads complete. During periods of contention, the default policy is to allocate bandwidth among directly connected peers proportionally; each peer is assigned a weight equal to the ratio of their net contribution and net consumption. A client improves its standing over time by participating in the system whenever possible.

Forwarding data is zero sum. Data consumption from the incoming peer connection is matched by contribution at the outgoing connection. At the granularity of individual paths, it is difficult to reason about whether a particular forwarding connection is helpful for a peer's long-term interests. If the outgoing peer often is on the path of a client's own transfers, forwarding will improve the transfer balance with that peer leading to increased priority for subsequent downloads. But, if the incoming peer is a more useful data source, forwarding will reduce long-term performance. To cope with this, OneSwarm uses a default forwarding policy inspired by peering relationships between ISPs. If the incoming/outgoing traffic ratio of a peer is approximately balanced or greater than 1 over the long-term, forwarding is permitted. But, if this ratio is significantly unbalanced, forwarding is not permitted during periods of contention. This default policy can be overridden. Users are free to assign static weights per-peer or forward data without regard to traffic imbalance.

In practice, the default policy has proven sufficient to induce a surplus of forwarding capacity in the system. I verify this in the evaluation (Section 3.5).

3.4 Security analysis

OneSwarm's overarching security goal is to improve privacy by allowing users to control information disclosure. When sharing data with permissions, disclosure is limited by familiar mechanisms: strong identities, capabilities, and end-to-end encryption. In this section, we focus on analyzing privacy properties in the more challenging case of data sharing without attribution.

3.4.1 Threat model

Our goal is to be resistant to the disclosure of user behavior to an attacker with control over a limited number of overlay nodes. Native BitTorrent is susceptible to just this attack, enabling a small number of monitoring agents to infer the behavior of tens of millions of users [86, 70]. Specifically, we assume that an attacker can join the network with a limited number of nodes, monitor network traffic to/from its nodes, and generate, modify, and delete OneSwarm overlay messages flowing through its nodes. The attacker can record timing information about the messages it sends/receives to infer information about the behavior of the rest of the OneSwarm network, and the attacker may spawn any number of OneSwarm instances on its nodes. We do not attempt to guarantee privacy against attackers that can sniff, modify, or inject traffic on arbitrary network links or attackers that can seize the physical hardware of OneSwarm users, e.g., law enforcement.

OneSwarm assumes that users are conservative when specifying trust in peers, as trusted peers can view files for which they have permissions. If trust is misplaced or a peer compromised, OneSwarm limits the resulting disclosure to only the trusted peers of the compromised nodes. This is in sharp contrast to private BitTorrent communities [107], where a single compromised member can monitor all users of the service.

3.4.2 Attacks and defenses

In this section, we outline several potential attacks and quantify their effectiveness using measurements of OneSwarm users in the wild. I restrict my attention to what I believe to be the most likely attackers conducting the most likely attacks: one or more colluding OneSwarm users bootstrapped via public community servers attempting to infer the source of a data transfer. The discussion highlights the following aspects of the OneSwarm protocol that significantly enhance user privacy.

- *Persistent peering relationships limit monitoring power:* In BitTorrent, peers are dynamically assigned, allowing attackers to become a peer of virtually everyone, given enough time. By contrast, OneSwarm peers are persistent, improving contribution incentives and also limiting the ability of attackers to snoop from arbitrary locations in the overlay.

- *Heterogeneity of trust relationships foils timing attacks:* OneSwarm users define links as either trusted or untrusted and keep this information private. As the protocol behavior varies with link type, the combined use of trusted and untrusted links greatly diminishes an attacker's ability to infer path properties based on timing information.

- *Lack of source routing limits correlation attacks:* OneSwarm does not provide peers with the ability to construct arbitrary overlay paths. Attackers could use this to correlate performance with ongoing transfers. Such an attack is known to degrade privacy in Tor, for example [108]. Individual clients have a limited view of the overlay and cannot control path setup beyond directly connected neighbors.

- *Constrained randomness frustrates statistical attacks:* The uncertainty arising from random perturbations in the protocol could be reduced through statistical analysis if repeated probes yielded different draws. OneSwarm prevents such analysis by making all random decisions deterministically with respect to a given query and link.

- *Network dynamics limit value of historical data:* While relationships in OneSwarm are long lived, the end-to-end paths between senders and receivers change rapidly due to churn and transient congestion. This reduces the window of opportunity for adversaries to combine data from multiple observations in order to reverse-engineer user behavior.

3.4.3 Timing attacks

Because data is forwarded through multiple intermediaries, an attacker seeking to determine whether one of its peers is the ultimate data source depends on determining whether the peer is relaying the data or not. For now, we make the optimistic assumption (from an attacker's perspective) that the attacker is directly connected to a data source and merely needs to confirm this. In the absence of a direct peer relationship, attackers are unable to learn the IP address of other clients, our notion of identifying a user.

By measuring the round trip time (RTT) of search / response pairs, an attacker can estimate the proximity of a data source. Usually, paths are lengthy, making the chances of being next to a particular data source quite low. If the attacker has sufficient resources to connect to nodes at many different points in the mesh, however, some of them might be able to infer that they are near to or directly connected to a data source based on the low RTT of response messages.

To frustrate this attack, OneSwarm artificially inflates delays for queries received from untrusted peers. Recall that attackers bootstrapped via community servers are marked as untrusted by default. All responses to untrusted peers are delayed by a random but deterministic amount (computed based on the content hash and a persistent local salt value) in order to emulate the delay profile of forwarded traffic from one or more hops away.

The RTT observed by an attacker over an untrusted link is similar to that of a data source that is one or two overlay hops away and connected via low latency, trusted forwarding links. In other words, the combined use of trusted and untrusted links provides more possible explanations for a given delay profile than a design using untrusted links only.

I now consider two experiments that illustrate the uncertainty associated with inferring data proximity based on timing information. First, we measure the variability of latency and path properties in practice using our PlanetLab deployment. Next, I consider the effectiveness of this attack for the last.fm topology and workload.

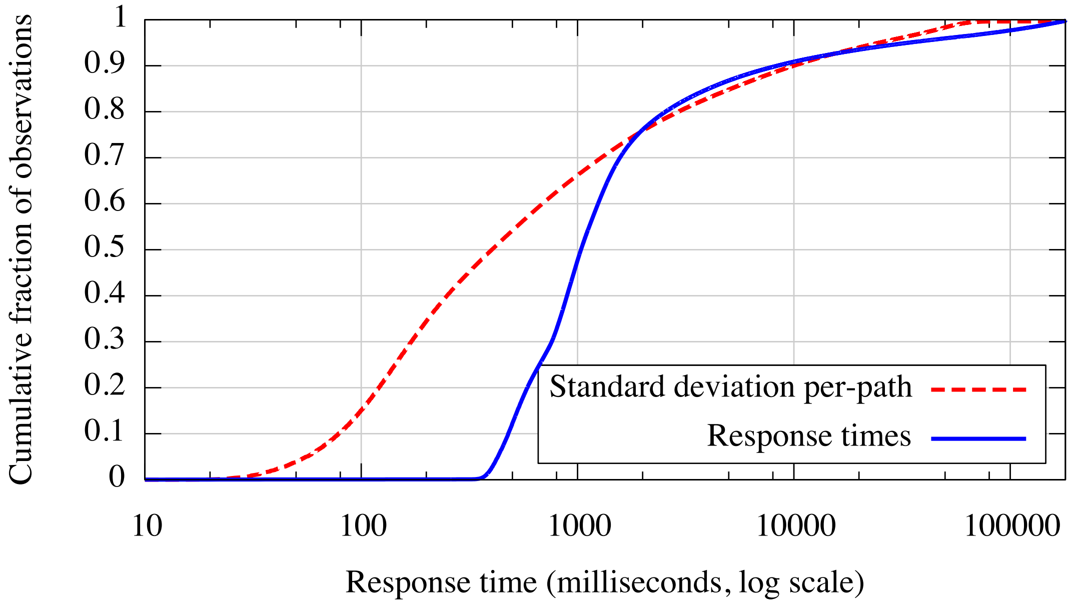

Figure 3.8: The distribution of search / response RTTs and the distribution of variance for RTTs on identical overlay paths with more than 10 search responses. Even for identical paths, variation in delay is significant.

PlanetLab

The feasibility of inferring behavior based on message timings depends on the length, stability, and diversity of paths to the object. Lengthy paths have greater variability due to mesh dynamics and network level effects. Similarly, the existence of a large, dynamic replica set and/or many paths confounds inference based on search response RTTs.

To evaluate this, I configured a set of PlanetLab nodes running instrumented OneSwarm clients to measure the RTT of search / response messages. As with would-be attackers, these nodes are bootstrapped via public community servers. Each node monitors all search requests it forwards, recording the RTTs of search response messages. For a given search, the peer responding with the smallest RTT across all measurement nodes is the likely closest hop to the data source. The stability of first responders for back-to-back search requests indicate the feasability of this attack; i.e., is the first responder for a given search the same as the first responder for the next search? With ten vantage points, 65% of back-to-back searches have the same first responder. Increasing the number of vantage points to 100 *reduces* back-to-back consistency to 63% as multiple attacker nodes at similar distance to the source get responses within a small enough interval that traffic conditions on the paths determines which attacker to first see the response. On the whole, it is difficult to reason about the true source of search response messages since the ordering of responses is highly variable.

The unpredictable ordering of search response messages is attributable to the naturally large variations in message delays. Figure 3.8 summarizes the distribution of response RTTs for more than 42 million searches, collected prior to the public release of a OneSwarm client incorporating artificial delays. Large RTTs suggest lengthy paths; the majority of search response messages are observed more than one second after forwarding their corresponding search. Even so, a variety of confounding factors make reasoning about path length on the basis of delay difficult. OneSwarm is willing to tolerate lengthy queueing delays at congested nodes (up to 7 seconds in our current implementation). Since search response messages are interleaved with data traffic, response times may be controlled by either 1) network delay, 2) lengthy overlay queueing delay at congested intermediaries, or 3) the protocol-imposed propagation delay of search messages. These effects manifest in significant variations in RTTs for even identical paths (i.e., responses carrying the same path ID).

This data was collected before the inclusion of randomized search response delays in the publicly available client. Thus the current implementation is likely to exhibit even greater variability.

Trace replay of last.fm

To complement the PlanetLab study, I used trace data from the last.fm music website to drive a large-scale simulation. A crawl of music playback histories and social relationships yields a trace of the user behavior for 1.7 million users. I interpret last.fm friend links as

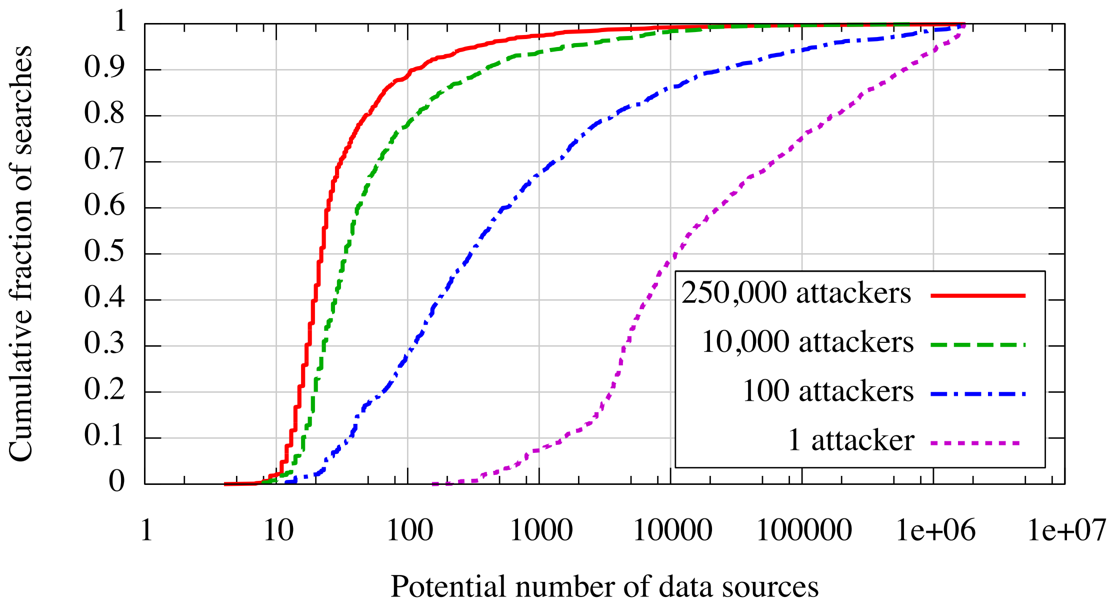

Figure 3.9: Using a latency and topology oracle, the number of potential data sources (x-axis) for a cumulative fraction of searches by attackers (y-axis). Even with thousands of attackers and complete topology/latency information, search response delays do not localize data sources.

trusted links in the overlay topology. For users with fewer than 26 friends, I add additional untrusted links until that users reaches a total of 26 links. Object download histories determine object placement and popularity. As an estimate for link latencies I randomly assign values provided by the iPlane project [58].

I use this trace to evaluate the timing attacks in the idealized setting of an unloaded network and attackers with complete information regarding the overlay topology and the network delay of every link. For varying numbers of attackers connected trough untrusted links, I simulate 1,000 searches in the last.fm topology, sampled to match the measured popularity of objects. For each search, we record the delay of the first response, and then inspect the topology and link delays to compute the number of possible data sources associated with a given delay and vantage point. Figure 3.9 summarizes the results. Even with complete topology and latency information as well as 250,000 vantage points, search response latencies never localize a single data source, and only rarely localize to fewer than

10 data sources.

3.4.4 *TCP based attacks*

Work by Prusty et al. [74] has suggested a TCP based attack that can be used to determine if a neighboring OneSwarm peer is the source of a particular data object. The attack works as follows: the attacker requests a object over the OneSwarm overlay. As the transfer starts the attacker leverage optimistic acking [82] to trick the sender into sending at a rate that is much higher than the senders link capacity, and thus much higher than the sender possibly could forward traffic. If the attacker seems high packet loss it indicates that the peer is the source as the source can read data from local disk at a high rate. If packets arrive at the same rate regardless of acking policy, the peer is forwarding data from another peer and can only forward data is it arrives from the previous hop.

OneSwarm is not susceptible to this attack when used with default settings due to application level rate limiting. At first launch OneSwarm initiates a bandwidth capacity measurement. The rate limit is then set to 80% of the measured capacity. Because the application delivers packets to the kernel at this maximum rate, using optimistic acking has no effect. Users changing their OneSwarm configuration to allow unlimited an upload rate are vulnerable to this attack.

An alternative defense against this attack is to limit upload rate to the median download rate for each peer, another is to switch to using a datagram based transport layer. What Prusty et al. highlighted is that bandwidth, just like latency, leaks information, so to foil this type of attack the software needs to control bandwidth as well as latency.

3.4.5 *Collusion attack*

Next, I analyze the case of multiple peers colluding to infer whether a directly connected user is sharing a particular file. In this case, an attacker A sends a targeted search to target T, receives a search response, and observes whether the search was forwarded to colluders

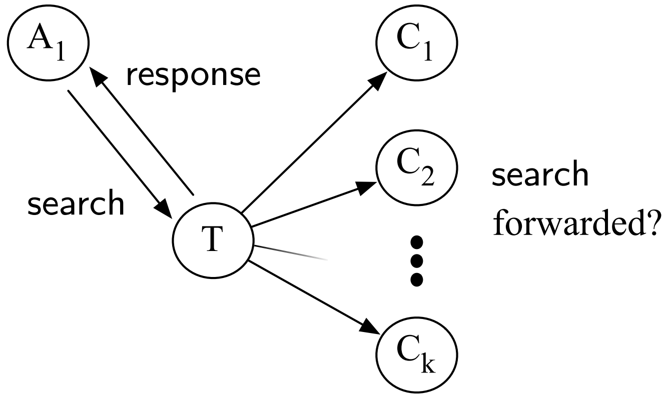

Figure 3.10: An attacker, A, with $C_1, ..., C_k$ colluders tests if a target T is sharing a file by sending a targeted search and observing a lack of forwarding.

$C_1, ..., C_k$ who are also peers of T. (This attack is illustrated in Figure 3.10.) Recall that forwarding search messages is probabilistic. Each search message has a configurable probability, p_f, of being forwarded to a particular peer. As a result, a lack of forwarding does not definitively identify a data source; missing search messages may arise from random chance. But, a lack of forwarding observed by many colluding peers is highly suggestive of T sourcing the object. Assuming a fixed forwarding probability of p_f and k colluding attackers, Pr[Not source|response received] $= (1 - p_f)^k$. With just a few colluders, an attacker can gain high confidence.

This attack requires both the attacker and colluders to be directly connected to the target. When matched randomly by a trustworthy public community server, the likelihood of an individual attacker being assigned a specific target for a community server with N members is $\frac{n_c}{N}$, where n_c is the number of peers returned for a single request. As a specific example, consider achieving greater than 95% confidence in the identification of a data source given $p_f = 0.5$ for peers received from a community server.[9] Achieving 95% confidence in identification requires at least six directly connected peers (an attacker and five colluders). For a community server with N users, the likelihood of achieving a particular number of direct connections is given by the complement of a binomial CDF with success

[9]Low values of p_f for community server peers are offset by the high amount of path diversity among them.

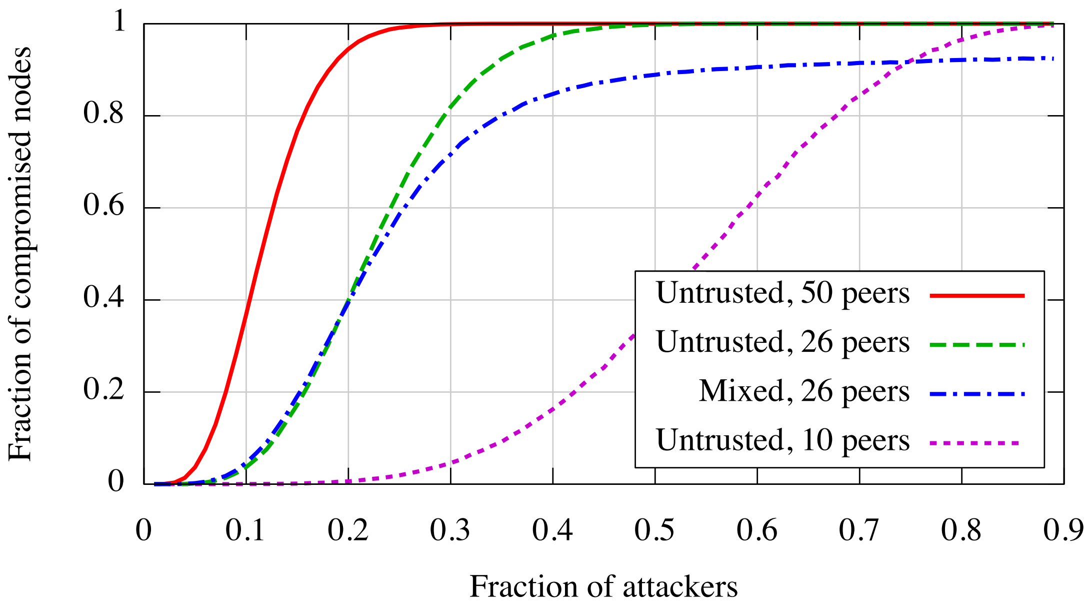

Figure 3.11: The cumulative fraction of nodes whose behavior can be inferred with 95% confidence (x-axis) by a given fraction of colluding attackers (y-axis). Even assuming widespread use of public community servers, a significant fraction of colluding attackers is required to infer user behavior.

probability $\frac{n_c}{N}$.

In practice, the effectiveness of systematic monitoring depends on the resources of an attacker relative to the population of a public community server. Privacy depends on this ratio being small, and privacy-conscious users are free to decrease their forwarding probability (p_f), avoid public community servers completely and rely on trusted peers, or request fewer peers than n_c. Queries to trusted peers are always forwarded. Figure 3.11 provides several concrete examples of the relationship between exposure, forwarding probability, topology, and the number of untrusted peers. In these examples, $p_f = 0.5$, and I vary n_c. Decreasing the maximum number of peers provided by a community server makes compromising its users more difficult. But, I find in my evaluation that increasing peers improves performance (Section 3.5).

Figure 3.11 also shows the privacy benefits associated with a mix of trusted and untrusted peers. For this case (Untrusted, 26 peers), I considered the vulnerability of clients in our last.fm trace when adopting a policy of peering with untrusted clients only when they did not have n_c or more contacts from their social network. Users with a large number of trusted friends are completely isolated from colluding attackers, shifting risk to users with few/no friends that are forced to more heavily rely on untrusted peers.

3.5 Evaluation

To evaluate OneSwarm, I measure its performance and robustness both in the wild and synthetically using trace replay. OneSwarm has been downloaded hundreds of thousands of times to date, and I use a combination of both voluntarily reported user data as well as instrumented clients to quantify OneSwarm's real-world effectiveness at the scale of thousands of users. To examine OneSwarm's operation at even larger scale, I replay traces of the social graph and usage behavior of more than one million last.fm users. In both cases, the main result is that OneSwarm provides high throughput and availability in spite of the overhead arising from preserving privacy. In support of this conclusion, I also measure the effectiveness of OneSwarm's protocol mechanisms and report usage and workload statistics.

3.5.1 Real-world deployment

Methodology

Although many aspects of user behavior are (deliberately) obscured by designing for privacy, I draw on two sources of data to profile OneSwarm's structure, performance, and utilization in the wild. The first of these is voluntarily reported summary statistics from more than 100,000 distinct users collected from a ten month period in 2009. These include the total number of peers, the method used for key exchange, and aggregate data transfer volumes.

The second source of data is instrumented OneSwarm clients running on hundreds of PlanetLab [68] machines. Subscribing to several public community servers bootstraps

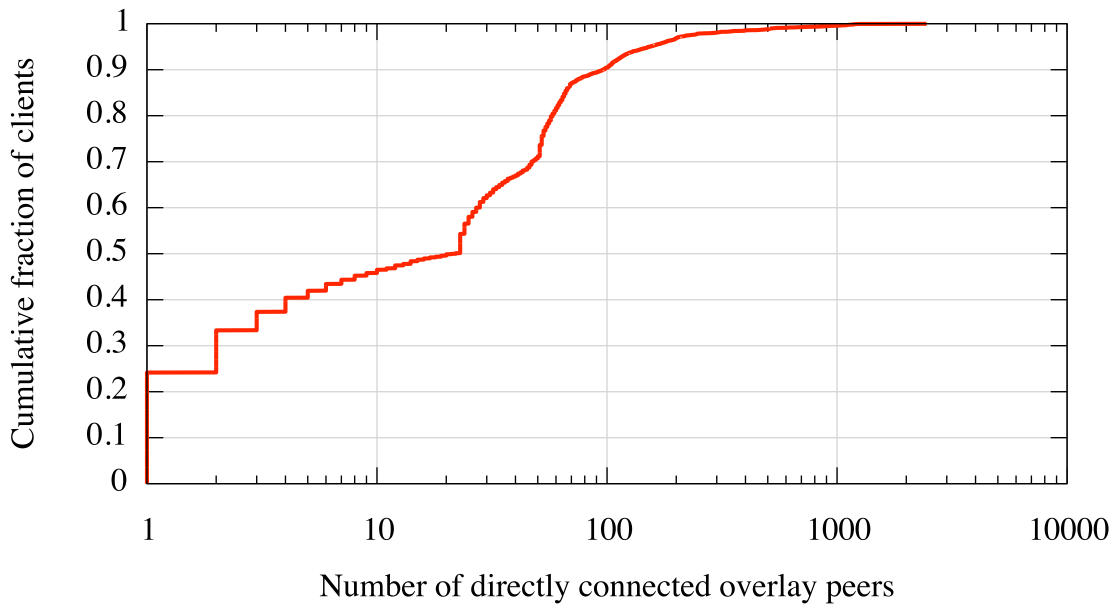

Figure 3.12: Cumulative distribution of peers per-client. The upper half of the bimodal shape is due to community server subscriptions. The wide range of the distribution reflects the diversity of usage behavior.

connectivity for these clients, providing each with dozens of OneSwarm peers drawn randomly from the user population. The PlanetLab nodes act as passive vantage points, measuring the the background traffic generated by users. (This includes both data forwarding and control traffic.) On average, these nodes relay more than one terabyte of data per day.

Overlay structure

Although many overlay links in OneSwarm are based on social relationships, the graph structure is also influenced by the random matching of public community servers, as well as the tendency for some users to import a large number of keys en masse from websites maintaining active user lists.

Both of these effects are reflected in the distribution of overlay peers per user shown in Figure 3.12. This distribution shows significant variations in connectivity. While some users maintain hundreds or even thousands of peer connections, the median value is 22. The sudden increase in mass near this value is attributable to community servers, which return 26 peers by default. Subsequent increases arise from users subscribing to multiple community servers. For clients reporting data, 53% of peers are imported from community servers, 46% are entered manually, with the remaining 1% of peers coming from LAN, email invitations, or social network import. The small fraction of peers arising from pre-existing social relationships reenforces the importance of bootstrapping connectivity via untrusted peers, often from community servers.

Multi-path transfers

Unlike systems that anonymize traffic at the granularity of TCP connections, OneSwarm tolerates out-of-order data delivery, allowing the use of multi-path and multi-source transfers to improve performance and robustness. This is crucial in wide-area P2P environments defined by heterogeneity, since an individual path is limited by the bandwidth capacity of its slowest link. Given the highly skewed bandwidth capacities typical of P2P participants [44], the capacity of any single multihop path is likely to be low.

To confirm this, I compare the multipath transfer rates achieved between PlanetLab nodes during overlay transfers to the performance of separately measured individual forwarding paths. Both distributions are summarized in Figure 3.13. Multi-path transfers average 457 KBps, while single path transfer rates average just 29 KBps. Among PlanetLab nodes, routing single path transfers over Tor yields similar results; transfer rates average 20 KBps. The combination of transient congestion, bandwidth heterogeneity, and potentially lengthy paths all contribute to the benefits of multi-path transfer, which is essential for providing good performance and robustness.

Comparison with existing systems

Tor's comparable single path transfer performance suggests that simply tunneling multiple BitTorrent connections over Tor might suffice to achieve the benefits of multi-path transfers. But, in practice, I find that OneSwarm significantly outperforms Tor. To evaluate this,

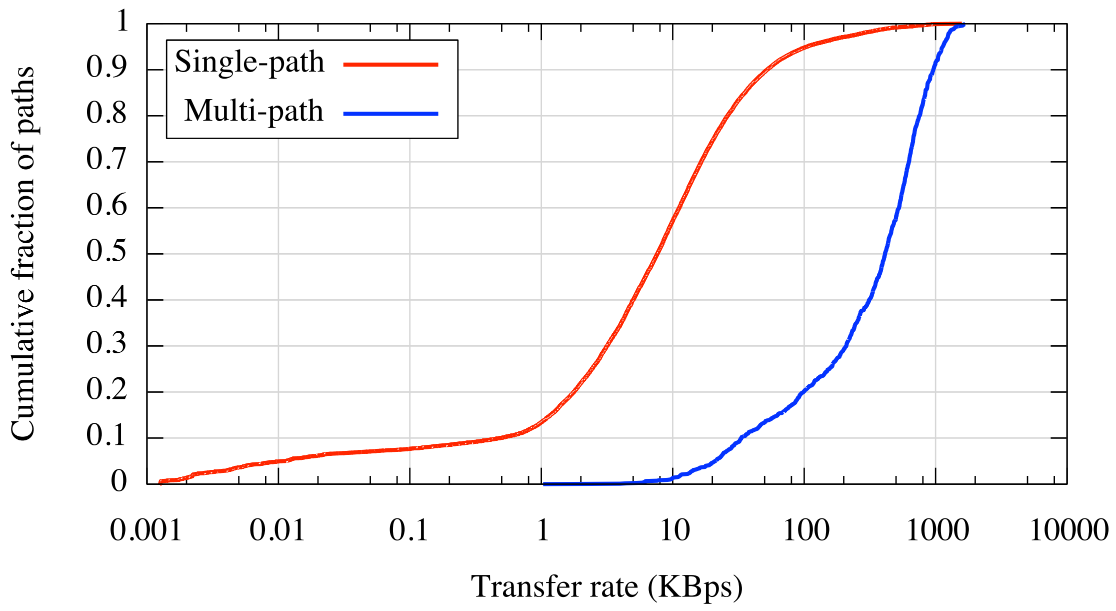

Figure 3.13: A comparison of single and multi-path transfer performance. Because most individual paths are slow, multi-path transfers are essential for achieving good performance.

I compared the transfer performance of BitTorrent, BitTorrent over Tor, and OneSwarm. Tor's reliance on address translation at exit nodes precludes bidirectional connectivity and, when used by a BitTorrent client as a tunneling agent, limits the benefits of swarming data transfer by creating bottlenecks at nodes with bidirectional connectivity. To limit overhead, Tor defaults to creating new paths only once every ten minutes. I modified Tor in my experiments, instead creating new paths every ten seconds to increase the opportunity for multi-path transfers. This configuration provides nearly a factor of 2 performance improvement relative to the default, but aggressive path creation is discouraged as it increases CPU load on intermediate routers. A further improvement would be to modify the BitTorrent tracker and client to understand Tor Hidden Services [66]. This would allow BitTorrent users to tunnel traffic through the core of the Tor network, avoiding the need to transit congested exit nodes. As this modification would require significant changes to both the BitTorrent client and tracker I did not attempt this. In addition to improving the performance of BitTorrent over Tor this modification would have the additional benefit that it would allow all users, and not just a fraction, to have their privacy protected by Tor.

To measure the scalability of BitTorrent over Tor, I compare transfer performance when 50% of downloaders use Tor to performance when 90% of downloaders use Tor. In back-to-back trials, I used each of these methods to download a randomly generated[10] 20 MB file hosted at UW from a set of 120 PlanetLab machines. In each case, all participants joined the swarm simultaneously and remained available as sources after completion. Figure 3.14 summarizes the results.

OneSwarm improves both the performance and scalability of data transfer relative to Tor, which slows down median download times relative to OneSwarm by a factor of 1.9 and 3.4 when used by 50% and 90% of participants, respectively. BitTorrent clients masked by Tor cannot communicate directly with one another, creating a scalability bottleneck as the fraction of Tor users increases. Downloads are effectively serialized by the limited capacity of a small number of detour nodes. The developers of Tor discourage BitTorrent users from using the Tor network to anonymize their transfers. The Tor network already suffers from congested exit nodes, and the Tor developers prefer that the capacity of the network is used from interactive traffic, such as web browsing.

In addition to Tor, I also compared OneSwarm's transfer performance to that provided by Freenet, an anonymous P2P publishing system [20], and found that it provides performance far short of either OneSwarm or BitTorrent/Tor. In Freenet, data distribution is a two step process. First, data is published, which involves proactive caching at several points in the mesh. Afterwards, client requests are serviced from this set of replicas, with more popular files becoming more widely replicated. This differs from OneSwarm where data users never store data persistenty unless they explicitly requested it. Despite the benefit of caching, Freenet yields worse performance.

As in my previous experiments, I attempted to distribute a 20 MB file from a set of

[10]For each experiment I generated a new random file to ensure that no existing copies of the file were available from other users in the network.

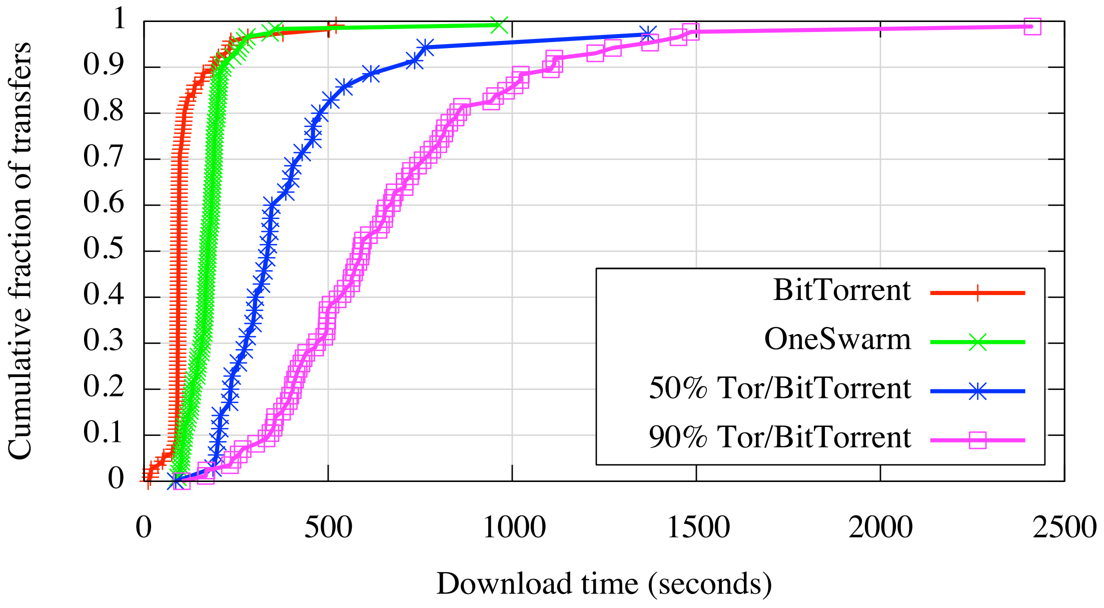

Figure 3.14: Transfer performance of OneSwarm, BitTorrent, and BitTorrent/Tor on PlanetLab. OneSwarm significantly outperforms existing anonymization systems and is performance competitive with BitTorrent.

Freenet nodes running on PlanetLab.[11] But, a large fraction of these transfers failed to complete, and publishing of 20 MB files often failed. Reducing the file size to 5 MB improved robustness, allowing me to compare Freenet and OneSwarm on the basis of transfer rate rather than completion time. For the PlanetLab nodes, Freenet's median transfer rate was just 17 KBps, compared to a median 118 KBps achieved by OneSwarm. This rate does not include publishing time, which would further reduce Freenet's effective distribution rate.

Overhead

Despite performance improvements compared to Tor and Freenet, the results of Figure 3.14 suggest that OneSwarm incurs a performance penalty relative to BitTorrent. I attribute this difference largely to the resource constraints typical of PlanetLab nodes rather than a fundamental performance property of OneSwarm. OneSwarm transfers are encrypted

[11]I allowed these nodes to operate for several hours before my experiments in order to quiesce in Freenet's mesh.

while BitTorrent transfers are not, and the OneSwarm transfer rate for (oversubscribed) PlanetLab nodes is often limited by the available CPU rather than their network capacity.

To verify this, and to directly measure the influence of multihop forwarding on transfer performance, a comparison is made between OneSwarm transfers 1) mediated by the overlay and 2) using a direct point-to-point connection between sender and receiver. (In both cases, transfers are encrypted, providing an apples-to-apples comparison.) If the overlay is not capacity constrained, the expected behavior is that both both direct and overlay transfers, on average, have a similar duration. The experiments show this to be the case for transfers conducted between PlanetLab nodes.

Figure 3.15 summarizes the ratio of the overlay and direct OneSwarm transfer times between PlanetLab nodes. The figure is based on transfers between 75 pairs chosen randomly while all other PlanetLab clients were disabled; i.e., the overlay did not benefit from any forwarding capacity beyond that of its existing user base.

A ratio of 1.0 means that overlay and direct transfers took identical time, with ratio > 1 indicating a faster direct transfer and ratio < 1 indicating a faster overlay transfer. This is a challenging case for OneSwarm as PlanetLab nodes are generally of higher capacity than typical OneSwarm peers, which are often hosted from ordinary home broadband connections. Even without the addition of PlanetLab forwarding capacity, overlay transfer does not impose a performance bottleneck in most cases. Some transfers are faster, and some transfers slower, but the median ratio of overlay and direct transfer times, 0.94, suggests that overlay forwarding is not a fundamental performance bottleneck.

Next I investigate how transfer performance in OneSwarm is affected by the number of directly connected peers. The transfer measurements are repeated while restricting the number of peers connected to each PlanetLab node to a randomly chosen value between 1 and 35. Performance increases with the number of connected peers. For example, increasing the number of connected peers from 17 to 29 doubles median transfer performance. But, returns are diminishing; a further increase to 35 peers improves median performance

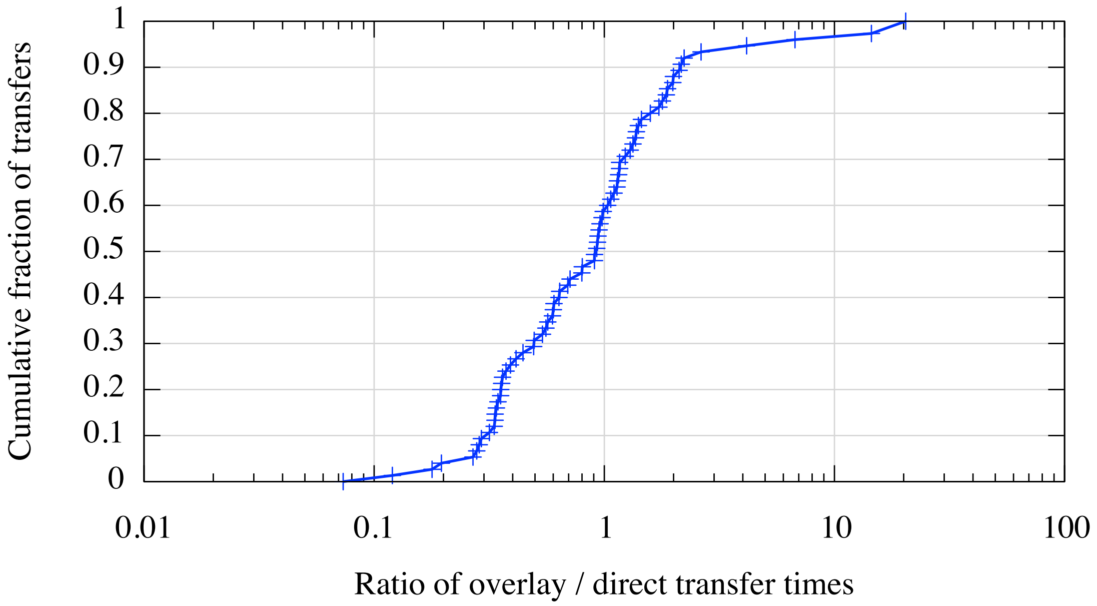

Figure 3.15: Comparing transfer times mediated by the OneSwarm overlay to direct transfer. Averaged over many trials, overlay transfers are performance competitive with direct point-to-point transfers.

by just 1%. The default value for the maximum number of peers provided by a community server—26—reflects the tradeoff between client performance and resistance to systematic monitoring (Section 3.4). Of course, this value is a configurable parameter.

I attribute the significant variability in a minority of transfers times (the tails of the distributions in Figure 3.15) to the underlying unpredictability of PlanetLab hosts, which vary greatly in their available CPU and network bandwidth, even over short time scales. Variability in either affects the performance of encrypted transfer.

Utilization

Although the overlay benefits from a surplus of capacity in aggregate, individual paths and individual nodes are often congested, motivating the use of congestion-aware search and multi-path transfers. To confirm this, I examine each user's reported utilization over time. For the set of users reporting transfer volume statistics, I compute the maximum transfer rate over all reported 15-minute intervals and treat this as the capacity for a given

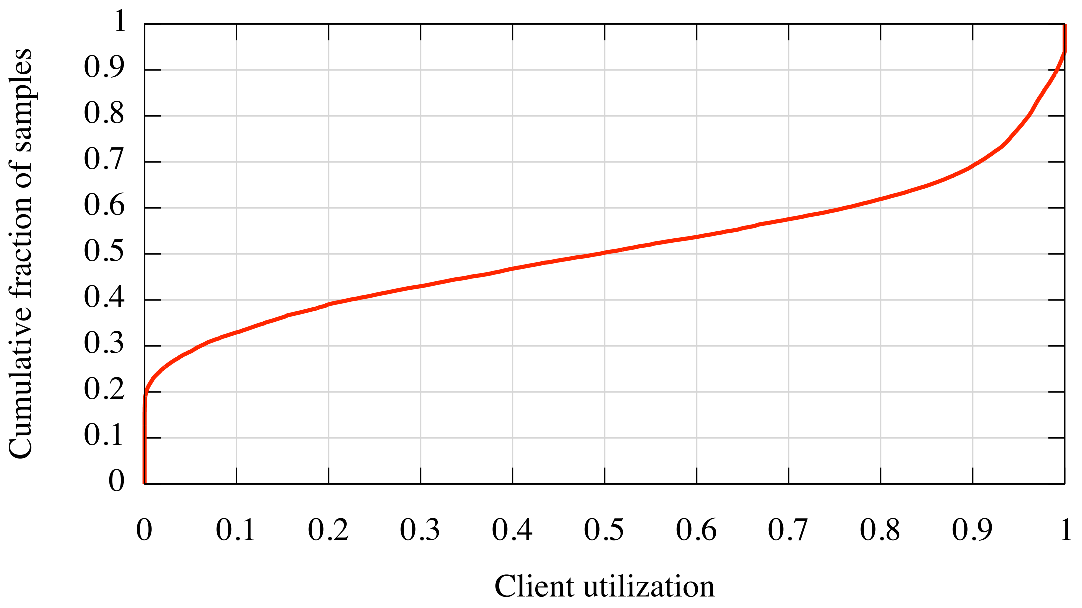

Figure 3.16: The distribution of client upload capacity utilizations over the course of one day. Although most clients have excess capacity, transient congestion occurs at many nodes.

IP address, computing utilization for all other 15 minute periods relative to this maximum. These samples are summarized in Figure 3.16. Although average utilization is 49%, many nodes are frequently bandwidth limited; node utilization is 95% or greater during 23% of measured intervals. In short, temporarily overloaded clients are not uncommon despite the overlay being over-provisioned on average.

3.5.2 Trace replay in the last.fm *social graph*

To evaluate OneSwarm's operation at grater scale than its current deployment, I use the last.fm trace data described in Section 3.1 to drive a large-scale simulation. While last.fm is focused on music, making it slightly different than OneSwarm where any data can be shared, I chose last.fm as it the only service I know that publishes both the social network and the media consumption of its users. This data allows us to parameterize the simulator with real values for both user interest and social network.

Unlike Section 3.4.3, I do not assign clients peers from community servers, and so the availability results should be taken as a lower bound of availability (additional paths would increase redundancy), and the overhead results an upper bound (random shortcuts would lead to shorter paths and reduced search propagation).

The main result is that relying solely on the social network is feasible for the measured last.fm workload; 94% of requests that can be fulfilled by the overlay (i.e., at least one replica is online and reachable) result in successful downloads, and the majority of transfers use the shortest paths available. In support of this main conclusion, I provide additional analysis of the details of OneSwarm's operation including search overhead, utilization, sensitivity to node lifetimes, and variation in contention across overlay paths.

Methodology

The last.fm trace data drives a discrete event simulator with ten second timesteps. Each last.fm user is interpreted as a OneSwarm user, friend links in the last.fm social graph correspond to OneSwarm friends, and each unique song request made by a user is interpreted as an object request in the overlay network. Searches are cancelled when 10 distinct paths are discovered. Once a user has downloaded an object, the user serves as a replica.

I assume that all users have unconstrained download capacity, and each user is assigned an upload capacity limit drawn from a measured distribution of BitTorrent capacities [44]. During periods of contention, capacity is split approximately equally between flows and search messages are not forwarded.

Initially, each user serves as a replica for songs that user listened to during the first week of the trace. I begin the trace playback at the outset of the second week. For all users, I have a record of playback histories at the granularity of weeks. I translate these coarse-grained playback histories to the shorter time-scales required for simulation using request frequencies and session lengths measurements of 1,000 randomly sampled users from the trace. At each point during trace playback, clients are added so as to approximate the measured diurnal pattern exhibited by the sampled users. Users are randomly selected to join the system. Each user is assigned a session time (in songs to be played) drawn from the distribution of sampled users. I consider a session to be active until back-to-back playback of songs is separated by more than 30 minutes.

Object sizes are derived from the measured lengths of 81,805 songs listened to by the set of 1,000 sampled users, and I assume a constant data rate of 128 Kbps. To exercise capacity constraints, I increase this data rate to 1 Mbps for indicated trials; this rate is consistent with high quality streaming web video.

To examine the impact of unavailable nodes in the social graph on object availability, I compare trace playback 1) when all users observed in the last.fm trace are active (I refer to this as “always on"), and 2) when users persist in the overlay for eight hours after playback of the final song of their session. Eight hour lifetimes approximate typical peer lifetimes in BitTorrent [38]. Relative to BitTorrent, OneSwarm users have an incentive to keep their systems online (to accumulate credit from friends for later use), and less disincentive in terms of privacy disclosure. “Always on” provides a bound on the benefit from these incentives.

Each trace playback is primed with eight hours of simulation time, and I report results for three subsequent hours of simulation time spanning peak load during the first day of the second week of the trace. Simulation during peak load exercises capacity constraints. I also performed a similar analysis of OneSwarm’s operation during minimum load, exercising availability.

Object availability

A simple metric that distills the feasibility of overlay forwarding is the fraction of objects requests satisfied; i.e., those that discover at least one replica in the overlay. During trace replay, 11% of searches fail for the last.fm workload with both always on and 8 hour lifetimes during peak load. During simulations spanning the time period of minimum load, the fraction of failed searches increases to 24%.

In the simulation, OneSwarm searches can fail for any of three reasons: 1) the song being requested occurred only during the second week of the trace (no replicas exist), 2) all available replicas are offline, or 3) no path exists to the query source from available replicas due to either overloaded or unavailable nodes along the path. Object requests of the first type (no replicas exist) account for 6% of total demand in the trace. These searches are certain to fail and correspond to the songs listened to by just one last.fm user in the trace. This implies that the remaining cases (capacity overload and/or replica unavailability) cause search failures in just 5% of cases during peak load and in 18% of cases during minimum load. This level of robustness is surprising given that most nodes have few friends, most songs have few replicas, and average path length between random users is high. Several workload properties ameliorate these challenges. First, although users have few friends, even these users are frequently connected to high degree nodes, providing high path diversity. And, although most songs have few replicas, most requests are for popular objects for which many replicas exist (along many paths). This also compensates for typically lengthy path lengths among users. Users with few friends experiencing poor performance can add untrusted peers from a community server to improve their connection with the overlay, shorting paths and improving availability.

Overhead

OneSwarm discovers paths to replicas by flooding search messages among friends. Search overhead is computed as the fraction of control messages making up overall traffic. For the last.fm workload with always on lifetimes, overhead is 27% of total data traffic. Assuming the increased data rate of video playback, while keeping the request pattern the same, reduces the fraction of overhead to 6%. Overhead with 8 hour lifetimes is higher than when peers are always on since the relative low density of the graph makes it difficult to find the 10 unique paths required to cancel the search. For peers with 8 hour lifetimes, the overhead is 77% for the last.fm workload and 43% for the video workload. Although large

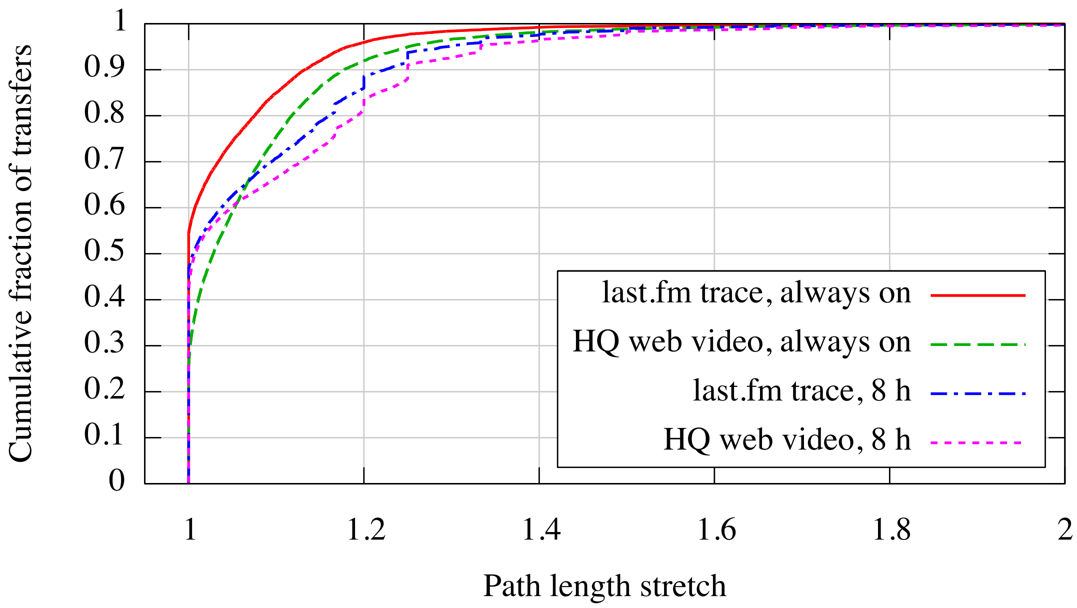

Figure 3.17: Path length stretch for various workloads. For the last.fm workload, the majority of transfers use shortest paths through the overlay. As data volume increases, capacity constraints induce stretch.

both fractionally and by total volume, recall that search messages are forwarded only when a node has idle capacity. As a result, capacity consumed by control traffic is not capacity lost during data transfers, assuming unconstrained downlink capacity.

An alternative search strategy trades worse search response latency for a reduction in overhead. Because search messages are forwarded only after a 200 millisecond delay per-hop whereas search cancel messages are forwarded without delay, OneSwarm clients can perform the equivalent of iterative deepening search in the overlay by issuing search queries followed by increasingly delayed search cancel messages. For popular objects with many replicas, this approach discovers replicas quickly. But, if lengthy paths are required, several iterations (and several seconds of delay) may be required to discover replicas.

Stretch

In addition to promoting availability by discovering potentially rare replicas, flood-based search also typically discovers short paths. When objects are large, trading control traffic for short paths is preferable; redundant forwarding of bulk data consumes traffic equivalent to an enormous volume of relatively tiny control messages. I measure how often OneSwarm discovers (and can use) the shortest available paths by computing the path length stretch for transfers during trace replay. I compute stretch as the average path lengths to all replicas used during a file download weighted by the fraction of total data attributable to a given replica. The distributions of stretch for various workload conditions are shown in Figure 3.17.

The last.fm workload with always on lifetimes is the best case, assuming no community servers. Path diversity is high and aggregate demand is much less than aggregate capacity. In this case, OneSwarm uses shortest paths for 55% of transfers with an average path length from source to replica of 4.8. 95% of objects have a stretch $\leqslant 1.2$. Path diversity is reduced when lifetimes decrease (8 hour, average path length 5.1); this increases stretch. In both cases, a small fraction of requests traverse paths with frequent contention, increasing stretch. Increased data rate (1 Mbit web video) increases stretch as well, but this increase is attributable to contention for bandwidth rather than node unavailability. Even with always on lifetimes, just 28% of video transfers use shortest paths (average path length 5.8). Reducing lifetimes decreases path diversity. This means a larger fraction of downloads use the shortest available path, but this path is longer than the shortest path available with always on lifetimes.

Contention

One measure for the load on the OneSwarm overlay is the contention for capacity at each node. I measure contention as the ratio of simulated download time and the download time if no concurrent transfers occur along all transfer paths. I call this the slow-down factor. If a given transfer is the only active transfer on its set of paths, this ratio is 1.

The distributions of contention ratios for the simulated trials are shown in Figure 3.18. For the last.fm workload with always on lifetimes, contention is rare; 73% of transfer paths

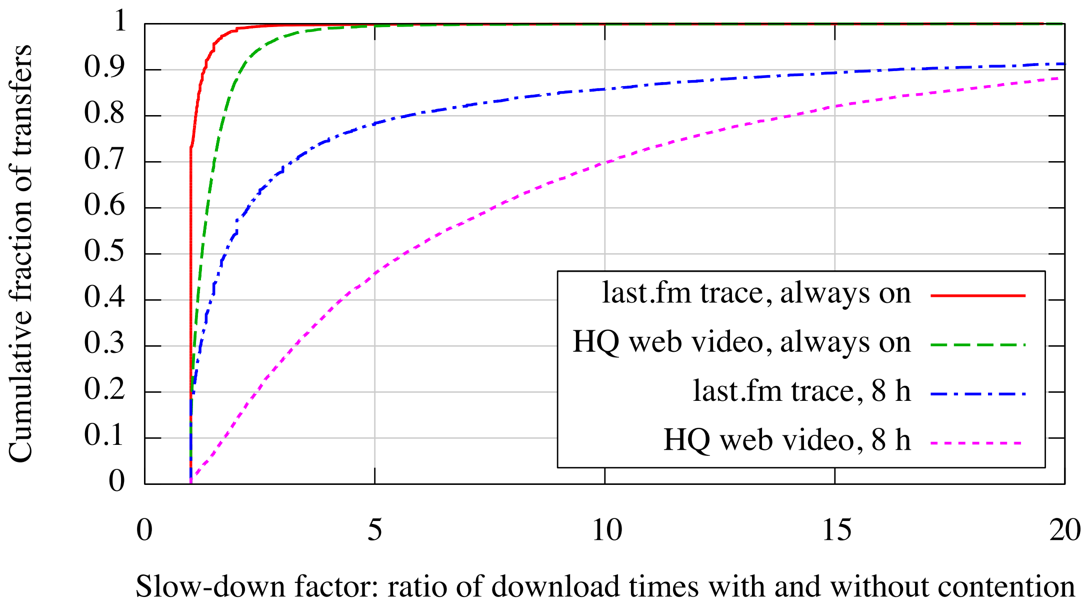

Figure 3.18: Slowdown due to contention. Each line gives the cumulative fraction of transfers (y-axis) with a given slow-down factor (x-axis) or less.

are not contended throughout the transfer. As expected, contention increases when either lifetimes are reduced (capacity and path diversity decrease) or data rate increases (capacity constraints influence performance). In the case of 8 hour lifetimes, many paths have high contention ratios since there are fewer paths available. When contention is high, stretch increases as core nodes shed load, leading to longer paths. Load balancing over longer paths of more highly utilized nodes can yield substantial slow-down for many transfers. Note that Figure 3.18 does not show slow down factor > 20. With 8 hour lifetimes 9% of transfers fall into this category for the last.fm workload and 12% for the web-video workload.

3.6 Summary

Although widely used, currently popular P2P file sharing networks expose users to silent, third party monitoring of their behavior. To address this, I have built OneSwarm, a file sharing system designed to reduce the cost of privacy to the average user. With novel techniques for efficient, robust, and privacy-preserving lookup and data transfer, OneSwarm provides users with flexible control over their privacy by defining sharing permissions and trust at the granularity of individual data objects and peers. The OneSwarm client is publicly available for download on Windows, Mac OS X, and Linux, and it is in widespread use around the globe. Measurements of the live OneSwarm deployment shows that it delivers on its promise: OneSwarm privacy-preserving downloads are performance competitive with BitTorrent and substantially outperform existing anonymization networks.

Chapter 4

UNBLOCK

In this chapter I describe the design of Unblock, a system for defeating Internet censorship. Unblock is based on an overlay constructed from an augmented social graph. Data is encrypted and forwarded in the overlay to an exit node located in a political domain where the requested data is available. A key design point is that trusted social links protect individual participants from exposure to adversaries. In addition, Unblock implements a novel method for introducing additional links in the network graph which improves connectivity without a significant increase in vulnerability. To improve performance, Unblock use an adaptive data transport technique that minimize latency and improve throughput compared to Tor. With measurements of a prototype implementation, and simulations of the system at scale I demonstrate its practicality for web traffic workloads.

My goal is to develop a system that is resistant to a censoring adversary, without requiring changes to the underlying Internet infrastructure. In the threat model for Unblock, a censor has control of some number of the physical links in the network, and can block communication with arbitrary IP addresses across these links. A censor can also try to join the system in order to discover the locations of users. Thus, upon discovery of any user's operation in the system, a censor can block further access to that user's IP address from within its administrative domain. Censors cannot read the contents of the encrypted traffic and users can (and often do) change IP addresses. Furthermore, the censor is not willing to disable all encrypted traffic to avoid impact on legitimate applications such as VoIP, gaming, and media streaming.

In order to demonstrate that Unblock is practical for wide-spread use I have implemented it as an extension to OneSwarm. I evaluate the performance of the prototype in controlled testbed settings and measure its ability to perform web requests under various configurations. I also evaluated the social network augmentation techniques using a simulator to measure the implications of the design decisions at scale. My measurements show that Unblock provides high availability and improved performance without introducing any significant vulnerabilities.

4.1 Challenges in building blocking resistant services

There has historically been two approaches to building overlay-based services. In this section, I consider the suitability of these approaches to building robust censorship-resistant overlays, the challenges that arise given a censoring adversary, and the inherent structural properties of the two approaches.

- **Public open-access overlays**: These systems are characterized by (a) their use of a centralized management system that publishes information regarding public relays which any user can utilize to route their traffic, (b) any relay can be used by any client to construct a circuit. These include privacy systems such as Tor [27], Ultrasurf [96], and Freegate [36], as well as open proxies and VPNs, which are also used to evade censorship [41].

- **Social network-based overlays**: In these systems, the identity of participating users remain hidden and traffic is routed through connections that are bootstrapped on preexisting trust. In addition to OneSwarm (see Chapter 3) these include systems such as Ostra [63] and Freenet [20].

I discuss the strengths and weaknesses of the two types of overlays in the remainder of this section.

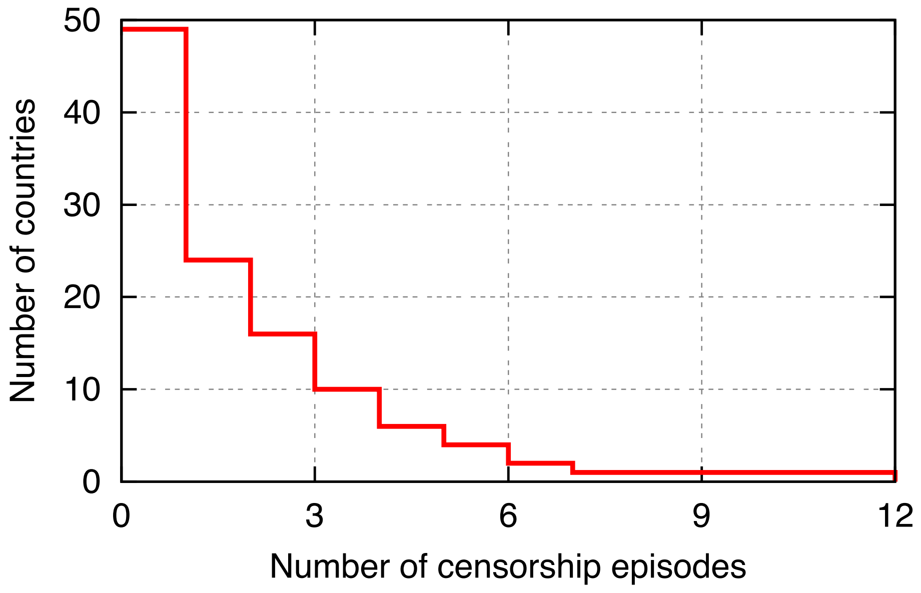

Figure 4.1: Number of observed censorship episodes against Tor (i.e. blocking Tor when it was previously not blocked). Surprisingly, Tor is blocked in many countries.

4.1.1 Ease of blocking open-access overlays

Open-access overlays are perhaps the most publicized approach to anti-censorship. At the heart of the overlay is a number of open proxy machines that can be used to forward traffic to a desired destination. Traffic is encrypted at each hop to protect its content, and forwarded over multiple hops to obscure the final source and destination. When the traffic reaches an exit node, it is forwarded to the intended destination. Several countries have been observed to actively censor access to open relays [92].

Fundamentally, open access systems are hindered by the fact that bootstrapping of new nodes requires the distribution of the public addresses of these nodes. As mentioned in Section 2.3.1 Tor provides a centralized service that returns a certified list of relays making it trivial for a censor to block access to this list of addresses [26].

While countries that actively block Tor are well known [26], a surprising number of countries occasionally block Tor. Figure 4.1 shows the results of an experiment that measured Tor access in 152 countries over the period from January 2010 to November 2011

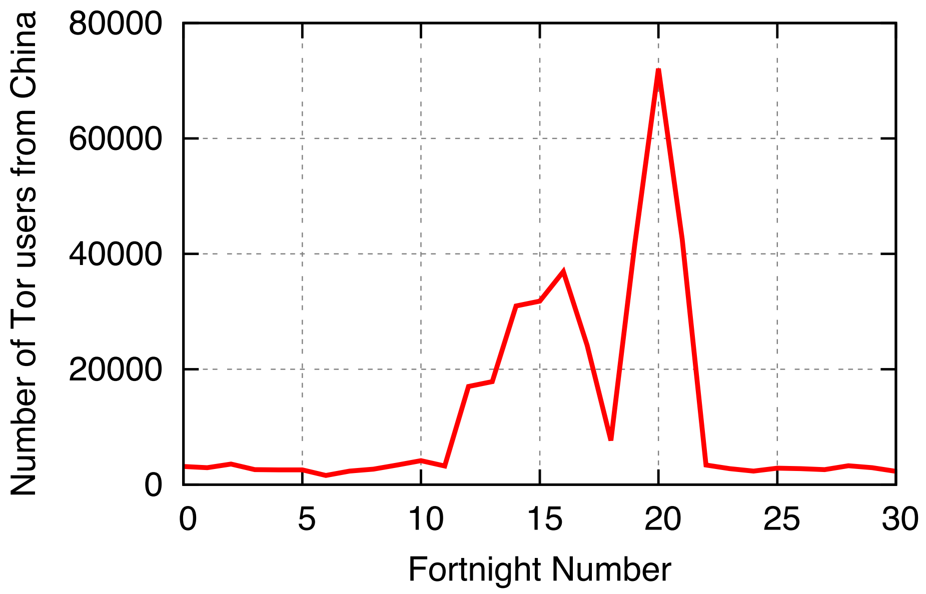

Figure 4.2: Number of Tor users from China over a 60 week time period. Usage spikes twice when new relays are added, but bottoms out when the new relays are blocked.

using a methodology developed by the Tor developers [24]. The number of clients that connected to each of the Tor directory servers were aggregated into two week periods by country. These totals were compared with the preceding period. Finally, these ratios were normalized to the total number of Tor users around the world for the 2 corresponding periods. The two week period acts as a low pass filter, minimizing variations in usage and overcoming weekly patterns. By normalizing against the global Tor user count, the analysis also accounts for overall trends in Tor usage.

A censorship episode is defined as an event where the Tor usage in a country where Tor usually works drops more than 4 standard deviations below expectation. I was surprised to find that out of 152 countries, 49 countries had experienced an episode of such censorship during the 2 year period, and several experienced multiple such episodes. These results imply that while the governments of many nations are debating what should be censored, many networks are already equipped with devices capable of censorship on a nationwide scale.

In Figure 4.2 shows the number of Tor users from China, a country known to actively

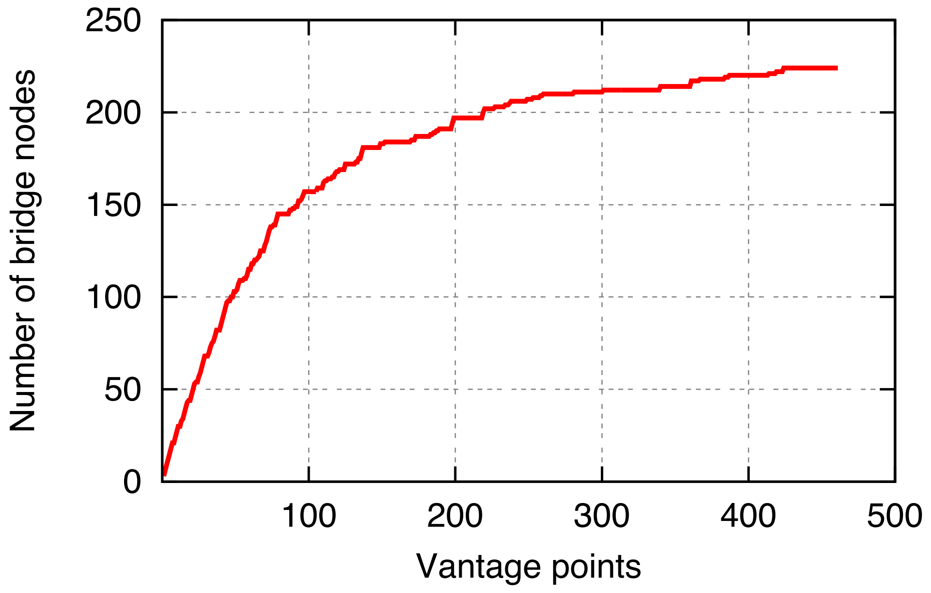

Figure 4.3: Number of exposed bridge nodes vs number of PlanetLab vantage points. By launching a Sybil attack from multiple locations, we can discover the number of bridge nodes and the IP address of each. A censor can use a similar technique to block Tor bridges.

block Tor, over a 60 week period. I found that Tor was essentially blocked and ineffective for the majority of the time. During the middle of the experiment, there were two spikes in usage when new relays were added. However, each instance was short lived as the new relays were blocked and usage subsequently dropped. Other measurement initiatives have reported similar data [92].

In response to censorship, open-access overlays may add new relays to the system. These relays are also vulnerable to blocking. Recently, Tor has added semi-secret relays called bridge nodes, which are disclosed to users in a limited, decentralized fashion. However, once a bridge node's IP address is discovered by a censor, it can be blocked. An adversary can use the same protocol to query the members of the overlay as the intended users to easily identify relays participating in the system. Even if systems use decentralized methods to limit the number of nodes exposed to any particular user, the availability of large numbers of hosting services makes it easy for crawlers to obtain node identities from a diverse set of IP addresses. In Figure 4.3 show a crawl of the Tor bridge discovery mechanism from multiple vantage points on PlanetLab. I found that by requesting bridge nodes from multiple locations, I was able to discover nearly 240 bridge nodes in Tor and their IP addresses. One could imagine a censor being able to use the same technique to discover and block new bridge node IP addresses to render this technique effectively useless. In fact the Chinese government performed this exact attack and quickly discovered all bridge nodes given out based on the source IP address. Shortly after they created a large number of gmail accounts and crawled all bridge nodes given out based on gmail source address [92, 26]. Tor employs a third method for distributing bridge nodes using social relations ships between Tor developers and activists. The activists then spread the bridge node IPs in their own personal networks. Giving out bridge nodes based on direct social links decreases the rate a censor discovers them, but limits the reach to a few select users. If these users in turn share bridge node locations with their friends it is likely that the censor will eventually learn of the location. The effectiveness of this method is encouraging for Unblock. Especially since Unblock provides a safe way for users to share overlay links crossing the censorship domain with their friends by only exposing the valuable cross censorship domain link to a few users. This limits the number of nodes that has to know the location of the cross censorship domain link, decreasing the probability of revealing it to the censor.

4.1.2 Social Network Based Overlays Have Poor Connectivity

Social network overlays have been explored in the past to improve trust and security in network systems. For example, Ostra [63] was a system that used social networks to impose a cost on unwanted email, and OneSwarm provides anonymous P2P file-sharing using social network routing. However, building a censorship-resistant system using a social network overlay presents new challenges. Particularly, routing traffic over a social network introduce additional latency as data is forwarded over multiple hops. While this is not an issue for latency insensitive services such as file sharing, it poses a challenge for interactive web services.

In social network based overlays, users explicitly define peers that they trust and all traffic is routed over the formed social network. Users can volunteer to serve as exit nodes in a particular geographic domain. For example, a user in the US can serve as an exit node to provide access to US websites for the users of the system. Data is forwarded from the source to the destination along a sequence of social network connections.

Because each user is only exposed to the neighbors that it explicitly defines as trusted, social network overlays conceal the identities of participants. However, routes in the the network are constrained to naturally formed social links and no centralized service has the ability to shape the overall topology of the network. Due to this constraint, these overlays must cope with unfortunate network topologies that arise from the underlying social connections and the inherent churn in the system.

I analyzed network properties of a number of social networks, including Youtube, Flickr, LastFM, LiveJournal, and Foursquare using datasets collected by [62, 80]. Irrespective of the size of the overall social network, most nodes in these networks have at most a handful of connections to other peers and a large number of users have only one social link. Few links translates to poor path diversity in the best case, and complete disconnection in the worst case. Compounded with the issue of churn and limited uptime of end hosts, the existing topology of these graphs would ensure poor performance for the majority of users.

I used simulations to measure the availability of paths through these social networks under varying churn. The results below examine the ability to obtain a working path to an exit node when 10% of participants serve as exit nodes. Figure 4.4 shows the results of this experiment. The availability of working paths is highly susceptible to churn. Due to the *stringy* nature of these social networks, churn easily disconnects some nodes entirely from any exit node, lowering the total connectivity to exit nodes from 100% to around 50% for

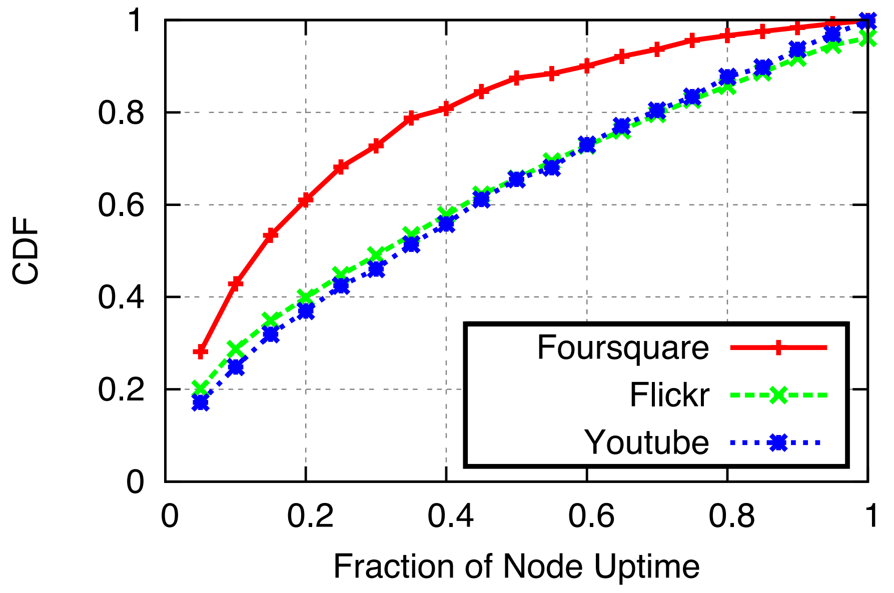

Figure 4.4: Fraction of nodes with paths to exit nodes on different social network datasets for varying node uptimes and with 10% of the nodes being exit nodes.

typical node uptimes seen in peer-to-peer systems [89, 37, 78, 55]. There are other unfortunate consequences of a constrained network topology. First, as nodes fail, the remaining nodes must route around them. This would result in longer paths to exit nodes, and higher latencies unsuitable for interactive web access. Second, the constrained network topology results in hotspots, especially under churn. In the simulations, a small set of nodes are on hundreds or even thousands of paths making them critical to the success of the system. The performance of these networks is thus limited by the availability and bandwidth of this small set of nodes, thus reducing overall system robustness.

4.1.3 Transport Inefficiencies in Overlay Networks

In general, overlay networks have suboptimal performance as data is transferred sequentially in and out of the Internet core multiple times in order to traverse the overlay, resulting in higher latency and greater possibility of traversing congested links. The Tor network exemplifies these issues, and it also suffers from over-subscription and poor congestion control[93, 28].

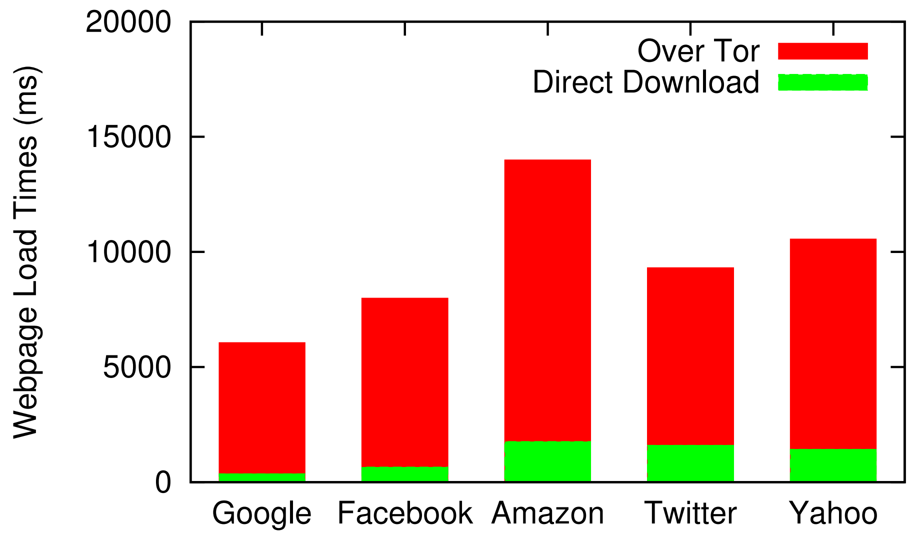

Figure 4.5: Slowdown introduced by Tor when loading popular domains in January 2012.

Measurements of page load times over Tor compared with "normal" direct connections from a set of 17 geographically diverse PlanetLab nodes exemplifies this. As can be seen in Figure 4.5, page load time for popular sites increase $6 - 17$X when web pages are loaded over Tor. Figure 4.6 shows the same experiment for the Alexa top 100 sites [3]. The median page load time increases from 2.3s to 12.6s[1].

The performance issues of overlay networks are not fundamentally limited by the use of end hosts or low bandwidth nodes in the network topology. Peer-to-peer systems like BitTorrent offer excellent performance, and can transfer large amounts of data even though the majority of participants have low upload bandwidths [70]. Forwarding traffic over multiple end-hosts limits the throughput to the slowest link. Recall measurements of the OneSwarm overlay in Section 3.5.1. End-to-end throughput in the overlay averaged 29 KBytes/s leading to poor performance unless multiple paths are used. The motivation for choosing multi-hop paths in overlay networks has traditionally been two reasons:

[1]Roughly 20% of Alexa top 100 domains did not load successfully during the experiment. The reasons include: no main page (e.g. googleusercontent.com, bp.blogspot.com), blocking either by institution (in the case of pornography) or a censor (e.g. taobao, qq). Many Chinese sites are not accessible either from Tor, or PlanetLab.

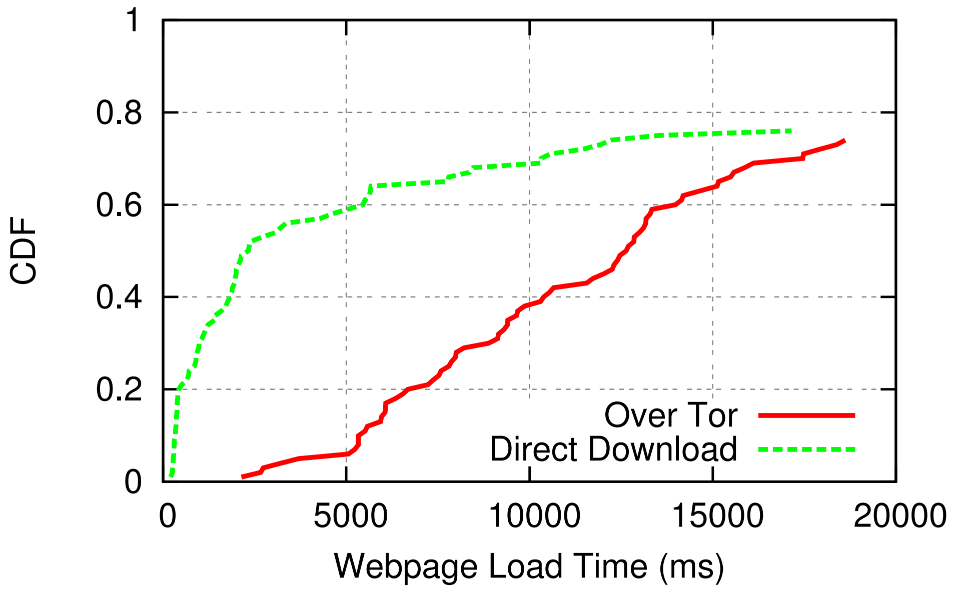

Figure 4.6: Load times for Alexa's top 100 domains in January 2012. Measurements are averaged from 17 geographically diverse PlanetLab nodes.

(a) anonymity is gained through indirection and aggregation and (b) application agnostic transport needs to guarantee in-order, end-to-end delivery. These reasons, not the available resources, are the major contributor to why systems like Tor are slow.

Location is an important factor in overlay path selection since it often is correlated with link latency. Nodes located nearby geographically can likely communicate with low latency and be connected by relatively few intermediate routers. One of the benefits provided by social network-based overlays is that the majority of social links connect nodes which are geographically close [80]. In contrast, many open access overlays like Tor do not use location as a factor in path selection. This increases the average hop latency, and contributes to high end-to-end latencies in the overlay.

Unblock aims to combine the security, concealing, and locality properties of routing over social networks, with the many innovations in transport optimizations of open access overlays in order to build a well-performing blocking-resistant anti-censorship system. The following section will describe the various mechanisms used to achieve the desired properties.

	Mechanism	Description	Purpose	Section
Topology	Untrusted links	Peers initiate random walks in the overlay to find shortcut paths to distant peers in a crawl resistant fashion.	Decrease overlay diameter, improve connectivity, without revealing membership to censor.	4.2.3
	Overlay-internal DHT	Overlay nodes maintain an encrypted key/value storage service mapping peer identities to current IP address and port.	Connection bootstrapping and name resolution for overlay IDs.	4.2.6
Datatransport	Locating exit nodes	Exit nodes periodically announce their existence to the overlay refreshing the routing table at overlay peers. Peers initiate a limited scope search for different branches to allow multi-path.	Create minimum latency routing paths, announce exit nodes, enable multiple paths.	4.2.4
	Hybrid transport layer	Reliable SSL connections are used between peers for control messages, encrypted UDP channels with back-pressure flow control are used for low latency overlay forwarding.	Reliable control traffic, low latency overlay forwarding.	4.2.5
	Multiple overlay paths	Clients automatically adjust redundant packet transmissions to minimize latency for short flows and maximize throughput for high bandwidth flows.	Minimize latency, maximize throughput, improve robustness.	4.2.5

Table 4.1: A summary of protocol mechanisms used by Unblock.

4.2 Protocol design

Unblock aims to combine the security, privacy, and locality properties of routing over social networks with the more robust connectivity properties of open access overlays in order to build a high-performance blocking-resistant anti-censorship system. This section will describe the various mechanisms used to achieve the following desired properties.

- **Availability:** The system should provide high levels of service availability and be capable of constructing usable overlay paths despite churn or sparsely connected regions in the underlying social network.

- **Security:** The system should be able to withstand various attempts to block communications or disrupt the normal operations of the overlay. In particular, the system should minimize the impact of infiltration attacks and conceal the identities of a large fraction of participants to ensure smooth operation.

- **Usability:** Users might use the system to circumvent censorship, achieve anonymous communication, or to avoid surveillance. The system should be able to handle different use cases, and achieve higher service quality made possible by the larger user base.

- **Performance:** The system should provide sufficient performance to enable interactive web access. Further, the overhead should be acceptable to users attempting to avoid surveillance in addition to those accessing otherwise censored content.

4.2.1 System Overview

Unblock achieves the desired properties by combining a set of techniques that are outlined below.

Augmenting the social graph: Relying on social network links is insufficient for users with few social connections. This is especially problematic for new users joining the overlay with a single initial trusted friend. To decrease the dependence on the initial peer, to increase connectivity for poorly connected users, and to provide connectivity when churn systematically reduces the availability of overlay routes, Unblock augments the social network with a *random overlay network* that allows users with few connections to (optionally) add additional peers located at random locations in the overlay network. In addition to providing the user with redundant connectivity to the overlay, these *untrusted links* provide shortcuts to remote locations in the overlay. This improves availability in the face of node churn, but potentially exposes the system to adversarial attacks. Details on how to add untrusted links while limiting the disclosure of overlay relay identities to adversaries is discussed in Section 4.2.3.

Resilient communications: The system should minimize its reliance on external services with the potential to be blocked. For instance, the system should not use an external DHT as such access can be easily blocked by the censor. Instead, the system and its operations should be self-contained to maximize blocking resistance.

A peer joins the Unblock overlay by exchanging bootstrapping information with an existing peer. This information includes the current network location of the peer, a one-time use capability authenticating the recipient during initial connection, and a public key allowing the recipient to verify that the remote side is who they claim to be. Returning peers are able to connect to the overlay as long as a discovered peer remains at its previously known network location. Once connected to the overlay, a new node can discover remaining peers' current addresses through an *overlay-internal DHT*; each peer maintains encrypted entries containing their current IP address and port in this overlay, which can be decoded by authorized peers. Discovery and communication with exit nodes is discussed in Section 4.2.4 and connection setup is detailed in Section 4.2.6.

High performance overlay transfers: To encourage user adoption, the system needs to minimize performance overheads associated with interactive web browsing. User satisfaction degrades quickly as page load latencies increase. On the other hand, small performance improvements can increase adoption and usage, which in turn provide greater path diversity – further improving performance.

Nodes in Unblock employ a *hybrid transport layer* that uses both SSL-over-TCP and encrypted UDP channels. The reliable SSL connection is primarily used for control traffic and as a fallback where UDP is blocked by firewalls.[2] Data is preferably transported over UDP, since forwarding multiple independent flows within one TCP connection has significant issues with congestion from large flows causing packet-loss and retransmission delays for all flows in the stream [76]. Handling congestion at the overlay-flow level allows Unblock to use novel techniques such as back-pressure based flow control and fair-queueing to ensure low latency and fair bandwidth allocation. The details of the transport are discussed in Section 4.2.5, and evaluated in Section 4.3.1.

Providing low-latency overlay connectivity over a chain of unreliable and slow end-hosts requires the endpoints to be dynamic in their behavior. To improve performance, Unblock uses *multiple parallel overlay paths* when available. Low latency flows benefit from redundant data transfer, which ensures robustness against churn, intermittent queueing, and packet loss on any single path. For high bandwidth flows the data is striped across paths to improve throughput. Unblock can dynamically adjust how redundantly data is transmitted depending on the current bandwidth characteristics.

Table 4.1 provides a road map for the rest of this section. Next I describe the threat model, and then expand on the design outlined above.

4.2.2 Adversaries and threat model

I expect the Unblock network to be under continuous attack by adversaries. I assume that the adversary is able to block DNS and IP addresses. I also assume that the adversary

[2]SSL, especially on port 443, is rarely blocked due the popular use of HTTPS.

is able to infiltrate a limited number of social links giving it access to the overlay. From nodes it controls it is free to generate, modify, and delete messages flowing through its nodes, record timing and other information, and correlate traffic from multiple nodes. I call nodes controlled by an adversary *Moles*, as they infiltrate the social network to spy and wreak havoc in the network.

Importantly I assume that the adversary is unable to break cryptographic primitives, seize client machines, or otherwise intimidate users into revealing information about their friends. I realize that these assumptions are insufficient for users in some countries but I believe that it still can provide a valuable service for a large number of users under varying levels of Internet censorship. Users in countries where authorities routinely round up and torture citizens circumventing Internet censorship *should not* use Unblock.

4.2.3 Adding Untrusted Links

In order to improve network connectivity, lower the diameter of the network, and to bring clients closer to the exit nodes, Unblock provides a mechanism for to users for adding random, untrusted shortcut links. This mechanism is necessarily limited, since any discovery mechanism may be abused by an adversary to discover the IP addresses of participants.

Existing systems such as SybilGuard and SybilLimit could be used to discover shortcut links, bounding the number of connections added to sybils from any particular user to be proportional to the number of attack edges. However, such a system could expose many more users in the process. Against a censor adversary, what is desired is a discovery method that bounds the number of nodes exposed to *moles* to be proportional to the number of attack edges.

In Unblock, when a node wants to find additional shortcut links, it initiates a shortcut discovery walk through a network of exclusively trusted social links, and returns a subset of nodes along the path. Figure 4.7 provides an example. New shortcut connections to these nodes are labeled as untrusted and are not used for subsequent walks. Shortcuts

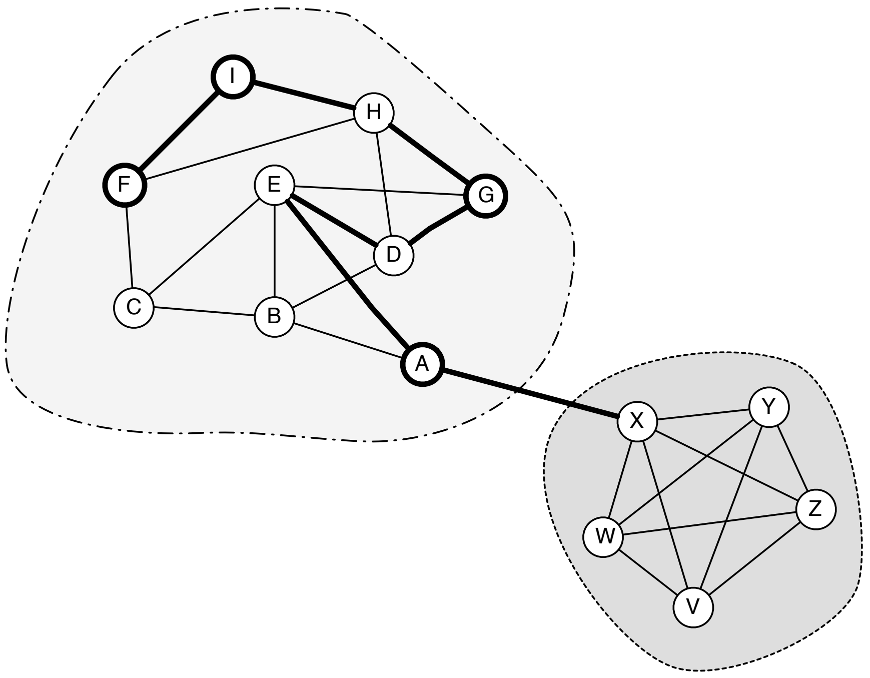

Figure 4.7: Example of a shortcut discovery walk to identify new shortcuts. Consider a walk initiated by X. It passes through A, E, D, G, H, I, and F. Nodes A, G, I, and F have degree less than $R_c = 4$, and return themselves as eligible shortcut candidates.

are used exclusively for routing data traffic. The shortcut discovery walk is designed in a way such that it returns the same deterministic set of nodes during an epoch, and tries to minimize changes between epochs.

An *epoch* defines a period of time, R_e, where the majority of users in the system have changed IP addresses. This period can be on the order of weeks or even months, depending on the underlying network. At the start of the epoch, each node will take a snapshot of its current trusted links, hash each with a local secret, and use these as the set of ID's in a consistent hashing [47] keyspace. The resulting keyspace is used to determine where to forward incoming walks. The outgoing link is chosen as the link preceding the incoming link in the keyspace. The keyspace is fixed for the duration of an epoch to ensure determinism within an epoch despite changes to the set of trusted links.

Each node makes a local boolean decision a_{il}, whether to expose itself to a shortcut discovery walk request. A node with degree greater than R_c will hide itself from all requests and simply forward the request to the next node. This parameter is chosen in order to protect high-value nodes in scale-free social networks. Previous work with friend-to-friend overlays have shown that capping the total connections to a user can improve aggregate bandwidth [45]. All other nodes will expose themselves with probability R_p and walks only proceed for a maximum of R_d steps. R_e, R_c, R_p and R_d are system-wide parameters. These shortcut relationships are timed out after a certain period of time.

When node x initiates a shortcut discovery walk along the path $i \in \{1, 2, 3 \ldots R_d\}$, it takes the following steps:

1. Node x will send the walk request along link l_x

2. Each node along the path will forward the request along a deterministic path, $\{l_1, l_2, l_3 \ldots l_{R_d}\}$

3. On the return path, each node, i, will include an entry in the walk result set if $a_{il} = \mathrm{True}$. An entry consists of the node's public key and IP address, signed by the node's private key.

In the midst of churn, these walks quickly become disconnected, preventing nodes from exploring their full set of shortcuts. This issue is avoided by caching partial shortcut results in the DHT in encrypted per-link mailboxes. When nodes come online, they can update these partial results.

The shortcut discovery method provides a number of useful properties:

- Local random decisions, l_i and a_{il} are seeded using only the local node ID, and do not depend on the hop count. Thus repeated queries to the same node do not reveal any additional information, even if those queries contain different TTL values or are initiated by nodes further downstream.

- Irrespective of which mole initiates the random walk, the set of nodes exposing their identities is fixed for a given attack edge. Thus, any censor's ability to find participants will be limited by the number of attack edges it can form, as well as the parameters R_d, and R_p.

- The random walk hides friendship information as long as R_p is chosen to be a relatively small value. A censor thus has limited ability to determine network topology using the shortcut discovery mechanism.

- The core of the network consisting of high-degree nodes are protected from being revealed, while focusing on improving path diversity from low-degree nodes

- Random walks tend to linger around areas of low connectivity. This property implies that shortcuts help the users who need it the most. This property is evaluated in Section 4.3

4.2.4 Accessing the Internet through exit nodes

Unblock uses exit nodes to create a bridge between the overlay network and the public Internet. Exit nodes can either provide global Internet connectivity or restrict transit to a small set of services. The user running the exit node is free to specify the exit policy of their node. Running an exit-node with global connectivity has been problematic for some Tor users[3], thus motivating a design with a more expressive policy. In Unblock, a user can opt to only provide access to certain domains (e.g., only access to wikipedia.org, twitter.com, google.com, and nothing else), thereby reducing the risk of abuse. Service providers that wish to improve access to their own site can run an "exit" node within their own network providing access to only their own service. Such restrictive policies are unlikely to cause trouble for the operator while still adding valuable capacity to the network. The hope is that many users will be willing to contribute bandwidth as long as the potential consequences are minimal.

[3]Law enforcement sometimes misattribute traffic from a Tor exit node to the owner of the node.

In order to contact an exit node, a peer must first know it exists. In Unblock, each exit node is identified by a public key, along with a recent announcement potentially signed by a Unblock directory service.[4] The announcement asserts which services are accessible from the node, and where the node is located (e.g., country or ISP domain). Signed announcements imply the signer has verified the claims in the announcement, which results in client preference towards those nodes, but announcements which are unverified can also participate; these exit nodes will however be used only when there are no routes to exit nodes with signed announcements.

Given the augmented overlay structure of Unblock, there are often a large number of possible paths to choose from when routing to an exit node. The goal of the routing protocol is to discover paths that: (a) minimize end-to-end latency, (b) discovers multiple parallel paths when available, and (c) is resilient to node failure/churn.

The approach is a hybrid path announce / on-demand path discovery protocol where an announcement from the exit node creates a minimum latency routing tree. The on-demand path discovery allows a node to find multiple branches in the tree, creating a set of alternative paths that can be used to communicate with the exit node. Figure 4.8 provides an example. The protocol maintains the routing tree using periodic refreshes. The discovery protocol is initiated on demand when a node initiates a connection to an exit node.

To route Internet traffic over Unblock the user configures their web browser to use a SOCKS5 proxy started by Unblock on the local machine. This proxy transparently monitors web-requests and tunnels them to the appropriate exit node. The policy for choosing which exit node to use for a particular target is determined by the the capability of the exit node, the overlay performance to the exit node, and the local policy on the client machine.

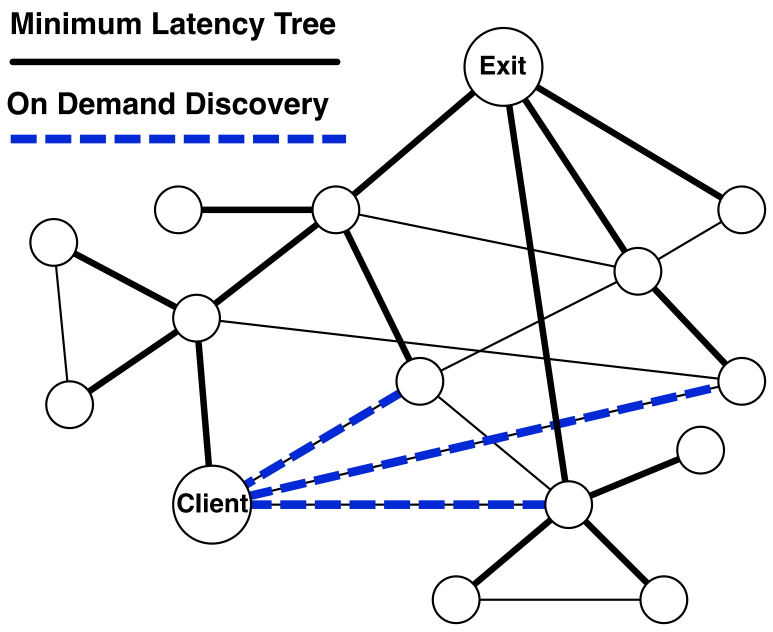

Figure 4.8: Example of how paths to exit nodes are found. Path announcements create a minimum latency routing tree. On-demand discovery finds alternate paths. In this example 3 additional paths are found.

Path announcements

Exit nodes announce their existence to the network. When nodes receive a previously not seen announcement they immediately forward the announcement to their neighbors. The return path of these announcements creates a minimum latency routing tree that is used when communicating the exit node. Announcements contain a timestamp, a nounce, hash of the public key of the exit node, optionally a set of exit node properties[5], a solution to a computational puzzle (more on this in Section 4.2.7), and a signature ensuring the announcement indeed is from the exit node.

[4]The purpose of the directory system is described in 4.2.7. The directory service is replicated, e.g., on PlanetLab nodes, to ensure higher availability and reachability.

[5]Such as region and domains reachable through the node.

On-demand multi-path discovery

While the minimum latency routing tree provides the lowest latency path to an exit node, link and path properties in Unblock can change rapidly. Instead of continuously trying to keep the routing tree up to date, Unblock instead makes the nodes capable of handling a slightly out-of-date routing tree. The approach relies on the fact that while a branch of the tree can become slow or even disconnected due to issues at an upstream node, other branches in the tree not containing that node are likely to be unaffected. When initiating a connection to an exit node the client will initiate a limited scope path discovery, asking its neighbors if they have an alternate path to the exit node. A path is acceptable if the neighbor's next hop is neither the current node nor the next hop of the current node. This process can be repeated by the neighbors if the number of found paths is low.

4.2.5 Transport protocol

Forwarding data across across the overlay has both performance and reliability implications that affect the design of the transport layer. First, forwarding traffic over a multi-hop overlay path limits the throughput to the slowest link even though there might be spare capacity at other intermediate nodes and along other paths. Second, path conditions could change either due to churn or temporary bursts of traffic through congested nodes. Third, traffic between pairs of overlay nodes can represent multiple independent circuits generated by different clients in the system. If these different flows are multiplexed on to a single reliable TCP connection between adjacent relays, the resulting interference could result in suboptimal performance for all flows traversing the link.[6]

The transport protocol in Unblock addresses the above set of issues using the following techniques. First, instead of using hop-by-hop TCP connections, Unblock use datagram based transport at each overlay hop and end-to-end congestion control across the entire

[6]Prior studies have diagnosed these issues in the context of Tor and proposed backwards-compatible fixes to Tor, while retaining the basic per-hop TCP transport and single path transfers [28, 4, 76].

overlay path. This minimizes the interference between flows that share the same overlay hop. Second, the payload is transported over multiple overlay paths to overcome bottlenecks and also handle churn in a seamless manner. The multi-path transport is also designed to optimize for latency in the case of short flows and maximize throughput for large flows. Third, the Unblock design borrows adaptive flow control mechanisms from the ATM literature to minimize queue build-ups that might arise from transient congestion or persistent disparities in capacities of overlay nodes.

End-to-end Congestion Control over Multiple Paths

The routing algorithm ideally yields multiple paths to a specific exit node. Data from the incoming stream is split into chunks, which are then transmitted across all available paths using UDP datagrams. The receiving endpoint assembles the packets and delivers it to the application in the correct order. Unblock handles congestion over end-to-end paths using a TCP style transfer window for each overlay path that is updated using the traditional additive increase multiplicative decrease mechanism upon packet losses over that path (as in MPTCP [101]).

Unblock also uses a redundancy mechanism to balance the goals of latency and throughput. Based on how much data has been transmitted, the sender will determine if the stream is a data-intensive, throughput-bound stream, or a bursty, latency bound stream. Initially, all transfers are assumed to be latency sensitive and messages will be duplicated and sent along multiple overlay paths. The amount of duplication is steadily reduced as more bytes are transferred over the end-to-end path. This balances the goal of minimizing latency when transmitting small pieces of content with the goal of using all of the available throughput for larger transfers.

Delay Based Adaptive Flow-control

Unblock operates in settings that could benefit from the circuit-based mechanisms used in ATM networks. More specifically, our environment has:

- *High end-to-end latencies.* Overlay paths span multiple hops, often spanning several continents. End-to-end congestion control responds to congestion over timescales of RTT, leading to slow ramp up and slow recovery from loss. Also, given bursty traffic, by the time an end-to-end protocol has responded to congestion, a large burst of packets can already be lost.

- *Diverse Link Characteristics.* Link bandwidth may vary by several orders of magnitude, from 1 Gbps fiber-to-the-home to 256 Kbps cable modems. Flows starting with a high bandwidth link will quickly fill the buffers of subsequent low bandwidth links.

While Unblock contains end-to-end congestion control compatible with multipath TCP, I augment that mechanism with a back-pressure based mechanism based on the N23 protocol used in ATM networks [51]. This mechanism minimizes queueing and eliminates packet loss on overlay nodes. Just as in credit-based flow-control for ATM networks, Unblock regulates the flow of data from upstream nodes using credits. Credit to send data to a downstream node is replenished through explicit control messages. When a node detects that a queue is building up, it stops issuing credits to upstream nodes, thus temporarily slowing or stopping the flow. In the N23 protocol, the amount of credit issued determines the balance between optimizing for latency and throughput. Lower values decrease worst case queueing but also lead to low utilization. An existing end-host congestion control protocol recommends a target queueing delay of 100 ms at the bottleneck link to strike the appropriate balance [85].

To maximize throughput it is important to fully utilize the bottleneck link, and for that reason Unblock allows up to 100 ms of queueing on that node. However, for non-bottleneck links it is neither required nor preferred to allow such lengthy queues.[7] Instead, intermediate nodes detect if they are non-bottleneck node and then chose a lower queueing target of 10 ms. Nodes can detect that they are non-bottleneck nodes when they are limited by credits rather than their own bandwidth.

4.2.6 Secure Connection Bootstrapping

The announce protocol creates a routing table at each node in the system. This table contains the next hop to each exit node. By running the DHT service only on exit nodes it is possible to plug in an existing DHT protocol such at Kademlia with the only modification being that the node_id is the public key hash of the exit node. No translation between node_ids and network addresses is required.

When a node wishes to query the DHT it can first look at its local routing table to find the exit nodes most adjacent in key-space to the desired key. There is no guarantee that the node knows about all the exit nodes[8] so in response to the query the node could either receive the result, or another node key that is closer in key space. When this happens the node will cache the results and iteratively repeat the query until the correct node is found.

Just as in OneSwarm each client is identified by a 1024 bit RSA key pair. This key is persistent, even when the IP address of the peer may change. At startup, a client will insert a copy of its current connection information into the internal DHT for each peer link. These copies will be encrypted with the public key of the remote node, and indexed into the DHT using a 20 byte, randomly generated shared secret, agreed upon during the first successful connection. Before the first successful connection the location information is stored at a location calculated as the hash of the remote node public key concatenated with the public key of the local node.

[7] In the worst case with nodes of monotonically decreasing capacities, the total queueing delay becomes $target\_latency * hop\_count$.

[8] This can happen immediately after startup for example.

4.2.7 Attacks and Defenses

In this section, I outline some of the potential attacks and ways in which they can be defended against. I consider scenarios where an attacker can join the network with a limited number of nodes, monitor network traffic to/from its nodes, and generate, modify, and delete Unblock overlay messages flowing through its nodes.

As discussed earlier, the network augmentation mechanism limits the exposure of relay addresses to k per attack edge. Further, the design of the DHT that allows for DHT operations to be performed using just the overlay topology and the encrypted storage of node location information in the DHT prevents the leakage of additional information regarding relays participating in the system.

An important property of the the Unblock protocol is a resilience to adversaries claiming to offer exit capabilities. The two mechanisms a malicious exit node could leverage to attack the system are: (1) flooding announcements to overwhelm the system, and (2) black-holing received traffic. Unblock mitigates these attacks through the certification mechanism. Nodes in the system will only forward exit node announcements if they are signed by a trusted directory service. This property allows a directory to throttle the total rate of exit node announcements on the network, and verify the functionality of exit nodes before signing proposed announcements.

I now briefly describe the certification process. An exit node will request certification either directly from the Unblock directory service, or through the Unblock overlay if the directory is blocked. In the common case where direct connections are functional, the directory will automatically sign one announcement request per unique IP address in each announcement period. In order to support certification in cases where the exit node cannot be identified directly, the directory will also provide a service to certify announcements through the overlay itself. This service will make use of computational puzzles in order to limit an adversary's ability to perform a Sybil attack wherein a single adversarial exit node makes multiple advertisements and thereby attracts a disproportionately high fraction of the system traffic.

The use of a minimum latency tree also prevents attacks where an adversary tries to position itself in as many overlay paths as possible. Note that exit node announcements are forwarded as soon as they are received by nodes in the system, so an overlay path would traverse an adversary only if it is in the minimum latency tree. This prevents traffic attraction attacks which might be possible if Unblock had used a more traditional distance vector style protocol for computing routes to exit nodes.

Finally, Unblock guards against eavesdropping by intermediate nodes using end-to-end encryption between clients and exit nodes in the system. This however provides somewhat weak guarantees, as the traffic might be routed through as few as two intermediates, namely the node that neighbors the requesting client and the exit node itself. For users desiring stronger anonymity properties, Unblock could allow back-to-back traversal of overlay routes, so that traffic upon reaching an exit node is routed back through the overlay to a different exit node, before it is conveyed to the actual destination.

4.3 Evaluation

The implementation of Unblock is constructed as an experimental extension to OneSwarm. The Unblock extension has not however been released publicly, nor do I have access to the social network topology of the user base, so I limit the evaluation results to controlled testbed settings.

I begin by evaluating the performance of the transport layer implementation using a multi-hop test framework in PlanetLab. In order to demonstrate the advantages of Unblock's design and practicality in forwarding web traffic, I compare its performance to standard transport mechanisms used in systems such as Tor.

Additional evaluation is done in a simulator to evaluate the security and performance properties of Unblock at scale. I begin by analyzing the impact of untrusted shortcut links on the network topology. Compared to the baseline in Section 4.1, shortcut links are shown

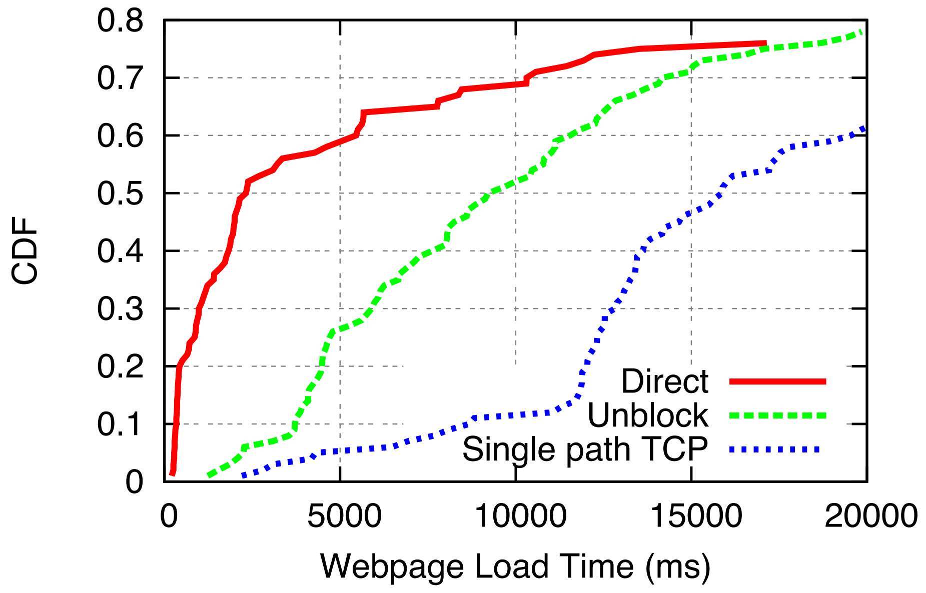

Figure 4.9: Web page load time across the Unblock overlay. Unblock represents load times across a three hop overlay using the optimized Unblock transport protocol. Single path TCP shows baseline load times for the same topology using per-hop TCP over a single overlay path.

to maintain connectivity to a significantly higher number of nodes during churn. I then explore the trade-off between better availability and risk of disruption of service by a censor adversary.

4.3.1 Transport Performance

I evaluated the performance of our transport by inserting the Unblock overlay as a relay to a SOCKS proxy. The PhantomJS headless webkit browser was used to measure page load times of popular websites. Much of the time spent rendering a page comes from dependent resources, making network latency more important than many systems acknowledge. Pages were loaded from the same set of domains as Figure 4.6.

This set of experiments demonstrates the importance of lowering latency in order to efficiently handle the small, bursty traffic associated with web requests. Figure 4.9 shows that Unblock has a fairly constant 2-5 second page load penalty compared with loading

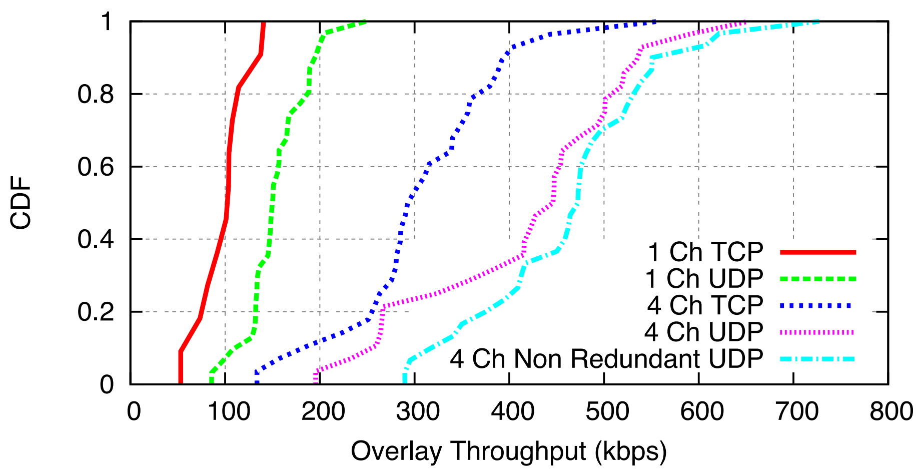

Figure 4.10: Throughput performance of Unblock. Non Redundant represents possible throughput when packets are only sent once, at the cost of higher latency. UDP performance improves with more paths until either the client or source is bandwidth limited.

pages directly. I attribute part of this penalty to the decision to develop the overlay purely at the socket level. By implementing a local SOCKS5 proxy, as the Tor system does, Unblock would be able to bundle multiple web requests within the same set of paths, rather than suffering a full round trip time of latency to establish paths before beginning each request. The use of UDP, the ability to take advantage of multiple channels, and the credit-based flow control already provides a significantly less variable and lower latency service than the baseline transport that uses per-hop TCP connections over a single overlay path.

Next I consider microbenchmarks that allows examination of the performance and latency enhancements made possible by different versions of the transport layer. Performance was evaluated using PlanetLab nodes located across the US. In all trials, the topology consisted of four disjoint paths from client to server, each with three hops. All nodes were selected randomly from the available pool, with nodes reselected between each trial. Each node had a bandwidth rate limit of 1Mbps. Figure 4.10 shows the observed throughput achieved with the various transport improvements: Transferring data using an encrypted UDP transport, transferring data concurrently over multiple paths, and dynamic use of redundant packet transmissions. Throughput is measured as the time required to

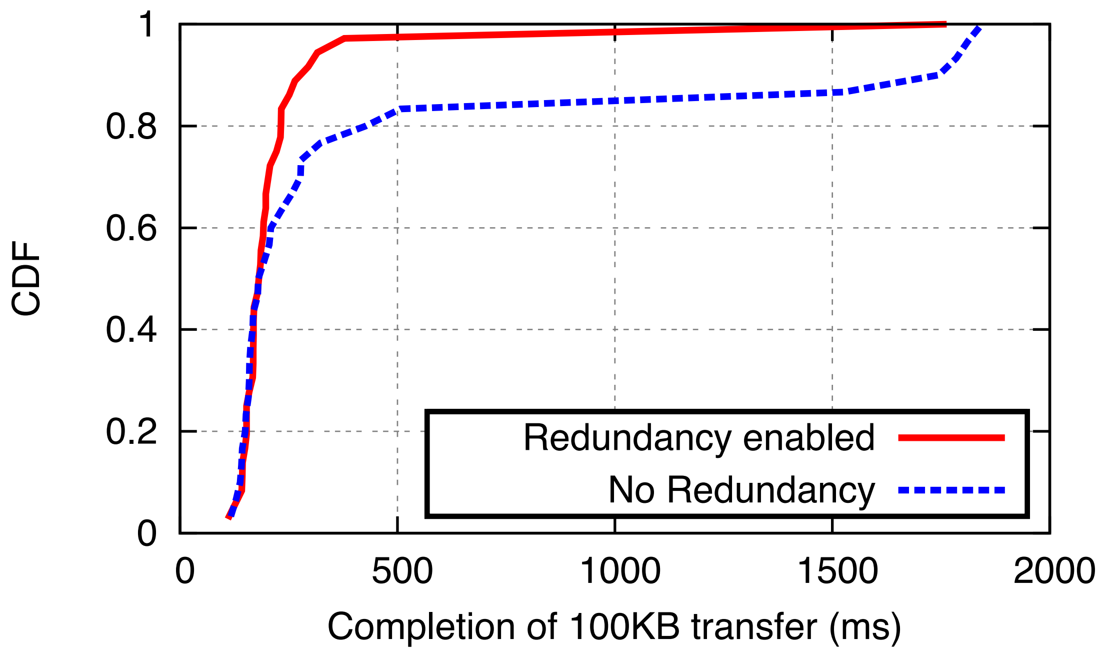

Figure 4.11: Latency performance within the overlay. With redundant transmissions, latency suffers less from the presence of slow or flaky paths.

transmit one megabyte of data. Using multiple paths with UDP improves throughput linearly until three paths, where bandwidth of either the source or destination limits its ability to transmit or receive more. I also examine the throughput of multi-path flows that do not perform any redundant transmissions in order to characterize the capacity lost due to redundancy; this scheme provides only a marginal increase in throughput indicating that the cost of redundant transmissions is low.

Figure 4.11 provides microbenchmark results that evaluate the use of redundant transmissions. It shows measurements of the transmission time for a 100 kilobyte flow across the same topology as the other experiments, with and without the adaptive use of redundant transmissions. While most links in the testbed had robust performance characteristics, when slow or flakey links were encountered, redundant transmissions were able to maintain a low latency connection by mitigating retransmissions and in-order delivery delays.

4.3.2 Simulation Results

The following experiments are performed on a simulator in order to measure the implications of our design decisions at scale. Using the simulator, I find that the shortcut discovery protocol effectively improves the connectivity to any particular exit node in the face of churn, while restricting the number of honest users that are exposed to a censor's *moles* in an attack. Even with a strong model adversary that can block 50% of the edges in the network, shortcuts effectively improve connectivity.

These measurements were performed using simulated networks based on the datasets collected by [62]. For some of these datasets, as in the Youtube social network, I was able to obtain the geographical location of the user. In such cases a latency between users is added using predictions from *iPlane* [58]. Exit nodes and moles are chosen at random from available nodes in the graph. The evaluation was repeated with varying churn, wherein the node uptimes and downtimes are modeled using Poisson distributions. Lastly, shortcuts are only created between nodes that have degree less than 50. This restriction protects high-degree nodes from being overloaded and restricts disclosure of high-value nodes to censors.

Figure 4.12 shows the improvement in the availability of paths to exit nodes as we add untrusted links to the underlying social network for the Youtube dataset.[9] In this experiment, 10% of the nodes were set to be exit nodes. The experiment was repeated for a range of node uptime values. For each value of expected node uptime fraction `f`, the number of untrusted links discovered by the shortcut mechanism is set to be `2/f`. This parameter setting implies that each active node, in expectation, will have two untrusted links to other active nodes in the system. The results show that the additional untrusted links can dramatically improve the availability of paths to exit nodes, especially when the node uptime fraction is low (as is the case with most peer-to-peer systems [89, 37, 78, 55]).

[9]The Youtube social network comprises of about a million users. The same analysis was performed on both smaller and larger social networks (e.g., the Foursquare network with about hundred thousand users and the LiveJournal network with about five million users) and obtained results that were qualitatively similar.

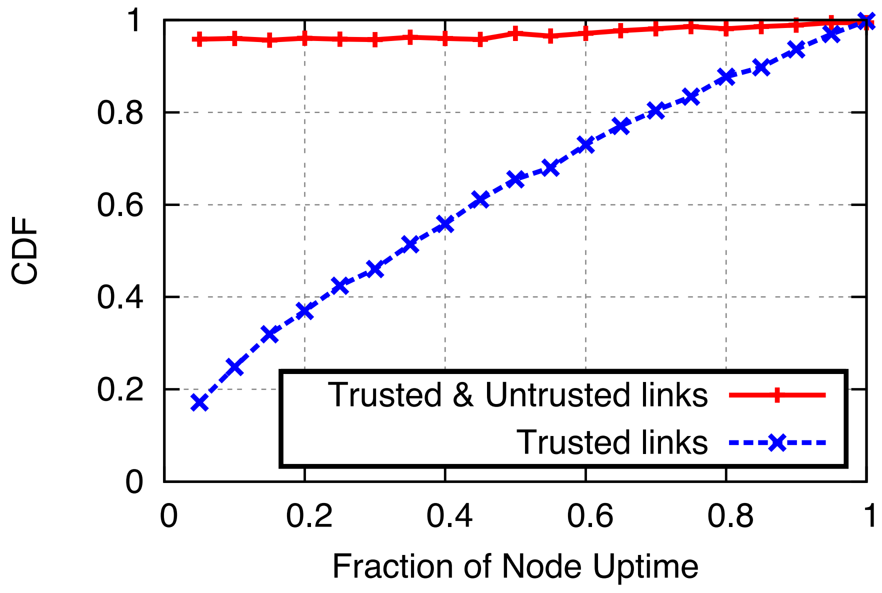

Figure 4.12: Fraction of nodes with paths to exit nodes on the Youtube social network dataset for varying node uptimes and with 10% of the nodes being exit nodes.

Figure 4.13 shows the CDF of latencies to any available exit node when nodes are online for 50% of the time. I examine this with and without untrusted links, and observe that the use of untrusted links also significantly lowers latency. I also model a strong adversary that monitors exposed shortcut nodes from 10000 moles in the network. The censor also has the power to block 50% of a node's links if exposed to its moles. As expected, I found a linear relationship between the number of attack edges and the number of honest nodes exposed to moles. More importantly, Figure 4.13 shows that there is minimal degradation in both connectivity and performance as a consequence of having the strong adversary.

4.4 Summary

Desire for uncensored Internet access has motivated the development of tools for censorship circumvention. However, most popular tools are not themselves censorship resistant and instead found themselves censored. This failing has prompted me to develop Unblock, a system designed to both provide uncensored communication, and resist censorship itself.

Through large scale simulations and measurements of a prototype implementation de-

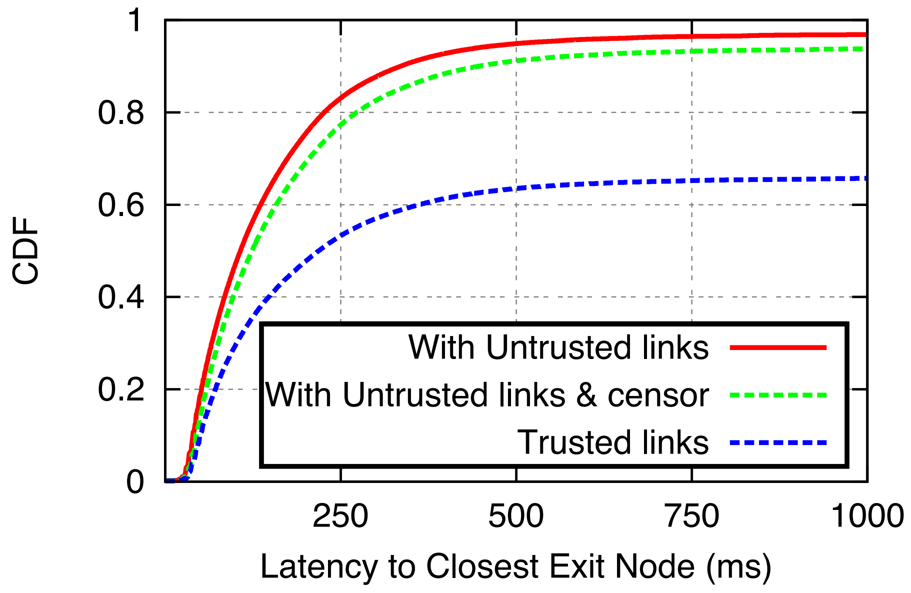

Figure 4.13: Impact of untrusted links on latency to exit nodes when 50% of users are online.

ployed on PlanetLab I show that Unblock can provide users access to blocked Internet services through a social trust based overlay. I believe the ideas behind Unblock will allow it to improve upon both the privacy and performance relative to existing solutions.

Chapter 5
RELATED WORK

Providing privacy and anonymity for Internet data transfers is a longstanding goal of the research community, and I draw on many existing ideas in the design of OneSwarm and Unblock.

5.1 Anonymous communications

There are two well known techniques for achieving anonymous communication. One approach is to interpose a third party (often referred to as a proxy) between the source and destination to hide the source's identity from the destination. For instance, Anonymizer provides anonymization services commercially, providing a centralized service that relays web traffic [6]. A proxy, however, allows a single entity to learn the identities of both the source and the destination. This lead to the development of schemes that convey traffic through multiple intermediaries. Crowds [77], provides anonymous web browsing by randomly tunneling requests via other system participants; an intermediary node either choose a random successor relay or simply submit the request to the destination. Tor [27] leaves the choice of relays to the source. An issue with recruiting volunteer relays is that the malicious activity emanating from exit nodes is often attributed to their hosting organizations, discouraging users from hosting exit nodes. I see this as a policy rather than technical issue, and mitigate it by distributing a default set of whitelisted services with whom exit nodes will relay traffic.

An other approach to achieving anonymous communications is to use mix nets. Relaying electronic messages through intermediaries to obscure the source and destination from a global eavesdropper was first proposed for anonymous email by Chaum [16]. In his scheme, the mix is an enhanced proxy that collects encrypted messages of equal length from senders, decrypts the messages using its private key, and forwards the messages to the appropriate recipients in a different order such that an eavesdropper cannot determine which output messages correspond to which input messages. A sequence of mixes provides stronger unlinkability properties and is robust to colluding mixes. More recent work has shown that mix nets functionality can be achieved without a public key infrastructure [49]. However, the message transmission times for mix-net based systems are typically on the order of several minutes or longer. This makes them unsuitable for use in applications involving interactive traffic, such as web browsing and VoIP, though they suffice for other services such as email making them complementary to Unblock and OneSwarm.

5.2 Censorship resistance

Naturally, anonymizing solutions have been adapted to achieve censorship resistance [75, 19]. A key stumbling block though is that most proxy-based anonymizing solutions are not membership concealing. An adversary interested in enforcing censorship can then infiltrate the anonymizing system, learn the identities of the proxies, and block all communications to them. The Tor developers recognized this challenge [92] and have proposed mechanisms to distribute a small subset of relays to each client using out-of-band communication channels [25], but adversaries can subvert such techniques [26]. Feamster et al. [33] propose a system that restricts the set of proxies known to each client based on the client's IP address, but their system cannot defend against an adversary who controls blocks of IP address space or a large set of IP-addresses, e.g. through a botnet.

Another piece of related work is Infranet [59], which uses stenographic techniques to tunnel sensitive requests through innocuous looking web requests. A gateway at the service provider side, upon detecting an Infranet client, sends back critical information hidden in images. By hiding information in unencrypted web traffic, Infranet works even if the censor blocks encrypted traffic and performs deep packet inspection on unencrypted traffic, However, it requires strong identities and a large amount of cover traffic to convey small bits of sensitive data.

Censorship resistant publishing can be achieved by pushing data onto nodes willing to serve as storage sites for public data [20, 97, 98]. It is worth noting that these systems face a different set of challenges, namely the ability to support anonymous publication, providing durability, and preserving the integrity of published data. These systems are not designed to provide interactive or non-interactive communication between a source and a destination across censorship domains. Neither OneSwarm or Unblock provides any mechanism for persisting user data on other user's machines, instead the overlay is used solely for connecting hosts with clients in an privacy preserving manner.

An alternative to overlay solutions running on end-hosts is to build anti-censorship technology into the network. Examples of this include Decoy Routing [48], Telex [103], and Cirripede [40]. Instead of contacting the proxy server directly using the IP address of a proxy, these systems instrument core Internet routers to detect special patterns in packets, and redirect matching flows flows to a proxy. The advantage with this scheme is that the actual IP destination can be any address as long as the path to the destination intersects with an instrumented router. The main disadvantage with this solution is that is requires cooperation from large ISPs. I believe there is value in both pursuing approaches that relies on private individuals contributing resources as well as those that require large corporations to act. My work is focuses on tools for private individuals.

5.3 Social networks

Incorporating real-world trust relationships has been a crucial design element in several recently proposed systems. Friendstore [94] is a P2P backup system where users store backup data only on other trusted nodes owned by friends or colleagues. In Ostra [63], the scarcity of social connections is used to combat spam. UIA [34] provides data routing and name resolution over a socially constructed overlay of personal devices. Johnson and

Syverson have recently described how to include preexisting trust when choosing paths in Tor [90]. Turtle [73] is a file-sharing application that limits direct communication to only the social graph in an attempt to circumvent eavesdropping. Waste [99] allows users to create password-protected communities for secure chat and file sharing. Once invited to a community, users can discover other members, connect to them, send them messages, and browse their shared files. Baden et al. have applied cryptographic techniques to enable data sharing with permissions in current social web services without exposing content to service providers [9].

5.4 Sybil defenses

Many defenses have been proposed to combat Sybil attacks. These include strong identities minted by a logically centralized authority [29], computational puzzles and bandwidth contributions to make peers prove that they are not Sybils [12], and leveraging social networks [106, 54]. Defenses based on social networks, such as SybilGuard [106] and SybilLimit [105], might seem appropriate for Unblock as they limit the creation of trust relations to unknown identities unless they have valid social network properties. Their techniques are insufficient for the threat model I consider because they do not provide any mechanisms for concealing the membership of the social network and because they provide weak bounds on the number of trust links created to Sybils by the network as a whole. In contrast, the mechanism for finding untrusted peers in Unblock does not reveal the identity of any users except the specific ones that will be connected to.

5.5 Sharing workload

There is a large body of work measuring the properties of peer-to-peer networks [1, 37, 69] and file sharing sites [14]. I conducted new experiments because previous studies did not show how content interest relates to the social network of the participants. By measuring the content interest as well as social network of last.fm users I am able to relate these factors allowing more realistic assumptions of how sharing in a social overlay occurs. These measurements allowed for a more accurate simulation based evaluation of the OneSwarm protocol.

Chapter 6
CONCLUSIONS AND FUTURE WORK

In this chapter I describe the contributions made by the dissertation and how they support the thesis. I then give an overview of future research directions in the area, and end with a short summary of the dissertation in Section 6.4.

6.1 Contributions

This dissertation demonstrates the following thesis: *By building overlay networks based on social trust we can improve security and performance relative to existing solutions.*

The work made the following contributions:

- **The design, implementation, and evaluation of a privacy preserving file distribution protocol for social network overlays.** I provide the design, evaluation, and implementation of OneSwarm, a file sharing protocol designed for high performance privacy preserving data sharing in a social overlay network. The protocol gives users complete control over their data, and allow them to share data only with friends and family, or with everyone, while still preserving their privacy. I have evaluated the protocol both in the wild and in a simulator to evaluate the system at large scale. The results show that the protocol protects user privacy while maintaining performance.

- **A software artifact, OneSwarm.** The OneSwarm client provides an open-source reference implementation of the OneSwarm protocol. The software binary as well as related source code is available to the public.

- **The design, implementation, and evaluation of a censorship resistant system for social network overlays.** I have designed, implemented, and evaluated Unblock, a protocol that can assist users when bypassing Internet censorship. Measurements of the implementation and simulations of the system at large scale shows that social network overlays can get around a censor while at the same time improve performance relative to public overlays based on onion routing.

6.2 Key ideas

Both OneSwarm and Unblock benefit from a common set of design techniques.

- **Base overlay topology on social network:** By basing the overlay topology on the social network of the users it is possible for the overlay to rely on the trust users have in each other. In the case of OneSwarm the system relies on trusted friends to not reveal the source of data when forwarding requests. In the case of Unblock users trust their friends to not expose their network address to the censor. By relying on this assumption is it possible to simplify protocol and improve performance.

- **Augment social graph with untrusted links:** Restricting the overlay to only contain trusted links makes it hard for new users to join the network and provides poor performance and availability for users with few trusted links. For these users is it critical to provide an avenue to increase their connectivity. Untrusted links allows new users to connect to other users that they necessarily do not trust. This mechanism increases their exposure to attackers, but the increased connectivity with the overlay improves performance. Users can adjust the privacy/performance tradeoff to suit their needs. Experience from the OneSwarm user community as well as simulation and real world experiments has shown that untrusted links provide a useful tradeoff, especially for new users.

- **Use multiple paths:** Basing the overlay topology on the social network of the users means that the system must depend on multi-hop paths through the overlay. Measurements of the OneSwarm overlay show that while the performance of each individual path is limited, the aggregate performance of multiple paths provide throughput similar to a direct connection. Using multiple paths is critical for good performance.

- **Provide flexibility with regards to users varying need for privacy:** Feedback from the OneSwarm user community showed that different users have different goals for the system. Some users add a large number of untrusted links, even though that increases the probability of exposure to attackers. For these users improved performance is more important than minimizing the likelihood of exposure. Other users restrict their connectivity to just a few close friends. These users have no exposure to an attacker as long as their friends remain honest.

  A similar design principle influenced the design of Unblock. Different countries have varying levels of censorship. In some countries the censorship technology is so primitive that simply detouring traffic through another country is sufficient. In other countries the censor actively tries to discover and block cross country overlay links.

- **Support network mobility:** Once two users have created a trust link they expect to be connected when both users are online. Some engineering is required to provide seamless support for dynamic IP addresses and mobility. Both OneSwarm and Unblock rely on a DHT to rendezvous between users. In OneSwarm a public DHT is used. In Unblock an overlay internal DHT is used as a public DHT can be easily blocked by a censor. Allowing users to "automatically" find each other without the need to manually enter IP addresses and ports decreases user friction and improves the connectivity of the network.

6.3 Future work

The work presented in in this dissertation opens up several areas of future work.

6.3.1 *Protecting OneSwarm users against malicious ISPs*

The threat model OneSwarm operates under assumes that the users local ISP will not collude with the attacker. However, projecting forward, it is quite possible that ISPs take a more active role in monitoring their networks. There are some obvious traffic pattern attacks an ISP can launch against OneSwarm users. Modifying the OneSwarm protocol to be robust against malicious local ISPs would be needed if this reality becomes true.

6.3.2 *Contribution incentives for anonymous systems*

Maintaining user privacy while at the same time rewarding user contributions is a challenging task. In OneSwarm the incentive system is based on local information about directly connected peers. Peers that maintain a positive upload to download ratio get preferential service during times of congestion. However, forwarding data is not necessarily rewarded as it merely shifts the balance from one peer connection to the next. Rewarding users for forwarding data would be good, but neighbors do not know the final source or destination of a transfer so it is hard to know if data is forwarded or not. As show in Section 3.5 the current OneSwarm overlay does not suffer from lack of forwarding capacity, but in a future where Internet service providers charge for data usage or enforce bandwidth caps it is possible that users will decrease the capacity they are willing to contribute. In such a future providing users with incentives for forwarding data would be required.

6.3.3 *Global deployment of Unblock*

The prototype implementation of Unblock allowed me to evaluate the feasibility of the design. Since there are no published anti-censorship social network overlays I had to rely on measurements of other social networks for the evaluation. To understand how these types of networks evolve and are used we must perform measurements of an existing deployment. Deploying Unblock in censored regions would make it possible to better understand how the system operates in the wild.

6.3.4 Directly hosting content in Unblock

The routing protocol in Unblock scales really well with the number of users, but scales poorly with the number of exit nodes. The current assumption is that the number of users will be significantly larger than the number of exit nodes, so announcing each exit node to each users does not impose a large burden. However, it is also the case the Internet services have a performance incentive to directly host their service within the overlay. The simplest way to do this is for the service to run its own exit node, where the exit policy restricts communication to just the service in question. If this behavior becomes common place the overhead of the exit node announcements could become significant and an alternative protocol would have to be deployed.

6.4 Summary

The core Internet protocols were not designed to protect the privacy of the participants of communication. Unfortunately, this choice has led to censorship and surveillance becoming increasingly common on the Internet today. The same problem persists at higher level protocols, for example: popular peer-to-peer networks are trivial to monitor even for an adversary with limited resources.

This dissertation describes the design, implementation, and evaluation of two systems that can bring us towards a future with less, not more, censorship and surveillance. OneSwarm is a privacy-preserving data sharing network designed to give users performance comparable to widely used peer-to-peer networks without exposing user behavior to third party surveillance. Unblock is a overlay network that expands on OneSwarm but is designed to circumvent censorship of general-purpose Internet services. Common to both systems is the use of existing social trust between participants to thwart surveillance and censorship. These systems are designed to run on today's Internet with no changes to core Internet infrastructure and protocols. Measurements of the systems in the wild, and simulations of their behavior at scale, show that they protect user privacy and improve performance over existing alternatives.

BIBLIOGRAPHY

[1] Eytan Adar and Bernardo Huberman. Free riding on Gnutella. *First Monday*, 2000.

[2] Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong. Analysis of topological characteristics of huge online social networking services. In *Proc. of WWW*, 2007.

[3] Alexa top 500 global sites, 2012. http://www.alexa.com/topsites.

[4] Mashael AlSabah, Kevin Bauer, Ian Goldberg, Dirk Grunwald, Damon McCoy, Stefan Savage, and Geoffrey Voelker. Defenestrator: Throwing out windows in tor. In *Proc. of PETS*, 2011.

[5] Julia Angwin. The web's new gold mine: Your secrets. http://online.wsj.com/article/SB10001424052748703940904575395073512989404.html.

[6] Anonymizer. http://anonymizer.com.

[7] Jacob Appelbaum. Technical analysis of the ultrasurf proxying software. https://media.torproject.org/misc/2012-04-16-ultrasurf-analysis.pdf.

[8] Robert Atkinson, Stephen Ezell, Scott Andes, Daniel Castro, and Richard Bennett. The Internet Economy 25 Years After .com. *The Information Technology and Innovation Foundation*, 2010.

[9] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: An online social network with user-defined privacy. In *Proc. of SIGCOMM*, 2009.

[10] Beijing orders new controls on 'weibo' microblogs. *BBC News*, December 2011.

[11] bluekai. The bluekai registry - putting consumers in control of their digital footprint. http://www.bluekai.com/registry/.

[12] Nikita Borisov. Computational puzzles as Sybil defenses. In *Proc. of the Intl. Conference on Peer-to-Peer Computing*, 2006.

[13] Tania Branigan. China boosts internet surveillance. http://www.guardian.co.uk/world/2011/jul/26/china-boosts-internet-surveillance.

[14] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I Tube, You Tube, Everybody Tubes: Analyzing the world's largest user generated content video system. In *IMC*, 2007.

[15] Pratap Chatterjee. The state of surveillance. http://www.guardian.co.uk /world/2011/jul/13/iran-tightens-online-censorship.

[16] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 1981.

[17] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making Gnutella-like P2P Systems Scalable. In *Proc. of SIGCOMM*, 2003.

[18] Shirong Chen. China tightens internet censorship controls. *BBC News*, May 2011.

[19] Citizens Lab. Everyone's guide to bypassing internet censorship. http://deibert.citizenlab.org/Circ_guide.pdf.

[20] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *Proc. of PET*, 2001.

[21] Aaron Clauset, Cosma Shalizi, and MEJ Newman. Power-law distributions in empirical data. http://arxiv.org/abs/0706.1062, 2007.

[22] Bram Cohen. Incentives build robustness in BitTorrent. *Proc. of P2PEcon*, 2003.

[23] Congressional Executive Commission on China, Annual Report 2011. 2011.

[24] George Danezis. An anomaly-based censorship-detection system for tor. August 2011.

[25] R. Dingledine and N. Mathewson. Design of a blocking-resistant anonymity system. https://svn.torproject.org/svn/projects/design-paper/blocking.pdf.

[26] Roger Dingledine and Jacob Appelbaum. How governments have tried to block tor [28c3]. http://www.youtube.com/watch?v=GwMr8Xl7JMQ.

[27] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *USENIX Sec.*, 2004.

[28] Roger Dingledine and Steven Murdoch. Why Tor is slow and what we're going to do about it. http://blog.torproject.org, 2009.

[29] John R. Douceur. The Sybil Attack. In *Intl. Workshop on Peer-to-Peer Systems*, 2002.

[30] Matthew Edman and Paul Syverson. AS-awareness in Tor path selection. In *Proc. of CCS*, 2009.

[31] Ernesto. Anti-piracy lawyers find cheaper way to identify bittorrent users. http://torrentfreak.com/anti-piracy-lawyers-find-cheaper-way-to-identify-bittorrent-users-110722/.

[32] Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Profiling a million user DHT. In *Proc. of IMC*, 2007.

[33] Nick Feamster, Magdalena Balazinska, Winston Wang, Hari Balakrishnan, and David Karger. Thwarting web censorship with untrusted messenger discovery. In *Proc. of PETS*, 2003.

[34] Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas, Sean Rhea, Frans Kaashoek, and Robert Morris. Persistent personal names for globally connected mobile devices. In *Proc. of OSDI*, 2006.

[35] Electronic Frontier Foundation. Nsa spying. https://www.eff.org/issues/nsa-spying.

[36] Freegate. https://simurghesabz.net/.

[37] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of SOSP*, 2003.

[38] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. Measurements, analysis, and modeling of BitTorrent-like systems. *Proc. of IMC*, 2005.

[39] James Hamilton. The cost of latency. http://perspectives.mvdirona.com/2009/10/31/TheCostOfLatency.aspx.

[40] Amir Houmansadr, Giang T.K. Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: circumvention infrastructure using router redirection with plausible deniability. In *Proc. of CCS*, 2011.

[41] How to bypass internet censorship. https://www.howtobypassinternetcensorship.org.

[42] OpenNet Initiative. Country profiles. http://opennet.net/research/profiles.

[43] Ipredator. https://www.ipredator.se/.

[44] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Leveraging BitTorrent for end host measurements. In *Proc. of PAM*, 2007.

[45] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Privacy-Preserving P2P Data Sharing with OneSwarm. In *Proc. of SIGCOMM*, 2010.

[46] Tomas Isdal, Will Scott, Raymond Cheng, Arvind Krishnamurthy, and Thomas Anderson. Toward blocking-resistant network services. In *Under submission*, 2012.

[47] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC*, 1997.

[48] Josh Karlin, Daniel Ellard, Alden W Jackson, Christine E Jones, Greg Lauer, David P Mankins, and W Timothy Strayer. Decoy routing: Toward unblockable internet communication. In *USENIX Workshop on Free and Open Comm. on the Internet*, 2011.

[49] Sachin Katti, Dina Katabi, and Katarzyna Puchala. Slicing the onion: Anonymous routing without PKI. *Proc. of HotNets*, 2005.

[50] Sanja Kelly and Sarah Cook. Freedom on the net 2011: A global assessment of internet and digital media. *Freedom House*, 2011.

[51] H. T. Kung, Trevor Blackwell, and Alan Chapman. Credit-based flow control for atm networks: credit update protocol, adaptive credit allocation and statistical multiplexing. In *Proc. of SIGCOMM*, 1994.

[52] last.fm. *http://last.fm*.

[53] Stevens Le Blond, Arnaud Legout, Fabrice Lefessant, Walid Dabbous, and Mohamed Ali Kaafar. Spying the world from your laptop: identifying and profiling content providers and big downloaders in BitTorrent. In *Proc. of LEET*, 2010.

[54] Chris Lesniewski-Lass and M. Frans Kaashoek. Whanaungatanga: Sybil-proof distributed hash table. In *Proc. of NSDI*, 2010.

[55] J. Li, J. Stribling, R. Morris, F. Kaashoek, and T. Gil. A performance vs. cost framework for evaluating dht design tradeoffs under churn. *Proc. of IEEE Infocom*, 2005.

[56] Vincent Liu, Seungyeop Han, Arvind Krishnamurthy, and Thomas Anderson. Tor instead of ip. In *Proc. of HotNets*, 2011.

[57] Oliver Luft. Wikileaks taken offline after publishing australia's banned websites. http://www.guardian.co.uk/media/pda/2009/mar/19/wikileaks-banned-australian-websites.

[58] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An Information Plane for Distributed Services. In *OSDI*, 2006.

[59] Nick Feamster Magdalena, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. Infranet: Circumventing web censorship and surveillance. In *In Proc. of USENIX Security*, 2002.

[60] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of IPTPS*, 2002.

[61] Alan Mislove, Hema Swetha Koppula, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the flickr social network. In *Proc. of WOSN*, 2008.

[62] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of IMC*, 2007.

[63] Alan Mislove, Ansley Post, Peter Druschel, and Krishna P. Gummadi. Ostra: leveraging trust to thwart unwanted communication. In *Proc. of NSDI*, 2008.

[64] The opennet initiative global internet filtering map. http://map.opennet.net/.

[65] Lasse Overlier and Paul Syverson. Locating hidden servers. In *IEEE Symposium on Security and Privacy*, 2006.

[66] Lasse Øverlier and Paul Syverson. Improving efficiency and simplicity of Tor circuit establishment and hidden services. In *Proc. of PET*, 2007.

[67] European Parliament. Directive 2006/24/ec. http://eur-lex.europa.eu/LexUriServ /LexUriServ.do?uri=CELEX:32006L0024:EN:NOT.

[68] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 2003.

[69] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in BitTorrent? *Proc. of NSDI*, 2007.

[70] Michael Piatek, Tomas Isdal, Arvind Krishnamurthy, and Thomas Anderson. One hop reputations for peer to peer file sharing workloads. In *NSDI*, 2008.

[71] Michael Piatek, Tadayoshi Kohno, and Arvind Krishnamurthy. Challenges and directions for monitoring P2P file sharing networks –or– Why my printer received a DMCA takedown notice. In *Proc. of HotSec*, 2008.

[72] Michael Piatek, Harsha V. Madhyastha, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Pitfalls for ISP-friendly P2P Design. In *Proc. of HotNets*, 2009.

[73] Bogdan C Popescu, Bruno Crispo, and Andrew S Tanenbaum. Safe and private data sharing with Turtle: Friends team-up and beat the system. In *Proc. of Intl. Workshop. on Sec. Prot.*, 2004.

[74] Swagatika Prusty, Brian Neil Levine, and Marc Liberatore. Forensic investigation of the oneswarm anonymous filesharing system. In *Proc. of ACM CCS*, pages 201–214. ACM, 2011.

[75] Psiphon. http://psiphone.civisec.org.

[76] Joel Reardon and Ian Goldberg. Improving tor using a TCP-over-DTLS tunnel. In *Proc. of USENIX security*, 2009.

[77] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Trans. Inf. Syst. Secur.*, 1998.

[78] RFC 4981. http://www.faqs.org/rfcs/rfc4981.html.

[79] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *Proc. of NSDI*, 2012.

[80] Salvatore Scellato and Cecilia Mascolo and Mirco Musolesi and Vito Latora. Distance Matters: Geo-spacial Metrics for Online Social Networks. In *Proc. of WOSN*, 2010.

[81] Sandvine. Global internet phenomena report: Fall 2011. http://www.sandvine.com/news/pr_detail.asp?ID=340.

[82] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP congestion control with a misbehaving receiver. *Proc. of SIGCOMM Comput. Commun. Rev.*, 1999.

[83] Maximilian Schrems. Facebook's data pool. http://europe-v-facebook.org/EN/Data_Pool/data_pool.html.

[84] Somini Sengupta. Group says it has new evidence of cisco's misdeeds in china. *The New York Times*, September 2011.

[85] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low extra delay background transport (ledbat). In *IETF Internet Draft*, 2006.

[86] Georgos Siganos, Josep M. Pujol, and Pablo Rodriguez. Monitoring the Bittorrent Monitors: A Birds Eye View. In *PAM*, 2009.

[87] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, 2001.

[88] W Ph Stol, H K W Kaspersen, J Kerstens, E R Leukfeldt, and A R Lodder. Governmental filtering of websites: The dutch case. *4th Annual Giganet Symposium*, 2009.

[89] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proc. of IMC*, 2006.

[90] Paul Syverson and Aaron Johnson. More anonymous onion routing through trust. In *Proc. of CSF*, 2009.

[91] Suren Ter. You have downloaded - we show what you downloaded. http://www.youhavedownloaded.com/.

[92] China blocking tor. https://blog.torproject.org/blog/china-blocking-tor-round-two.

[93] Tor metrics portal: Performance. https://metrics.torproject.org/performance.html.

[94] Dinh Nguyen Tran, Frank Chiang, and Jinyang Li. Friendstore: cooperative online backup using trusted nodes. In *Proc. of WSNS*, 2008.

[95] Joseph Turow, Jennifer King, Chris Jay Hoofnagle, Amy Bleakley, and Michael Hennessy. Americans Reject Tailored Advertising and Three Activities That Enable It. *SSRN eLibrary*, September 2009.

[96] Ultrasurf. http://www.ultrareach.com/.

[97] Marc Waldman and David Mazieres. Tangler: A censorship-resistant publishing system based on document entanglements. In *Proc. of CCS*, 2001.

[98] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *USENIX Security*, 2000.

[99] Waste. http://wasteagain.sourceforge.net/.

[100] Oliver Widder. The "free" model. http://geekandpoke.typepad.com /geekandpoke/2010/12/the-free-model.html.

[101] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Proc. of NSDI*, 2011.

[102] Reporters without Borders. Internet enemies report 2012. http://march12.rsf.org /i/Report_EnemiesoftheInternet_2012.pdf.

[103] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. Telex: Anticensorship in the Network Infrastructure. In *Proc. of USENIX Security*, 2011.

[104] Vivian Wai yin Kwok. Aussie internet blacklist has gray areas. http://www.forbes.com/2009/03/19/australia-internet-censorship-markets-economy-wikileaks.html.

[105] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *Proc. of IEEE Security and Privacy*, 2008.

[106] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham D. Flaxman. SybilGuard: defending against sybil attacks via social networks. *Proc. of SIGCOMM*, 2006.

[107] Chao Zhang, Prithula Dhungel, Di Wu, Zhengye Liu, and Keith W. Ross. BitTorrent Darknets. In *Proc. of INFOCOM*, 2010.

[108] Ye Zhu, Xinwen Fu, Riccardo Bettati, and Wei Zhao. Anonymity analysis of mix networks against flow-correlation attacks. In *Proc. of GLOBECOM*, 2005.

VITA

Tomas Isdal is a graduate student at the University of Washington.

He welcomes your comments at tomas@isd.al and at http://isd.al