The GNUnet System

Christian Grothoff

Thèse d'habilitation à diriger des recherches
Universitée de Rennes 1
Mention: Informatique

The GNUnet System

Christian Grothoff

Soutenue le 10 octobre 2017 devant le jury composé de Messieurs les Professeurs:

Anne-Marie Kermarrec (Université de Rennes 1)
Tanja Lange (Technische Universiteit Eindhoven)
George Danezis (University College London)
Joe Cannataci (University of Groningen)
Saddek Bensalem (University of Grenoble)

Au vu des rapports de Messieurs les Professeurs:

Tanja Lange (Technische Universiteit Eindhoven)
George Danezis (University College London)
Saddek Bensalem (University of Grenoble)

Revision 1.0

Abstract

GNUnet is an alternative network stack for building secure, decentralized and privacy-preserving distributed applications. Our goal is to replace the old insecure Internet protocol stack. Starting from an application for secure publication of files, it has grown to include all kinds of basic protocol components and applications towards the creation of a GNU internet.

This habilitation provides an overview of the GNUnet architecture, including the development process, the network architecture and the software architecture. The goal of Part 1 is to provide an overview of how the various parts of the project work together today, and to then give ideas for future directions. The text is a first attempt to provide this kind of synthesis, and in return does not go into extensive technical depth on any particular topic. Part 2 then gives selected technical details based on eight publications covering many of the core components. This is a harsh selection; on the GNUnet website there are more than 50 published research papers and theses related to GNUnet, providing extensive and in-depth documentation. Finally, Part 3 gives an overview of current plans and future work.

Acknowledgements

Anne-Marie Kermarrec, Tanja Lange, George Danezis, Joe Cannataci and Saddek Bensalem have accepted to participate on the jury for my habilitation. I feel very lucky and honored, and thank them for that. I especially thank Tanja Lange for detailed and constructive comments.

I am grateful to and for the GNU project, in particular Richard Stallman and Werner Koch, for their long-standing and loud support for me and my projects. I thank all of the Free Software developers, in particular the hundreds of people that have contributed directly to GNUnet over the years.

All parts of this document stem from some type of collaboration and the text is based on the respective papers written with various co-authors, in particular my PhD students Nathan Evans, Matthias Wachs, Sree Harsha Totakura, Bartlomiej Polot, Alvaro Garcia-Recuero and Florian Dold. I thank Hernâni Marques for proofreading the entire document.

I also specifically want to thank developers from other projects sharing our goals and values, in particular I2P and Tor. Our discussions with them have been frequently insightful and I hope we will continue to productively work together in the future.

Finally, the GNUnet project would not be where it is today without the support by NLnet, the DFG (ENP GR 3688/1-1) and the Renewable Freedom Foundation.

To Torsten

Contents

I Overview **9**

1 Introduction **11**
1.1 The need for private communication 12
1.1.1 Authenticated encryption 12
1.1.2 Metadata 13
1.1.3 The client-server architecture 13
1.2 Decentralized Peer-to-Peer networks 14
1.3 Objectives for the GNUnet 15

2 Architecture **17**
2.1 Software architecture 17
2.1.1 Evolution 23
2.2 Security architecture 23
2.2.1 Access control 23
2.2.2 Secure APIs 25
2.3 Process architecture 26
2.3.1 Responsible disclosure 27
2.3.2 Peer review 27
2.3.3 Verification 27
2.3.4 Testing 28
2.3.5 Deployment 29
2.3.6 Monitoring 29
2.4 Network architecture 30
2.4.1 Overlay or underlay? 30
2.4.2 Structured or unstructured? 30
2.4.3 Bootstrapping 32

3 Key contributions **33**
3.1 Transport underlay abstraction 33
3.1.1 Automatic selection and resource allocation 33
3.1.2 Autonomous NAT traversal 34
3.2 Byzantine fault-tolerant routing 34
3.2.1 Secure network size estimation 35
3.2.2 R^5N: A secure distributed hash table 36

3.2.3 CADET: Confidential ad-hoc decentralized E2E transport 37
3.3 The GNU name system . . . 38
3.3.1 Revocation . . . 39
3.3.2 Conversation . . . 39
3.3.3 Protocol translation . . . 40

II Contributions in depth 41

4 Transport 43
4.1 Introduction . . . 43
4.2 Semantics of the Transport Abstraction . . . 44
4.2.1 Security Considerations . . . 45
4.3 Example: SMTP Implementation (Historic) . . . 46
4.3.1 Sending Email . . . 46
4.3.2 Receiving Email . . . 47
4.3.3 Security considerations for SMTP . . . 47
4.4 Related Work . . . 48
4.5 Autonomous NAT Traversal . . . 49
4.5.1 Technical Approach . . . 50
4.5.2 Implementations . . . 54
4.5.3 Experimental Results . . . 55
4.5.4 Discussion . . . 56
4.6 Transport selection problem . . . 56
4.6.1 Objectives for transport selection . . . 57
4.6.2 Scope . . . 58
4.7 Transport selection design . . . 58
4.7.1 The heuristic solver . . . 58
4.7.2 The linear optimisation solver . . . 59
4.7.3 The machine learning solver . . . 59
4.8 Implementation . . . 60
4.9 Evaluation . . . 60
4.9.1 Solver scalability evaluation . . . 61
4.9.2 Solver quality evaluation . . . 61
4.10 Discussion . . . 64
4.11 Conclusion . . . 65

5 Secure routing 67
5.1 Introduction . . . 67
5.2 Secure network size estimation . . . 68
5.2.1 Related Work . . . 69
5.2.2 Our Approach . . . 71
5.2.3 Security Analysis . . . 76
5.2.4 Experimental Results . . . 78
5.2.5 Discussion . . . 87
5.3 R^5N . . . 88

5.3.1 Related Work . . . 89
5.3.2 Restricted-Route Topologies . . . 90
5.3.3 Design of R^5N . . . 92
5.3.4 Experimental Results . . . 95
5.3.5 Performance Analysis . . . 103
5.4 CADET . . . 103
5.4.1 Connectivity . . . 103
5.4.2 Security . . . 105
5.4.3 Multiplexing . . . 106
5.5 Implementation . . . 109
5.6 Results . . . 109
5.6.1 Churn Resistance . . . 109
5.6.2 Latency . . . 111
5.6.3 Bandwidth . . . 113
5.7 Related Work . . . 113
5.7.1 TCP/IP . . . 113
5.7.2 Tor . . . 114
5.7.3 net2o . . . 115
5.8 Conclusion . . . 115

6 The GNU Name System 117
6.1 Introduction . . . 117
6.2 Requirements Analysis . . . 119
6.2.1 Adversary Model . . . 119
6.2.2 Functional Requirements . . . 120
6.3 Design Space for Name Systems . . . 120
6.3.1 Hierarchical Registration . . . 122
6.3.2 Cryptographic IDs and Mnemonics . . . 123
6.3.3 Petnames and SDSI . . . 123
6.3.4 Timeline-based Name Systems . . . 124
6.4 Practical Considerations . . . 124
6.4.1 Interoperability with DNS . . . 125
6.4.2 End-to-End Security and Errors . . . 126
6.4.3 Petnames and Legacy Applications . . . 126
6.4.4 Censorship-Resistant Lookup . . . 126
6.4.5 Case study: Usability . . . 127
6.5 Design of the GNU Name System . . . 128
6.5.1 Names, Zones and Delegations . . . 128
6.5.2 Zone Management with Nicknames and Petnames . . . 129
6.5.3 Relative Names for Transitivity of Delegations . . . 130
6.5.4 Absolute Names . . . 130
6.5.5 Records in GNS . . . 131
6.6 Query Privacy . . . 132
6.7 Security of GNS . . . 133
6.8 Special Features . . . 134
6.8.1 Automatic Shortening . . . 134

6.8.2 Relative Names in Record Values . . . 135
6.8.3 Dealing with Legacy Assumptions: Virtual Hosting and TLS . . . 135
6.8.4 Handling TLSA and SRV records . . . 136
6.8.5 Revocation . . . 137
6.8.6 Shadow Records . . . 138
6.8.7 Availability and Caching . . . 138
6.8.8 Intercepting DNS queries . . . 139
6.8.9 A HTTP SOCKS Proxy for Legacy Browsers . . . 140
6.9 Applications . . . 141
6.9.1 Key Exchange . . . 141
6.9.2 Telephony . . . 141
6.9.3 Other Applications . . . 142
6.10 Censorship in Other Layers . . . 142
6.11 Related Work . . . 143
6.12 Summary and Conclusion . . . 144

III Outlook 145

7 Ongoing and future work 147
7.1 Secure multiparty computations . . . 147
7.1.1 Private scalar product . . . 147
7.1.2 Private set intersection cardinality with signatures . . . 150
7.2 Social networking . . . 152
7.3 Payment and incentives . . . 154
7.4 News distribution . . . 155
7.5 Process architecture improvements . . . 156
7.5.1 Testing . . . 156
7.5.2 Monitoring . . . 156
7.5.3 Deployment . . . 157
7.6 Network architecture evolution . . . 157
7.7 A new underlay . . . 158

8 Related projects 161
8.1 Patch projects . . . 161
8.2 Sledgehammer projects . . . 162
8.3 Semi-related projects . . . 162
8.4 net2o . . . 162

9 Discussion & Conclusion 163

Part I

Overview

Chapter 1

Introduction

Internet protocols need a general overhaul to make them suitable as the main communication infrastructure for a sustainable civil liberal society. When the US military created the Internet, the fundamental goal was to interconnect existing networks, which was a simple practical objective. What is more lasting were the second level goals *in order of importance* as elaborated by David Clark in [Cla88]:

1. Internet communication must continue despite loss of networks or gateways.
2. The Internet must support multiple types of communications service.
3. The Internet architecture must accommodate a variety of networks.
4. The Internet architecture must permit distributed management of its resources.
5. The Internet architecture must be cost-effective.
6. The Internet architecture must permit host attachment with a low level of effort.
7. The resources used in the internet (sic) architecture must be accountable.

We note that the only security-related goals relate to availability and usability. While some Internet architects were aware of encryption due to their proximity to the NSA, encryption was classified and thus could not even legally be included in the emerging Internet design that was deliberately non-classified.

As a result of these 1970 design goals, the lack of data protection remains visible at all layers of the Internet architecture today. Figure 1.1 shows the design of the IPv4 header from 1981, which is still in use today. It unnecessarily leaks information about the sender address and the encapsulated transport protocol. Additionally, the checksum is insufficient to provide authenticity, and it is difficult to prove ownership of a destination address.

Version	HDL	ToS	Length	
Identification			Flags	Fragment offset
TTL		T. Protocol	Checksum	
Source IP address				
Destination IP address				
Options (optional)				
Data (Length–HDL bytes)				

Figure 1.1: The IPv4 header (Sept. 1981)

1.1 The need for private communication

Today, the actual use and thus the social requirements for a global network differs widely from those goals of 1970. While the Internet remains suitable for military use, where the network equipment is operated by a command hierarchy and when necessary isolated from the rest of the world, the situation is less tenable for civil society.

Due to fundamental Internet design choices, Internet traffic can be misdirected, intercepted, censored and manipulated by hostile routers on the network. And indeed, the modern Internet has evolved exactly to the point where, as Matthew Green put it, "the network is hostile".[1]

1.1.1 Authenticated encryption

With the emergence of e-commerce, a throughly broken [TP11] cryptographic protocol called "Secure Sockets Layer" (SSL) was introduced to provide a minimum of authenticity and confidentiality. The rushed deployment and lack of understanding of cryptography by the designers resulted in a complex design that took the Internet community 20 years to sufficiently understand and thereby obsolete. The surviving complex cryptographic libraries, like OpenSSL, GnuTLS and libnss, remain a treasure trove for vulnerabilities.

While we could probably eventually provide good authenticated encryption on the existing Internet, this is by far not sufficient. Most schemes for authenticated encryption require the use of a public key infrastructure (PKI), and thus we need to consider the vulnerabilities of the PKI when assessing the security of the system. In particular, the dominant X.509 PKI fails to provide adequate protections against attacks by large corporations or governments, which has resulted in a series of proposed security extensions and monitors to work around the broken paradigm of certificate authorities [Hol14]. As modernist proposals for opportunistic encryption [Duk14] without a PKI simply blank out the existence of active adversaries, a new security architecture is required to provide even just this most basic level of security.

[1] `http://blog.cryptographyengineering.com/2015/08/the-network-is-hostile.html`

1.1.2 Metadata

However, even if we consistently deployed authenticated encryption on the Internet, the network would still learn way too much. Authenticated encryption does not obfuscate sender and receiver, and also discloses the times, frequency and volume of the communication, which enables reverse engineering of the set of pages visited via traffic analysis [MHJT14]. How revealing this information actually is becomes clear if one considers research on website fingerprinting in the Tor network. Here, the user is assumed to use Tor [DMS04b] to hide the IP address of the sender and receiver from the network. As Tor provides a low-latency non-padding anonymizing proxy, the adversary now only learns the time, frequency and volume of the communication. However, based solely on this information researchers were able to correctly classify 97% of requests on a sample of 775 sites and 300,000 real-world traffic dumps [HWF09].

Metadata is by definition data about data, and thus provides a higher level abstraction of the underlying data. Therefore, leaking metadata is not a cosmetic issue. As Michael Hayden, the former head of both NSA and CIA explained, the US government "kills based on metadata". The democide [Rum07] that has resulted from the NSA's application of statistics and machine learning to metadata leaked by information networks [EG15] is thus partially the responsibility of computer scientists, as we have failed to build and deploy systems that adequately protect this critical information.

1.1.3 The client-server architecture

A second crucial issue with modern network architecture is the client-server paradigm. The client-server paradigm is about resource-strapped clients communicating with high-powered servers. The clients are expected to obey by the (protocol) rules of the server, which provides a service to the client. The server is in control of either the critical data for storage or messaging applications, or the critical computation for transaction processing. The paradigm has the advantage that it scales well: the roles in the protocol are well-defined, the clients are (relatively) simple and most of the administrative effort is concentrated at the data centers operating the servers. It also provides simple business models for the service providers, who can either sell the service to the client or the client's data to other businesses.

Authenticated encryption is of limited utility against the client-server paradigm. While it is practical to encrypt data before uploading it to a storage service, this rarely applies to other types of network services. Consequently, the NSA's PRISM program was used to systematically access citizen's data at providers of messaging services, including most popular online social networking application providers. Here, the hunger for data by the government is paralleled by the hunger for data by the advertising-based service providers.

However, even if the service provider did not have this inherent incentive to collect as much information as possible, the mere existence of easily identifiable large service providers is problematic, as for most computational appli-

cations the service provider needs to learn crucial information to even provide the service. Researchers have frequently proposed that the servers should perform their computation on encrypted data. However, despite recent theoretical advances in fully-homomorphic encryption [Gen09] and private information retrieval [BDG14], the concrete realization of scalable network services using this paradigm remains elusive for the foreseeable future.

The client-server paradigm is dangerous to a liberal society for at least two fundamental reasons. First, it results in client systems that are virtually useless without the service provided by large service providers operating servers. The Android operating system is a key example, as most of it only operates in conjunction with services provided exclusively by Google's data centers. Such systems are a clear violation of the user's autonomy and they unduly restrict how citizens can cooperate amongst themselves.

The second problem with servers is that it is hard to design client-server systems in such a way that the server does not learn sensitive information about the clients. In fact, often the business model of the server operator is to collect and monetize client information. Thus, privacy and data protection issues will continue as long as the dominant paradigm on the Internet is client-server.

We note that outside of TCP, the client-server relationship is often described using the terms "master" and "slave". These terms more aptly describe the social roles the client-server paradigm produces.

1.2 Decentralized Peer-to-Peer networks

The peer-to-peer (P2P) paradigm is the alternative to the client-server paradigm for multi-party systems. In P2P systems, the participants interact with each other on equal terms, with the peers jointly providing a service to each other. There can still be roles in the system, such as the *initiator* of a handshake or the *sender* or *receiver* or the *resource provider*, but by design all peers may assume any role at any time.

Naturally, some peers tend to be better at providing resources, which sometimes gives rise to designs that distinguishes peers and "super-peers" [BYGM03]. What distinguishes super-peers from servers is that any peer can decide to assume the role of a super-peer. Thus, these super-peers are merely "special" because they typically provide more resources and thus can be used by the designer in particular ways to improve performance, but do not exert control. In contrast, in the client-server paradigm the server is authoritarian [Alt06].

Shifting from a client-server architecture to a decentralized peer-to-peer architecture liberates participants from the dictatorial rule of the server, but does not inherently improve security and privacy. In fact, unless the protocols are carefully designed, the result of moving from client-server to peer-to-peer creates *additional* data leakage: instead of just the server and the network learning sensitive information, it is now possible that arbitrary peers may obtain sensitive information. For example, hosting files on a public FTP server allows the FTP server to see who is downloading the files. Hosting files on BitTorrent [Coh03]

enables all peers participating in the transmission of the Torrent to observe the download, and not just the FTP server and the network.

We also note that it is both almost trivially easy to ensure privacy and security with a trusted third party and entirely inadequate. The introduction of a trusted third party makes it relatively easy to provide a public key infrastructure, reliable storage, authentication and a host of other core security features. However, the reliance on a trusted third party also ensures that this party becomes extremely powerful and thus is likely to be either inherently evil or a victim for high-profile attacks, and most likely both. As a result, designs with a trusted third party are likely to eventually fundamentally fail at least some of their users.

Thus, liberal societies need a network architecture that uses the anti-authoritarian decentralized peer-to-peer paradigm and privacy-preserving cryptographic protocols. The goal of the GNUnet project is to provide a Free Software realization of this ideal.

1.3 Objectives for the GNUnet

The fundamental goal of the GNUnet is to lessen the threat of networks supporting totalitarian control over the population. This includes both control over their networking and computing, as well as ensuring that the network protects privacy. To clarify this, we propose the following ten objectives for the design of the GNUnet in this order:

1. The GNUnet must be implemented as Free Software, to ensure that citizens enjoy the four essential freedoms of Free Software [Sta02].

2. The GNUnet must only disclose the minimal amount of information necessary, to second and especially third parties. Each user decides what information can be shared and with whom.

3. The GNUnet must be decentralized and survive Byzantine failures in any position in the network. Increased malicious participation may cause security and privacy assurances to deteriorate commensurate with the resources expended by the adversary.

4. The GNUnet must make it explicit to the user which entities must be trustworthy when establishing secured communications.

5. The GNUnet must use compartmentalization to isolate sensitive information against Byzantine failures even among layers of the implementation on the same device.

6. The GNUnet must be open and permit new peers to join.

7. The GNUnet must be self-organizing and not depend on administrators.

8. The GNUnet must support a diverse range of applications and devices.

9. The GNUnet must scale and be cost-effective.
10. The GNUnet must provide incentives for peers to contribute more resources than they consume.

As with the Internet architecture, the order of these objectives matters. For example, a proprietary solution (violating (1)) may disclose less information simply (improving (2)) by forcing an adversary to reverse engineer the protocol, but this is not acceptable as it violates the four essential freedoms. Another example would be a user who decides that they do not want to disclose that they are even participating in the network (improving (2)). Thus, GNUnet optionally allows peers to restrict connections to "friends-only", even though this may prevent new peers from joining the network (violating objective (6)).

We note that many of Clark's original Internet design goals correspond to design goals at the bottom of the prioritized list for the GNUnet: enabling host attachment, diversity of use, cost-effectiveness and accountability remain important, but need to fall behind the need for data protection and Byzantine fault tolerance.

GNUnet's objectives also goes beyond the simple slogan of "privacy-by-design" [Cav09]. While objective (2) is enshrining data protection as a key goal, privacy-by-design only calls for respect for the user's privacy, while objectives (1) and (3) are more broadly protecting the user's autonomy. In contrast to privacy-by-design, in GNUnet the use of proprietary software or centralized service providers is not acceptable, as they make citizens dependent on software vendors or service providers.

Structure

The next chapter will give a high-level description of the GNUnet architecture, followed by a summary of the key contributions in Chapter 3. Challenges for ongoing and future work are discussed in Chapter 7.

Chapter 2

Architecture

This chapter will present the architecture of GNUnet. We will consider the architecture from four different angles. The *software architecture* gives an overview of how the GNUnet software is supposed to operate from the point of view of a single computer implementing a peer participating in the GNUnet. The software architecture strongly relates to the *security architecture* which describes the various architectural means used to improve the security of the system. The *process architecture* covers the architecture of the development process, that is how designers and developers contribute towards building the system. Finally, the *network architecture* describes the structure of the overlay network, that is how the various peers structure their communication.

2.1 Software architecture

When introducing network architectures, educators typically begin by describing a layered design. The numbering differs depending on the model, but the design typically ranges from the physical layer to the application layer. In this section, we will follow this method and leave the higher human, organizational and political layers to Section 2.3. Figure 2.1 illustrates how the various layers of GNUnet relate to well-known layers in the TCP/IP stack.

The use of layers to describe a software architecture is usually a drastic simplification. "IP/BGP" absolutely understates the complexity at the routing layer for the modern Internet, and of course the BGP protocol runs above TCP/UDP, but at the same time is really part of the IP layer. Recent IETF proposals to run DNSSEC over TLS [HZH+15] to ultimately secure the exchange of TLSA records over DNSSEC [Wel00] illustrate that real networks do not follow a simple layered design, and that the layers are merely an illustration to provide a first orientation.

What Figure 2.1 shows is that there is some high-level resemblance between the GNUnet architecture and the Internet: there is an underlying communication mechanism ("TRANSPORT") with very weak capabilities, semantics and

Internet	*GNUnet*
Facebook/Google	Applications
HTTP	PSYC
DNS/X.509	GNU Name System
TCP/UDP	CADET
IP/BGP	R^5N DHT
Ethernet	CORE
Physical Layer	TRANSPORT

Figure 2.1: Layers in the Internet and GNUnet architectures (simplified)

assurances, a first link-level mechanism to facilitate a fundamental level of service ("CORE"), followed by routing ("DHT"), reliable end-to-end communication ("CADET"), a system for reliably naming objects on the network ("GNS"), a messaging protocol ("PSYC") and finally applications.

However, this is just a first rough orientation to highlight similarities with the Internet architecture. Figure 2.2 shows a more realistic, but still simplified picture of the interdependencies between GNUnet subsystems. The different styles used for the components indicate whether it is a simple library (route), a simple binary (rectangle), a service exporting an API (oval) or an application visible to the user (house). We note that this figure does form a directed acyclic graph, which is necessary because it literally corresponds to the interdependencies during compilation of the various independent compilation units.

Figure 2.2 is still a simplification, as certain common subsystems are omitted, such as the "UTIL" library which provides a library with basic functions that virtually all subsystems depend on, or the "STATISTICS" which is a subsystem used by most other subsystems to track performance data. The figure also does not even show all of the over 60 subsystems available today, and instead focuses on those most relevant for an understanding of the overall architecture and the key contributions that will be presented in Chapter 3.

We note that even a complete picture of all subsystems implemented or even planned today would not describe the GNUnet of the future, as the most important point of the architecture is that it is extensible. As shown in Figure 2.2, it is quite common for one layer to have various higher layers build on it, and in general each service is designed to be used and extended by anyone who finds the provided API to be useful. Thus, we need to think about GNUnet primarily as a collection of APIs providing various core functions for future Internet applications. Over time, additional abstractions will be added, and the existing 60 APIs are just the beginning.

Currently, the following GNUnet subsystems are relevant for understanding the overall architecture and key contributions:

util Library with general utility functions, all GNUnet binaries link against this library. Anything from memory allocation and data structures to

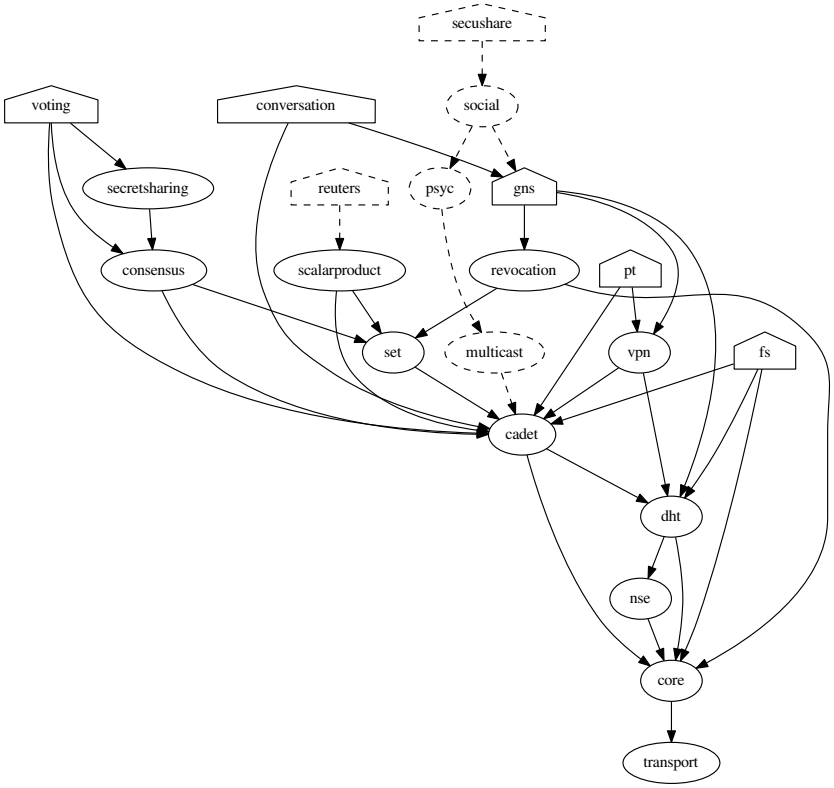


Figure 2.2: Dependencies between GNUnet subsystems (simplified). Houses represent applications. Dashed lines represent components that are still incomplete at the time of this writing.

cryptography and inter-process communication. The goal is to provide an OS-independent interface and more "secure" or convenient implementations of commonly used primitives. The API is spread over more than a dozen headers, developers should study those closely to avoid duplicating existing functions.

block The DHT and other components of GNUnet store information in units called *blocks*. Each block has a type and the type defines a particular format and how that binary format is to be linked to the key for the DHT. The block library is a wrapper around block plugins which provide the necessary functions for each block type.

statistics The statistics service enables associating values (of type `uint64_t`) with a component name and a string. The main uses are debugging by counting events, performance tracking and user entertainment, as users like information about what their computer does.

arm The automatic restart manager (ARM) service is the GNUnet master service. Its role is to start gnunet-services, to restart them when they crashed and finally to shut down the system when requested.

peerinfo The peerinfo service keeps track of which peers are known to the local peer and also tracks the validated addresses for each peer in the form of a "HELLO message" for each of those peers. The peer is not necessarily connected to all peers known to the peerinfo service. Peerinfo provides persistent storage for peer identities — peers are not forgotten just because of a system restart.

ats The automatic transport selection (ATS) service is responsible for deciding which address, i.e. which transport plugin, should be used for communication with other peers, and at what bandwidth. [WOG14]

nat Library that provides basic functions for NAT traversal. The library supports NAT traversal with manual hole-punching by the user, UPnP and ICMP-based autonomous NAT traversal. The library also includes an API for testing if the current configuration works and the `gnunet-nat-server` which provides an external service to test the local configuration. [MEGK10]

transport The transport service is responsible for managing the basic P2P communication. It uses plugins to support P2P communication over TCP, UDP, HTTP, HTTPS and other protocols. The transport service validates peer addresses, enforces bandwidth restrictions, limits the total number of connections and enforces connectivity restrictions, such as friends-only. [FGR03]

core The core service is responsible for establishing encrypted, authenticated connections with other peers, encrypting and decrypting messages and forwarding messages to higher-level services that are interested in them.

testing The testing library allows starting and stopping peers for writing test cases. It also supports automatic generation of configurations for peers ensuring that the TCP ports and file system paths are disjoint. `libgnunettesting` is also the foundation for the testbed service.

testbed The testbed service is used for creating small or large scale deployments of GNUnet peers for evaluation of protocols. It facilitates peer deployments on multiple hosts (for example, in a cluster) and establishing various network topologies (both underlay and overlay). [Tot13a]

nse The network size estimation (NSE) service implements a protocol for securely estimating the current size of the P2P network. [EPG12]

dht The distributed hash table (DHT) service provides a distributed implementation of a hash table to store blocks under hash keys in the P2P network. [EG11b]

hostlist The hostlist service allows learning about other peers in the network by downloading HELLO messages from an HTTP server, can be configured to run such an HTTP server and also implements a P2P protocol to advertise and automatically learn about other peers that offer a public hostlist server. [GG08]

topology The topology service is responsible for maintaining the overlay topology. It tries to maintain connections to friends and also tries to ensure that the peer has a decent number of active connections at all times. If necessary, new connections are added. Peers generally need at least one service that demands connections to ensure that they remain connected with the network. Enabling the topology service is thus one way to ensure that connectivity is maintained. If friend-to-friend networking is enabled in the configuration, the topology service also tells the transport service which connections are allowed or forbidden.

cadet The CADET service [PG14] provides a general-purpose routing abstraction to create end-to-end encrypted tunnels on top of a DHT like R^5N.

fs The file sharing (FS) service implements GNUnet's file sharing application. Both anonymous file sharing using the `gap` routing protocol [BG03] and non-anonymous file sharing using CADET are supported. [BGHP02]

dns Service that allows intercepting and modifying DNS requests of the local machine. Currently used for IPv4-IPv6 protocol translation (DNS-ALG) [STAH99] as required by PT and GNS. The service can also be configured to offer an exit service for DNS traffic.

vpn The virtual public network (VPN) service provides a virtual tunnel interface (VTUN) for IP routing over GNUnet. Needs some other peers to run an EXIT service to work. Can be activated using the "gnunet-vpn" tool or integrated with DNS using the **pt** daemon.

exit Daemon to allow traffic from the VPN to exit this peer to the Internet or to specific IP-based services of the local peer. Currently, an exit service can only be restricted to IPv4 or IPv6, not to specific ports and or IP address ranges. If this is not acceptable, additional firewall rules must be added manually. exit currently only works for normal UDP, TCP and ICMP traffic; DNS queries need to leave the system via a DNS service.

pt protocol translation daemon. This daemon enables 4-to-6, 6-to-4, 4-over-6 or 6-over-4 transitions for the local system. It essentially uses the DNS service to intercept DNS replies and then maps results to those offered by the VPN, which then sends them using **cadet** to some daemon offering an appropriate **exit** service.

identity Management of egos (alter egos) of a user; identities are essentially named ECC private keys and used for zones in the GNU name system and for namespaces in file sharing, but might find other uses later.

set Efficient two-party set operations (union or intersection) using (invertible) Bloom filters [EGUV11].

revocation Key revocation service, can be used to revoke the private key of an identity if it has been compromised. [WSG14]

gns GNU name system (GNS), a GNU approach to name resolution and public key infrastructure. [WSG13, WSG14]

dv A plugin for routing based on (bounded) distance-vector (DV). DV consists of a service and a transport plugin to provide peers with the illusion of a direct P2P connection for connections that use multiple (typically up to 3) hops in the actual underlay network.

regex Service for the distributed evaluation of regular expressions. [Sze12]

scalarproduct The scalar product service offers an API to perform a secure multi party computation which calculates a scalar product between two peers without exposing the private input vectors of the peers to each other.

consensus The consensus service allows a set of peers to agree on a set of values via a distributed set union computation. [DG16]

rest The rest API allows access to GNUnet services using RESTful interaction. The services provide plugins that can exposed by the rest server.

Many of these subsystems are described in research publications or theses relating to the development of the GNUnet. Chapter 3 will go into details on some of the most important technical contributions.

2.1.1 Evolution

Thinking in APIs for subsystems is key for GNUnet development as it allows us to evolve the implementation. GNUnet may use one particular type of routing mechanism or authenticated encryption today, and tomorrow research may provide us with a significantly better option. As long as the API provided by the improved design is the same, we can evolve subsystems independently. Naturally, transitioning a network to a new version of the protocol may create interoperability issues, but the use of APIs allows developers to often separate the evolution into two types of evolution: improvements to the API and improvements to the protocol. While these are sometimes interdependent, we are often able to involve or investigate improvements to one without impacting the other, thus maximizing our chances of preserving compatibility while supporting the system's continuous evolution.

2.2 Security architecture

When building a complex new network architecture, a key consideration is how to engineer the architecture to minimize the impact of bugs. While the software development process (described in Section 2.3) is supposed to minimize the *existence* of bugs, the GNUnet security architecture tries to minimize the *impact* of bugs.

The first key choice in this context was to implement GNUnet outside of the operating system kernel. While this almost ensures that GNUnet will never be as fast as say TCP/IP, we remind the reader that data security and compartmentalization are objectives that trump scalability and cost-effectiveness for GNUnet (Section 1.3). In fact, GNUnet not only does not live in the kernel, but generally uses a separate process for each subsystem. These processes then communicate via IPC, typically UNIX domain sockets, providing strong isolation of components from each other.

Programming with GNUnet components thus is a bit like playing with legos: each component itself is (presumably) an "indestructible" self-contained "brick" which exposes APIs to the outside, and itself sits on top of APIs provided by other components (Figure 2.3). In practice, there are always two APIs: a (C or Java) library exposing a language-specific API, and the component itself exposing an API in the form of an IPC protocol. We note that GNUnet did not really invent this architecture: remote procedure calls [Mic88] predate GNUnet by about two decades. However, we are not aware of other networking projects atomizing their stack into small components to the extend done by GNUnet.

2.2.1 Access control

Because components run as separate processes, they can also be easily subjected to different security policies at the system call API. For example, a subsystem performing routing may not have write access to the disk, or a database backend might not have access to the network. On Linux, GNUnet provides

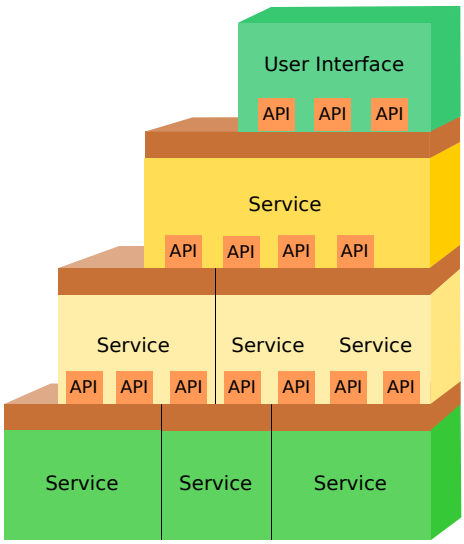


Figure 2.3: GNUnet: Programming with legos.

Apparmor [Inc15] profiles to restrict the various processes accordingly. When subsystems communicate with each other, they are expected to perform input validation just as if the communication came over an insecure network. Subsystems providing services thus always need to validate inputs, and, if applicable, rate-limit requests from clients.

Typically, subsystems running on one host execute either as the system-user "gnunet" or under the rights of the respective user. Some subsystems require even stronger isolation. GNUnet uses a few processes that need to run with administrator ("root") privileges or otherwise elevated rights. Thus, when configured with stringent policies, a GNUnet installation consists of subsystems that fall into one of four categories of privileges:

User interfaces User interfaces are not security sensitive and are supposed to be run and used by normal system users. The GTK GUIs and most command-line programs fall into this category. Some command-line tools (like `gnunet-transport`) should be excluded as they offer low-level access that normal users should not need.

System services and support tools System services should always run and offer services that can then be accessed by the normal users. System services do not require special permissions, but as they are not specific to a particular user, they probably should not run as a particular user. Also, there should typically only be one GNUnet peer per host.

Privileged helpers Some GNUnet components require administrative rights to open raw sockets or perform other special operations. These "gnunet-

helper-*" binaries are typically installed SUID and run from services or daemons.

Critical services Some GNUnet services, in particular the DNS service, can manipulate the system in deep and possibly highly security sensitive ways. In particular, the DNS service can be used to intercept and alter any DNS query originating from the local machine. Access to the APIs of these critical services and their privileged helpers must be tightly controlled.

To achieve the latter, our installation guidelines recommend the creation of a special group `gnunetdns` to control access to the `gnunet-helper-dns`. The binary should then be owned by `root` and be in group `gnunetdns` and be installed SUID and only be group-executable (2750). Note that the group `gnunetdns` should have no users in it at all, ever. The `gnunet-service-dns` program should be executed by user `gnunet` with the binary owned by the user `root` and the group `gnunetdns` and be SGID (2700). This way, only `gnunet-service-dns` can change its group to `gnunetdns` and execute the helper, and the helper can then run as root (as per SUID). Access to the API offered by `gnunet-service-dns` is in turn restricted to the user `gnunet` (not the group!), which means that only "benign" services can manipulate DNS queries using `gnunet-service-dns`.

GNUnet is not the only component in our overall system that goes to this extreme to isolate faults. The GNU libextractor [Gro05] library is used by the GNUnet file sharing subsystem to extract meta data, such as ID3 tags, previews or EXIV image information, from published files. GNU libextractor uses a wide range of standard libraries, such as `libavcodec`, `libjpeg` or `libexiv2` to obtain this meta data. As these libraries are complex parsers operating on potentially malicious inputs, GNU libextractor isolates the parsing logic in a restricted process that literally is only allowed to load its code, allocate memory, compute (for a finite amount of time), read from a shared memory segment and communicate with its parent via an inherited pair of file descriptors. Consequently, even arbitrary code execution vulnerabilities in the parser libraries would be harmless as long as mere computation does not expose access to vulnerabilities in the operating system or hardware.

The complex multi-process, multi-user, multi-group setup provides GNUnet superior fault isolation, at the expense of a significantly more difficult installation process. However, while there is hope that such an installation process can be sufficiently automated, there is much less hope for monolithic architectures like modern browsers or in-kernel network stacks where vulnerabilities are much harder to contain.

2.2.2 Secure APIs

Another key part of GNUnet's security architecture is the provisioning of APIs that make it harder to write buggy code. While most of GNUnet is currently written in C, many of C's security nightmares are rare because the APIs make it harder to run into the respective problems. For example:

- `GNUNET_malloc()` checks for allocation failures and prohibits large allocations ($\geq$ 40 MB) which might be the result of an integer underflow.
- `GNUNET_asprintf()` provides an alternative to `sprintf()` which allocates a buffer of the appropriate size for the user, thereby making it impossible to supply a buffer of insufficient size.[1]
- The `GNUNET_TIME` API provides routines for manipulating time stamps and units of time and internally protects against overflows or underflows in the computations.
- In the GNUnet C code, the use of threads is categorically forbidden, thus eliminating the possibility of deadlocks and data races.

Naturally, these are only some limited techniques to address some C-specific issues. However, the basic rule to provide abstractions that are safe to use, or at least hard to use badly, applies both to the design of new APIs and to GNUnet components written in other languages.

2.3 Process architecture

Developers rarely join the team as part of a formal hiring process. Most often, developers are volunteers that freely decide what issues they want to work on, and when to do the work. This common structure for Free Software projects dictates many choices for the process architecture: volunteers cannot be forced to do something, they can only be reasoned with. Thus, the GNUnet development process rarely tries to prevent undesirable changes, and instead strives to inform about inadequacies.

As with virtually any modern software development, the central place for all development activity is the version control system. Here, developers can inspect the current code and propose or directly make changes. Version control also allows us to easily undo problematic changes, thus it is not so critical that every change is consistently perfect. Read-access to the main repository is available to anyone. Write-access to the main version control system is granted to all developers that have convinced the maintainer that they are able to contribute constructively to the project.

The development process architecture described in the rest of the document is an attempt to "control" the development, given the constraint that we need to facilitate development performed by volunteers. In particular, the process cannot simply try to restrict the activities of volunteers. Instead, it must be constructive, inform about opportunities for contributions, and give feedback about the benefits or problems that may have resulted from the work of the volunteers.

[1] The API naturally is named after and corresponds to the GNU libc extension `asprintf()`, except that allocation failures are dealt with more harshly.

2.3.1 Responsible disclosure

The first process any modern software should have is a method to support responsible disclosure. GNUnet is a GNU package, thus the GNU project provides some basic infrastructure for development, including a global security team that can act as a point-of-contact as a last resort.[2] This has so far never been necessary, but in the theoretical case that everyone else is unreachable this is a good fallback to have.

In general, GNUnet has a public bug tracker, but submitters can also mark a sensitive bug report as "private", thereby restricting its visibility to registered developers. For more urgent or highly sensitive reports, it is also possible to reach most developers and the maintainer via GnuPG-encrypted email.

2.3.2 Peer review

GNUnet uses peer review in three ways. First, when applicable new interesting designs are documented in the form of research papers and submitted to appropriate academic conferences for review. If applicable, feedback obtained from the reviews is then addressed in the design and implementation. A second type of review is the direct discussion of the design with interested researchers, developers or activists. In practice, this less formal review tends to be more productive, likely because of the higher qualifications and attention span of the parties involved. Finally, any change in the code is automatically sent via email to an archived, global public mailing list, and many developers are subscribed to that list. This typically leads to a review of small changes, as they are fast and easy to do, and larger architectural changes, as they create curiosity as to what on earth is going on. However, speaking from personal experience, this process also allows medium-sized complex changes confined to a particular subsystem to go through without any attention.

On occasion, be it because a component is "finished", particularly sensitive or particularly buggy, the team organizes code reviews where one or more team members review the code of a subsystem with its author. These tend to be particularly productive with inexperienced students that often fail to use existing APIs correctly.

2.3.3 Verification

We have subjected the GNUnet code base to various static analysis tools over the years. The most regularly used tool is the C compiler itself, as developers typically compile with warnings enabled (`-Wall`). Using a variety of C compilers, in particular `gcc`, but also `clang` and `cparser` from Firm [Lin02] can sometimes give additional relevant warnings.

The LLVM framework used for `clang` also provides the foundation for the Clang static analyzer[3], which has occasionally been useful to identify minor

[2] `https://www.gnu.org/security/`

[3] `http://clang-analyzer.llvm.org/`

issues. Another tool which finds stylistic bugs — but rarely significant issues — is cppcheck[4].

We also have experimented with various commercial tools, but the licensing typically imposes serious restrictions, including on publishing comparisons between the tools. The main difference is that the commercial tools tend to be better at analyzing the control- and data-flow graph across compilation units.

Given that GNUnet is largely written in C, the guarantees that these tools can provide in terms of which types of bugs are no longer present are rather weak. However, they do tend to do a reasonable job at informing developers that return values are unchecked and can be helpful at quickly finding missing logic to clean up resources, especially in error-handling routines.

2.3.4 Testing

As with any large software project, systematic testing is part of the GNUnet development process. Comparatively speaking, GNUnet has rather few unit tests in which the functionality of individual functions is tested. Most tests in the existing system focus on the subsystems, as they provide and export well-defined APIs. This both facilitates testing and gives the test a reasonably stable abstraction to work against. Also, given the use of event-loops and callbacks and many `static` functions to limit the scope of symbols, testing individual functions using unit tests is generally more difficult and often less meaningful.

The TESTBED subsystem [Tot13a] of GNUnet provides the foundation for integration tests. With TESTBED, it is easy to start a number of peers, connect them into a particular overlay topology and run experiments against the exported APIs. TESTBED also facilitates collecting results, including performance data. GNUnet uses the TESTBED subsystem both for integration tests as well as for large-scale performance evaluation. Here, TESTBED can manage experiments run across a cluster of machines or even an HPC system, potentially running experiments where a million peers are emulated.

The quality of the test suite is evaluated using `lcov`[5], a code coverage analysis tool which integrates with `gcc` to highlight which functions, lines or branches of the code are not covered by existing tests. This way, developers and management can quickly identify which parts of the code still ought to be tested. Given good code coverage, most typical C-specific bugs relating to memory management are then easily found using `valgrind`[6].

To verify that the code works on various platforms (both hardware and operating systems), GNUnet uses the Buildbot continuous integration tool[7]. Buildbot uses "slave" machines which connect to the Buildbot "master" server to receive information about jobs to be executed. Typical job descriptions include downloading the latest version of the code, compiling it and running the test suite. The results of the test are then visualized on the GNUnet website.

[4] `http://sourceforge.net/projects/cppcheck/`
[5] `http://ltp.sourceforge.net/coverage/lcov.php`
[6] `http://valgrind.org/`
[7] `http://buildbot.net/`

This makes it easy for developers and management to identify which subsystems have failed tests on which platforms. Many of the buildslaves are run as virtual machines, but the team also operates buildslaves directly on Sparc64, PowerPC and ARM hardware. These buildslaves often highlight issues relating to timing, endianness or memory alignment.

Performance can also sometimes be a correctness issue. However, especially across platforms it is generally difficult to set hard performance requirements. Thus, the approach for GNUnet has been to track and visualize performance per platform. This allows developers to spot performance regressions. For this, we developed the cross-language Gauger tool [PG11]. Gauger collects performance data, which is typically submitted by Buildbot slaves as part of their test suite runs, and allows users to group data by machine or metric over time. Gauger does not group by submission time, but by the respective revision numbers in the version control system, thus allowing developers to identify which changes in the code resulted in performance regressions or improvements.

2.3.5 Deployment

The various metrics provided by the systems described in the previous sections inform the release manager about the quality of the current code and its readiness for release. If the code is ready, a version number is decided based on backwards-compatibility considerations. If the network protocols are incompatible, the major version is updated, otherwise the minor version. Similar considerations apply for the versioning of the libraries included in the system, where the `current:revision:age` tuple must be updated based on symbol compatibility.[8]

Once the versions have been decided, the release manager creates a release file following the GNU Autotools process, and in particular by using the `make dist` command. Following the GNU maintainer guide, the resulting source file is signed using GnuPG and uploaded together with a directive file to the GNU FTP upload server. From there, it is automatically distributed to all GNU FTP mirrors worldwide. Then, a notification about the new release is posted on the mailing lists and the website.

Various GNU/Linux distributions then package the source file for the respective distribution. The resulting packages, often containing binary code for a particular platform, are then distributed to the mirrors of the respective distribution, and then made available via the package manager of the distribution to the respective user base.

2.3.6 Monitoring

Once deployed, users or developers can monitor the running peer using various tools. In particular, the STATISTICS subsystem collects ongoing metrics about the execution of the peer, but other subsystems also generally export custom

[8] `https://www.gnu.org/software/libtool/manual/html_node/Updating-version-info.html`

information about their execution. For example, TRANSPORT and CORE can be asked about the active connections of the peer. GNUnet includes both command-line tools and GUIs to display and visualize the resulting streams of information.

2.4 Network architecture

In this section we describe the network architecture of GNUnet, that is how peers are expected to communicate with each other. GNUnet is special in that instead of falling into one of the usual categories, it typically falls into *all* of them.

2.4.1 Overlay or underlay?

GNUnet is an *overlay* network, as it is initially supposed to primarily operate *over* the existing Internet network. However, GNUnet does not assume that this is always the case. For example, GNUnet can also operate directly over WLAN or Bluetooth to create an ad-hoc wireless mesh network, and with the GNUnet "PT/VPN" subsystems we can run TCP/IP *over* GNUnet. So in this configuration, GNUnet would be an *underlay* network, a bit like MPLS or B.A.T.M.A.N. [NAL07]. In reality, we in fact expect to see both at the same time: some peers will run over the existing Internet, while others may connect to GNUnet on Layer 2. And IP traffic initially run *over* GNUnet may use the EXIT subsystem to cross to the Internet at other peers where GNUnet runs *over* IP. The situation is thus more related to that of IPv6, where IPv6 can run over IPv4 and IPv4 can run over IPv6 and protocol translation can convert between the two. Naturally, GNUnet also runs over both IPv4 or IPv6 and the GNUnet protocol translation ("PT") service supports providing protocol translation for IPv4/IPv6 traffic that runs over GNUnet.

2.4.2 Structured or unstructured?

Given that a GNUnet peer may be connected via any kind of communication mechanism to some subset of the network (Figure 2.4), the lowest layers in the GNUnet stack cannot make any hard assumptions about the structure of the underlying network. Thus, the most basic characterization of the GNUnet topology is *unstructured*. However, for *performance evaluation*, we generally impose some restrictions, such as a small-world assumption which states that short ($O(\log n)$) paths *exist* to all other nodes in the network.

Some GNUnet protocols operate directly over the unstructured topology. Examples include the *gap* anonymous query protocol [BG03], the protocols for network size estimation [EPG12] and key revocation [WSG14]. The TOPOLOGY subsystem is used to maintain a "random" mesh topology. Here, the goal is simply to try to connect to a "reasonable" number of peers, where the definition of "reasonable" is proportional to the total bandwidth available. Beyond

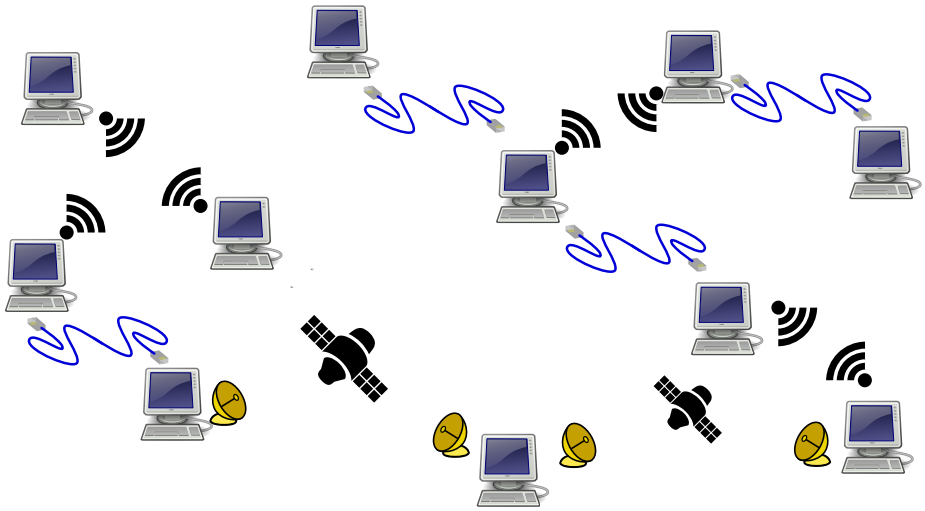


Figure 2.4: GNUnet peers may use different physical communication mechanisms below, and thus cannot expect universal connectivity or support for particular network topologies from the underlay.

that, no particular structure is imposed by TOPOLGOY. However, the user may require that the underlay restricts its connections to certain *friends*, which are nodes explicitly designated by the user to be trusted. The result is then a *friend-to-friend* network. However, the constraint that a peer should only talk to friends is not transitive.

Routing protocols provide structure on top of the underlay. Routing protocols include the bounded distance vector ("DV") subsystem and the R^5N DHT [EG11b]. The DHT may *request* that the ATS service establish particular links, i.e. because they are close in the distance metric used by the DHT, but the TRANSPORT service maybe unable to do so as the physical network underlay may not support a direct connection. Thus, in combination with the DHT the network is semi-structured: there is a desired structure, but the DHT has to handle arbitrary constraints. Various applications are built on top of the DHT, including in particular the GNU Name System [WSG14].

Using the DHT to discover routes, the CADET subsystem then composes routes to provide a global end-to-end transport. Thus, applications working above the CADET layer see the network through an abstraction that provides a virtual *clique* as the overlay structure. Finally, subsystems like MULTICAST then build additional more complex network topologies (trees, forests, etc.) on top of CADET.

2.4.3 Bootstrapping

A central question in topology construction is bootstrapping, that is how a peer can join the network and find its place in the overlay. Fully decentralized methods are generally quite expensive [GG08], especially for small emergent networks. GNUnet instead uses a diverse set of techniques to bootstrap:

- addresses of well-known peers are shipped with the software, and once connected, peers gossip addresses of other peers
- URLs with HTTP servers offering lists of peers are included with the software, and once connected, can be learned from gossip with other peers
- UDP IPv4 broadcast in the LAN is used to discover peers in the vicinity
- UDP IPv6 multicast in the LAN is used to discover peers in the vicinity
- WLAN broadcast is used to discover peers in range
- users can manually export and import addresses of peers via the command-line tool `gnunet-peerinfo`

Chapter 3

Key contributions

In this chapter, we will briefly present some of the key technical contributions used in the GNUnet system. The chapter is selective; overall, there are over 45 research papers and theses related to GNUnet listed on the GNUnet website[1], and that list is incomplete. To facilitate understanding, we again describe the GNUnet system bottom-up, starting with the transport layer.

3.1 Transport underlay abstraction

GNUnet was (probably) the first P2P overlay network that supported more than two methods of communication between peers as a way to bypass firewalls and censorship [FGR03]. This is achieved by the use of *transport plugins*. A transport plugin must implement an API that allows GNUnet's transport subsystem to exchange messages with another peer. The transport plugin defines the address format (i.e. an email address, or an IPv6 address and a port) as well as methods for sending and receiving messages using the respective addresses.

As different transport mechanisms have different semantics, the transport subsystem expects rather weak semantics from the plugin and also only offers the same weak semantics to the higher layers. Specifically, message delivery is allowed to be unordered and unreliable, but must be bi-directional.

3.1.1 Automatic selection and resource allocation

A given peer can support many transport mechanisms at the same time, and each mechanism can have zero or more addresses associated with it. A mechanism offering zero addresses might be an HTTP client: it can establish bi-directional connections to an HTTP server, but the client itself is not addressable. Given this, the transport subsystem must decide which transport plugins and which address(es) should be used for communication with which peers. Fur-

[1] `https://gnunet.org/bibliography?f[keyword]=2`

thermore, given resource limits, such as bandwidth, the peer needs to allocate resources to each connection.

We have implemented three different strategies for transport selection and resource allocation [WOG14]: a greedy heuristic, a linear program, and a machine learner using reinforcement learning. The greedy heuristic is very cheap and provides adequate solutions by assigning bandwidth proportional to the needs from the application, and selecting transport mechanisms primarily by latency. The linear solver can take significant CPU time on larger problems involving hundreds of connected peers, but by allowing for approximate solutions (i.e. within 95% of optimal) and by re-using the previous solution as a starting point the execution time becomes practical. However, the linear solver is unable to predict trends as it only uses the current state of the peer and thus is not forward-looking. Using reinforcement learning was attempted as a strategy to be both more predictive and less predictable for an adversary. However, so far the reinforcement learner is very slow to learn and to adapt to changing network conditions. Furthermore, our higher-level applications generally do not yet provide feedback. Even if they do, it is inherently challenging to compare the LP-solver with the reinforcement learner as they use different metrics (feedback vs. current performance goals set by the applications), and thus differences between the two approaches may be due to differences in the nature of the inputs provided by the applications and not due to qualitative differences between the methods.

3.1.2 Autonomous NAT traversal

Another challenge shared by virtually all modern P2P networks is NAT traversal. Various transport mechanisms support the most simple techniques for hole punching (manual via configuration, or using UPnP or PMP). NAT traversal using STUN [RWHM03] is not used so far, as it requires a trusted third party. Instead, we implemented *autonomous* NAT traversal which can traverse about 50% of the NAT devices we studied using ICMP error codes [MEGK10] and without a trusted third party. The main limitation of autonomous NAT traversal is that it typically only works to initiate a connection to a peer behind NAT if the initiator is not behind NAT.

3.2 Byzantine fault-tolerant routing

Routing algorithms are at the heart of any network. Like the Internet, GNUnet uses more than one routing algorithm. At the transport layer, bounded-distance distance-vector routing can be used to provide the illusion of higher connectivity, similar to the use of RIP [Mal93] in LANs. In this chapter, we will focus on *global* routing using GNUnet's DHT, which somewhat corresponds to routing with BGP [LR91] on the Internet. However, a DHT obviously does not provide a routing protocol in the style required by IP, so in Section 3.2.3 we will discuss how GNUnet realizes a transport in the style of SCTP [OY02] on top of the

DHT.

3.2.1 Secure network size estimation

Various P2P algorithms [BGK+09, EG11b] require estimates on the network size, that is the number of peers participating in the network. An estimate is generally sufficient, as the exact number fluctuates rapidly due to peers joining or leaving. Centralised networks can trivially provide the number of participants. A decentralized method for gossip-based counting is described in [vdBKM12], but like previous methods [MKM06, KPG+05, BJBS+08] it fails in the presence of malicious participants.

GNUnet provides an efficient, Byzantine fault-tolerant mechanism to estimate the size of the network [EPG12]. The first idea is to use the current time (rounded to, say, the hour) as the seed to generate a predictable but otherwise random hash shared across the network without prior communication. The time is rounded so that some clock skew between peers is easily tolerated. Next, each peer is identified by a public key, and the peers check how many leading bits of the hash of their public key are identical to the leading bits of the hash of the current time.

The key idea here is that the chance that the maximum number of matching leading bits across the network will increase with the size of the network: if the network doubles in size, there will on average be one peer with an additional leading bit matching. Thus, we can estimate the size of the network from the maximum number of leading bits that match. By repeating the protocol (e.g. once an hour) and averaging the last k-values, the peers can derive a reasonably good estimate of the network size.

But how can peers learn the maximum number of overlapping bits? To determine this global value, the peers[2] with the most overlapping bits must flood the network with a message saying how many bits they have overlapping, providing their peer identity as proof. But how can a peer know that he has the maximum number of overlapping bits and thus start the flood? The answer is that the number of bits is also used to determine *when* a peer should start the broadcast. The largest number (i.e. 256 bits) goes first, and a peer with 0 bits goes at the end of the time allotted for the round. Once a peer has received a flood message (usually with a larger number of overlapping bits than he has himself), he knows not to initiate the flood for this round and instead forwards the message he received.

To thwart malicious peers, peers will not forward messages that arrive too early for the number of overlapping bits provided. To limit an adversary performing a Sybil attack, the initiating peer must provide the result of an expensive proof-of-work calculation. This calculation is done by each peer once and specific to the respective peer's identity. Thus, only peers that have completed the proof-of-work are included in the network size estimate. This ensures that a computationally bounded adversary is limited in how much bigger he can

[2]There often is more than one peer with the same number of overlapping leading bits.

make the network appear to be. Malicious peers can also make the network appear smaller, but only by not participating in the protocol. In the worst case, an adversary controlling a separator in the overlay can cause the network size estimates to only count the peers in the subgraphs induced by the separator.

The protocol is highly efficient, as on average about one message needs to traverse each link in the network. Additionally, randomized small delays are used to avoid creating traffic spikes when the flood is triggered. We note that the final calculation of the network size from the number of overlapping bits is not simply $n = 2^b$, as one might casually expect; for the correct derivation we refer to [EPG12].

3.2.2 R^5N: A secure distributed hash table

R^5N is the Byzantine fault-tolerant distributed hash table (DHT) used in GNUnet [EG11b]. Like Kademlia [MM02], R^5N uses the XOR metric to establish distances between peers, as this way links between peers are used bi-directionally. Unlike Kademlia, R^5N does not make the assumption that the underlay is well-connected, as in GNUnet peers may be limited by firewalls on the Internet or use transmissions over a physical layer with limited range where the underlay thus does not use IP at all. Furthermore, the DHT may be subject to malicious peers participating in the network.

To address these challenges, R^5N changes Kademlia in three significant ways. First, R^5N uses *recursive* routing instead of the *iterative* routing from Kademlia. As a result, the initiating peer learns less about where the query ends, but also does not have to be able to connect to each hop. Second, Kademlia replicates at the α closest peers at the last hop. In contrast, R^5N replication does not fan-out at the end but during the routing process itself. In particular, if the network topology is not dense, there will be many locally-closest peers to a given key, as the α-peers that are closest to the key may be unable to establish direct connections in the underlay. R^5N's goal is to replicate at $O(\sqrt{n})$ of these peers, where n is the size of the network. By controlled branching during the routing process, a single request can end up at multiple of these local minima. Finally, R^5N is *randomized* to ensure that repeated requests take different paths. Specifically, before performing Kademlia-style greedy routing towards the key, each request must take $O(\log n)$ *random* hops through the neighborhood with the restriction that loops are forbidden. If the topology is a small-world network [WS98] (and thus $O(\log n)$ diameter), this results in R^5N finding data in $O(\sqrt{n}\log n)$ messages with $O(\log n)$ hops per message.

The randomization ensures that eventually a path is found on which there are no faulty peers, if such a path exists, and as the overall path length is limited to $O(\log n)$ the bandwidth amplification potential of malicious peers is also limited. As the DHT does not impose any strict structure on the underlay, churn also does not offer a means to perform denial-of-service attacks. Finally, the big-O performance is in line with other Byzantine fault-tolerant DHTs of this type [LLK10, MCB11], but preliminary experiments suggest that absolute performance is better [Sin14].

R^5N also introduces an improved DHT API. In addition to supporting the canonical `GET` and `PUT` operations, R^5N also supports *MONITORING* to allow applications to observe `GET` and `PUT` operations routed to or via a peer. This is not merely useful for diagnostics, but also enables new applications to trigger operations if they, for example, receive a `GET` and are thus responsible for a particular data item. In R^5N, each block stored in the DHT under a key is associated with a `type`, and developers can specify custom logic to be executed to verify the integrity of a received block based on its type and possibly the associated key. Thus, if well-formed blocks must have a particular format, be cryptographically signed, or if data is to be stored under the hash of the block, the respective custom logic can ensure that invalid blocks do not propagate through the DHT, and thus that applications will never receive ill-formed replies. This ability to validate blocks already in the network using custom validation code is used by both the GNU Name System (Section 3.3) and GNUnet's file sharing. R^5N can return more than one answer for a given query, and again the custom validation logic can determine if multiple replies are expected, valid or redundant. Here, a Bloom filter is used to probabilistically detect redundant replies. R^5N also allows applications to set a flag that triggers the *path* of requests to be tracked. If the flag is set, the receiver of a response will be able to see a possible path through the overlay network to the origin of the data. This is a key feature for the CADET subsystem presented in the next section.

3.2.3 CADET: Confidential ad-hoc decentralized end-to-end transport

GNUnet's confidential ad-hoc decentralized end-to-end transport (CADET) subsystem uses the R^5N DHT to find multiple *paths* through the overlay, creates an end-to-end encrypted *connection* over which applications can run many *channels* [PG14]. Each connection provides authenticated encryption and uses Axolotl [PM14] for asynchronous off-the-record messaging. The applications can set different delivery semantics for each channel, in particular channels can be out-of-order, in-order, reliable or unreliable.

CADET finds paths to peers by having each peer periodically store a self-advertisement into the DHT using its own public key to derive the key. As a value, the peer includes its own public addresses, if it has any. For CADET's use of the DHT, it is critical that the path tracking option is always enabled. A peer that wants to establish a connection to some other peer then performs a GET for the respective key. By combining the paths from the PUT and the GET, the initiator can determine one initial path from the source to the destination through the overlay. CADET also tries to establish a direct link if possible. Furthermore, each hop is told the remaining hops and the ultimate destination and can use its local knowledge about the network topology to possibly produce a shorter path. Local knowledge in particular includes information obtained from other paths that a peer is supporting. Finally, CADET tries to establish multiple paths to the destination and uses all of them in parallel to maximize throughput and resiliency.

As a result, CADET performs better after the network has been operating for a while: the peers learn the topology and optimize paths, resulting in subsequent paths being shorter, connections using a more diverse set of paths and thus operations becoming more robust.

Latency in CADET is comparable to typical WAN connections, the additional cryptography and the context switching between GNUnet subsystems only significantly increases latency within a LAN. However, with respect to throughput, CADET's extensive use of cryptography[3] imposes a more significant limitation as the system can become CPU-bound given good connectivity. In a LAN setup, we achieved "only" 5–15 MB/s of throughput. While this may be a serious limit for servers in traditional client-server architectures, we do not believe that this is problematic in collaborative P2P applications, as decentralization should allow application developers to balance any load across many peers, and not require one peer to directly handle hundreds of thousands of clients. Peers operating at the edge of the network also inherently do not have data center-style high-speed connections, and it will take a long time until the typical household bandwidth exceeds 40–120 MBit/sec. Thus, the 5–15 MB/s throughput is unlikely to be a bottleneck for the foreseeable future.

3.3 The GNU name system

The GNU name system (GNS) [WSG14] is used to name network services, systems or to identify users. GNS's design was informed by Zooko's triangle, an insightful hypothesis that says that name systems like to provide three features: secure names, global names and memorable names, but that it is only possible to provide two at the same time. We showed in [WSG13] that Zooko's triangle is correct if one defines "secure" to include a participating adversary with the only computational limitation that the adversary cannot break cryptographic primitives, and defines memorable names as those that are enumerable. For a weaker adversary model, NameCoin provides a design that "squares" Zooko's triangle [Swa11].

Given this fundamental design limit, GNS simultaneously offers the user all possible choices: the user can use cryptographic identifiers as names (forgoing *memorable* names), the user can stick to hierarchical names (forgoing *security* due to the need to trust the hierarchy), or the user can use petnames [Sti05] (forgoing *globality* due to the personalization from petnames).

Like in DNS, the owner of a GNS zone can *delegate* a label to another zone. However, while in DNS the definition of a zone "is the complete database for a particular pruned subtree of the domain space" [Moc87], in GNS a zone is simply a public-private key pair and a set of labeled records.[4] The *authority* for a GNS zone is simply the owner of the respective private key. Like with DNSSEC [3rd99], record sets in a zone are cryptographically signed. GNS al-

[3] We have link-encryption at the CORE layer, and then end-to-end encryption in CADET, both using AES and TwoFish and SHA-512.

[4] This makes it trivial to determine zone cuts.

ways keeps all records under the same label and the same zone in one response. This avoids the problem of TLSA records being shipped separately from A and AAAA records for the same host, as is the case with DNSSEC [HS12].

A key feature of GNS is that all public records are stored in the R^5N DHT and not at some authoritative server. To retrieve the record, the requester derives a hash from the label and the public key of the target zone. The DHT then delivers an *encrypted* response, which can only be decrypted if one knows both the label and the public key of the zone. However, while both query and response are thus not available in the clear, intermediaries (the DHT) can check that the signature over the record set data is correct and thus the record set contains the correct response. Thus, unlike DNS [GWE+15] the GNS avoids leaking meta data about user's name resolution activities to the network. However, GNS does allow a confirmation attack, so an adversary who knows the full request (label and zone) learns everything. Defeating this would require implementing private information retrieval [CKGS98], which is too expensive to be practical in this context.

Compared to X.509, the trust paths in GNS are explicit: given a name, it is obvious which zones operators need to be trusted. With X.509 the certified name typically bears no (obvious) relationship to the responsible certificate authority. Similarly, with DNS, out-of-bailiwick NS entries also make it difficult to predict which set of DNS servers may pose a security threat for a given resolution.

3.3.1 Revocation

GNS supports fully-decentralized and fast key revocation. Basically, when a user wants to revoke the key for a zone, the private key is used to sign a revocation message which is then flooded across the network. A proof-of-work is required to deter adversaries from flooding the network with useless revocation messages. Whenever two peers connect, Eppstein's efficient set reconciliation [EGUV11] is used to quickly compute the union of the two revocation sets. This ensures that temporary network partitions do not result in permanent divergences of the sets of revoked keys.

This style of key revocation is significantly *more* efficient and faster than X.509 certificate pinning, X.509 online certificate revocation checks [MAM+99] or even clients shipping with lists of revoked certificates. However, it depends on the existence of a P2P communication infrastructure across all participants to execute the flooding.

3.3.2 Conversation

GNUnet conversation uses the GNU Name System (GNS) as a public key infrastructure for a telephony application. A GNS zone is used as the *phone book*, and a `PHONE` record is defined in the zone to communicate to the dialer the address of the callee. Thus, GNUnet conversation demonstrates how to use GNS as a decentralized PKI.

GNUnet conversation uses CADET to establish a reliable channel to communicate control messages (i.e. to establish, suspend, resume and close connections) and a second unreliable channel for sending voice packets. The voice packets are encoded using the OPUS codec using a fixed bitrate encoding to minimize data leakage from packet sizes.

3.3.3 Protocol translation

GNUnet uses CADET's end-to-end communication ability and the ability to name hosts and services with GNS to allow users to operate TCP/IP on top of CADET. For this, the GNUnet VPN service opens a virtual network interface and routes all incoming traffic over CADET. Traffic destined to an IPv4 or IPv6 address is forwarded to a peer running a compatible EXIT service, where the traffic is then forwarded onto the normal Internet. This typically happens in conjunction with the protocol translation (PT) service, which supports IPv4/IPv6 protocol migration. When a peer receives a DNS reply in an address family not supported by the host, the PT service transforms the DNS reply to an IP address in the range of the VPN's virtual interface and informs the VPN about the intended destination address. Applications receiving the transformed DNS packet will then send their traffic to the virtual interface, from where the VPN forwards them to the intended destination.

Traffic for peers hosting services within the P2P network is handled in a similar way. When an application requests an A or AAAA-record and GNS instead obtains a `VPN` record which states that the desired service is hosted at a particular peer, GNS requests an IP from the VPN service and instructs the VPN service to forward the TCP/IP traffic to the hosting peer (based on the information from the `VPN` record). GNS then responds to the application with an A/AAAA record with the IP address from VPN service.

Part II

Contributions in depth

Chapter 4

Pluggable transports and resource allocation

This chapter is based on [FGR03, MEGK10, WOG14]. These papers were co-authored with Ronaldo A. Ferreira, Paul Ruth, Andreas Müller, Nathan Evans, Samy Kamkar, Matthias Wachs and Fabian Oehlmann.

4.1 Introduction

Reliable connectivity and reasonable application-level throughput (goodput) with low latency are important for the success of peer-to-peer (P2P) networks. However, the contemporary Internet rarely provides unrestricted communication: various parties try to restrict or shape traffic for political [FBH+02], economic or technical reasons [BER11].

Modern P2P networks focus on antagonising the restrictions built into the Internet infrastructure by obfuscating information flows and sending traffic via routes or mechanisms that are not easily restricted. Leading P2P designs have started to support multiple "pluggable" transport mechanisms, resulting in architectures that can leave a compromised or degraded medium of communication and switch to a different communication mechanism.

Peer-to-peer networks are typically overlay networks that are built on top of the existing Internet infrastructure. In an ideal overlay network, every node can communicate with every other node. However, this is not always the case with the modern Internet. Firewalls, network-address translation (NAT) devices, and dynamic IP assignment via DHCP create obstacles that global peer-to-peer applications need to overcome. One central design goal for a peer-to-peer framework must thus be to virtualize the network and give the application a view of a uniform address space and communication model. While it may not always be possible to guarantee connectivity from every node to every other node, the details about the implementation of the transport layer should clearly be hidden from the application.

It is desirable for a peer-to-peer system to offer transport protocols that can be used in spite of these circumstances. UDP and TCP can easily be blocked based on the port number associated with a specific application; on the other hand, some protocols, such as SMTP or DNS, cannot be conveniently blocked without entirely interfering with a significant portion of users.

One of the most important design requirements for a peer-to-peer system is thus the support for a wide variety of transport mechanisms. This can be achieved using a transport abstraction which supports a wide spectrum of possible transport mechanisms. For our work, the only requirement we make is that the mechanisms must offer bidirectional communication. Beyond that, they can be stream-oriented or record-oriented, reliable or unreliable, and low-latency or high-latency. Furthermore, two peers A and B may want to use different modes of communication on the same link. For example, suppose node B is behind a NAT box and cannot be reached directly via UDP or TCP. In a system with multiple transport protocols, A could initiate a connection by sending an email to B (SMTP) and then have B contact A via TCP, allowing A to continue further communication on a bidirectional TCP connection.

Being able to choose between different communication mechanisms creates new challenges as peers typically communicate with multiple other peers at the same time, and allocating resources for one peer may impact the quality of the connection to the others. Thus, transport selection for pluggable transports in P2P networks must consider not only performance and availability of the respective transport mechanisms, but also resource constraints and application-specific preferences.

This chapter presents solutions to the challenge of transport selection and resource allocation with a focus on the specific requirements of censorship-resistant decentralized networks. For security reasons, we require that peers do not make their resource allocation decisions based on unreliable information from other peers; to minimize information leakage, peers also do not exchange information about their resource limitations, neighbor set or even the results of the resource allocation process.

We present a formalization of the problem and three different approaches to solve the problem and evaluate these approaches with respect to usability of the approach and quality of the resulting solutions. We compare three solutions: a greedy heuristic, an algorithm based on linear constraint optimisation problem, and a method using reinforcement learning where agents learn to allocate resources while maximising social welfare.

4.2 Semantics of the Transport Abstraction

Node to node communication in peer-to-peer networks is inherently unreliable. In contrast to client-server architectures, node failure is part of the normal mode of operation. But even if nodes do not fail, the transport layer may be built on top of an unreliable communication protocol such as IP or UDP. The design question in this case is whether or not the transport layer implementation should

hide this fact and guarantee delivery if the other node is reachable. In other words, the question is whether or not the transport layer or core should provide reliable communication like TCP and hide the unreliability of the network, or if all network problems should be exposed to the application.

In peer-to-peer systems, it is better to expose the unreliability of the transport layer to applications or higher-level abstractions that go beyond the scope of a simple link-level transport. There are multiple reasons for this. Links in a peer-to-peer overlay network are bound to be even less reliable than the physical links that IP is concerned with. Connections with asymmetric bandwidth and P2P protocols that require forwarding messages to multiple other peers frequently force peers to drop messages. Congestion control would be difficult for a generic transport abstraction that has to deal with one-to-many or even many-to-many connections. Application specific solutions that can take the specifics of the protocol and potential security problems into account are needed. Another reason is that many applications may not require reliable communications; for example, a flooding search may send out 12 queries in parallel, and if one of them is lost on the transport layer, it is still possible that the remaining 11 queries will return a sufficient number of results. Adding retransmission on the transport layer in these cases merely increases the overhead without providing any major benefit.

The same rationale applies to the question of ordered delivery. Choosing the weaker semantics (no guarantee for order of delivery) makes the transport layer cheaper and more resilient. For example, an adversary that changes the message order or delays messages would have no impact. Of course, these less strict semantics also make the implementation of the transport over UDP (no order preservation) or SMTP (high latency) easier. The transport layer implementation may still use an underlying protocol such as TCP that has stronger semantics; this might happen, for example, because the network or the host configuration does not allow the use of cheaper protocols such as IP or UDP.

In GNUnet, the CADET layer (see Section 5.4) handles congestion control and can provide reliable and in-order delivery semantics. Applications and services that do not require these features can either disable reliability or congestion-control in the CADET layer, or do not use the CADET layer in the first place.

4.2.1 Security Considerations

An interesting security problem in peer-to-peer networks arises when malicious nodes advertise invalid or incorrect peer addresses. For example, it would be possible in Gnutella [Gnu02] to advertise `example.com` as a peer; even the port can be freely chosen in the advertisement. If peers spread this advertisement and frequently attempt to connect to this host, the peer-to-peer network could become a tool for a distributed denial of service attack as it may enable traffic amplification. On the other hand, without a central server, the ability of peers to advertise other peers cannot be avoided.

```
:0:
* \^X-mailer: GNUnet
/tmp/gnunet.smtp
:0:
/var/spool/mail/$USER
```

Figure 4.1: Example procmail configuration with "X-mailer: GNUnet" as the filter line and "/tmp/gnunet.smtp" as the name of the pipe.

Our solution to this problem is that every peer A that receives an advertisement for another peer B must check that the advertised address is valid by sending a PING message containing a *challenge* (a randomly chosen integer) to the advertised peer B. If B receives the PING, it responds with a PONG message which also contains the challenge, confirming that it can be reached under this address. Only after this protocol has been run should A notify other peers of B's existence. This prevents a malicious node M from advertising a non-participating third party T on the network since T would not properly respond to A's PING. M also cannot easily fabricate a PONG for T because the message sent to B contains a challenge which is unknown to M. While M has tricked A into sending a single message (the PING) to T, this cannot be used to seriously attack T since M had to send a message to A first. If M had sent the message directly to T, it would have caused an equal amount of traffic. The only gain that M has potentially achieved is that it was able to hide its identity from T.

4.3 Example: SMTP Implementation (Historic)

When GNUnet starts running, it loads all the transport modules defined in its configuration file. During this process, the initialization code of the SMTP transport opens a connection to an SMTP server (sendmail, qmail, etc.) that is running either on the local host or remotely; this connection will be used to send messages to the other peers. Observe that GNUnet does not establish a direct SMTP connection to the other peers, but relies instead on existing mail transfer agents (MTAs) to send the messages.

4.3.1 Sending Email

When the SMTP transport service receives a message from the GNUnet core, the message is extended with a header that contains the node identity of the sender and the meta-information provided in the parameters of `send`. The resulting message is base64 encoded, encapsulated according to the MIME conventions [FB96], and sent to the MTA over the pre-existing TCP connection.

Most MTAs store the mail on the drive before sending an acknowledgement to the client in order to ensure guaranteed delivery even after a crash. This disk I/O created by standard SMTP implementations that we use is a significant overhead for the SMTP transport, even though is not actually required for the GNUnet transport (since the GNUnet semantics only require unreliable communications). The MTA then resolves the destination address using DNS (MX record) and contacts the remote mail server, which again receives the message via SMTP and initiates delivery.

4.3.2 Receiving Email

In order for GNUnet to receive an inbound email, the mail must first be delivered to the local machine. If the local machine is the receiving host according to the MX record for the email address, this step is handled by the SMTP protocol. But in the case where the GNUnet node runs behind a NAT box, the mail will typically be stored on the mail server at the ISP. In this case, the host will periodically poll for new mail, for example using a POP client. Under this last configuration, the polling interval will be a major contributor to the delay in the SMTP transport. For GNUnet to work properly, we assume that one minute is a reasonable interval. Polling with POP can easily be automated using `fetchmail`, a tool that is available for most UNIX systems.

In many cases, this is not the only problem. Normally users will have only one email account available. Thus it is necessary to filter the inbound GNUnet messages from the other messages that are destined for the user. Since we do not want to tag all GNUnet emails with a uniform header (this would make it too easy for adversaries to filter and effectively censor GNUnet traffic), the advertisement for the SMTP address of the peer does not only contain an email address but also a *filter* line. The sender is required to add this line to the header. Since the receiver of the email specified which filter to use, `procmail` can be used to distinguish mails that have the appropriate filter line. The user can change the filter line whenever he wants; it will, of course, take some time to propagate the new address information into the network.

Finally, `procmail` needs to be informed of how to deliver the GNUnet mail to the GNUnet process. The easiest way is to use a named pipe (fifo). The user specifies in the GNUnet configuration the name of the pipe, and `procmail` writes the filtered mail into that pipe. The SMTP transport then reads the mail, decodes the base64 encoded body and forwards the message to the GNUnet core. An example `.procmailrc` configuration file is given in Figure 4.1.

4.3.3 Security considerations for SMTP

The primary security problem with SMTP is the potential for harassment of users. Other transports (UDP, TCP, HTTP) have this problem to a much lesser extent. While sending massive amounts of traffic can become an attack with every transport protocol, fairly moderate amounts of data can become a problem when sent to a user via email, especially if the user is not educated

enough to filter the spam. Still, it is possible to use SMTP as one possible transport mechanism for peer-to-peer networking. Since GNUnet is completely decentralized, a solution to the security challenge requires that peers be able to advertise email addresses of other peers on the network.

In order to prevent attacks, every peer first validates the advertisements before using the email address for actual transmission or advertising it further using the PING/PONG mechanism described before. This validation mechanism ensures that a malicious peer that sends an advertisement for an invalid (non-GNUnet) email address will trick the receiving peer into sending at most one small message to that address. The bandwidth that the adversary spends on sending advertisements is thus proportional to the amount of email that the victim receives. More importantly, the adversary is not anonymous. While the victim does not receive the mail directly from the attacker, it is clear that the attacker is the node sending the advertisements since no honest node will send advertisements without having received the PONG confirmations. Thus, it is possible to track down the attacker.

A more sophisticated attack involves mailing lists. The problem here is, that an adversary could subscribe to a mailing list and then advertise the address of the mailing list on GNUnet. Peers would send mail to the list and the adversary could send responses to the PING messages since he is one of the recipients. Since the peers can confirm that the address is valid, they would now start advertising the address, causing even more traffic for the list. In this way, an adversary could anonymously drown any open mailing list in unsolicited traffic. The solution to this problem is to ensure that GNUnet SMTP traffic will not be forwarded by any modern mailing list software. This can be achieved by making every GNUnet email look like a bounce message [MV96]. Bounce messages are used to notify the sender of an email about an invalid or unavailable receiver address. Since mailing lists often have the problem that one of its members is unavailable, it is safe to assume that bounces are always filtered.

4.4 Related Work

Encapsulating one networking protocol in another protocol and tunneling the traffic is a well-known technique that has been used for a long time (e.g. IP over X.25 [Kor83]). Asynchrony, high-reliability and universal availability have made the encapsulation of various services in email a popular choice [Boy02]. The high latency and the low efficiency of SMTP are for many applications not a problem. Research has instead mostly focused on addressing the security issues inherent in the protocol, mainly attempting to allow users to filter unwanted mail [PL98, SDHH98, AKC+00].

Infranet [FBH+02] steganographically hides traffic in HTTP requests to provide users with a high level of security. While HTTP itself has a fairly low overhead compared to other protocols, the steganographic encryption increases the traffic requirements by at least an order of magnitude. The peer-to-peer framework JXTA [Gon01] is another example of a networking protocol that can

encapsulate traffic in HTTP requests. JXTA allows the traffic to be encrypted but does not use steganography. JXTA supports peers that use network address translation (NAT [SE01]). If two peers that use NAT want to communicate, their traffic is routed via a peer that is globally addressable.

Another approach to establish connections with machines behind NAT boxes was described by Dan Kaminsky at DefCon.[1] Both hosts synchronously send the messages of the initial TCP handshake to the other hosts using a very small value for the TTL in the IP header. The NAT boxes see the outbound connection and start routing future messages. The small TTLs cause the handshake messages to be dropped at a router between the NAT boxes and thus the ICMP connection refused messages are never returned. The problem with this approach is that it still needs a way for both peers to synchronize. Furthermore, it assumes that the NAT box ignores ICMP TTL expired messages.

4.5 Autonomous NAT Traversal

Traditional NAT traversal methods require the help of a third party for signalling. This chapter investigates a new autonomous method for establishing connections to peers behind NAT. The proposed method for autonomous NAT traversal uses fake ICMP messages to initially contact the NATed peer. This chapter presents how the method is supposed to work in theory, discusses some possible variations, introduces various concrete implementations of the proposed approach and evaluates empirical results of a measurement study designed to evaluate the efficacy of the idea in practice.

A large fraction of the hosts in a typical peer-to-peer network are in home networks. Most home networks use network address translation (NAT) [EF94] to facilitate multiple computers sharing a single global public IP address, to enhance security or simply because the provider's hardware often defaults to this configuration. Recent studies have reported that up to 70% of users access P2P networks from behind a NAT system [CF06]. This creates a well-known problem for peer-to-peer networks since it is not trivial to initiate a connection to a peer behind NAT. For this chapter, we will use the term *server* to refer to a peer behind NAT and the term *client* for any other peer trying to initiate a connection to the server.

Unless configured otherwise (protocols such as the Internet Gateway Device Protocol [IW01] are counted as configuration in this context), almost all NAT implementations refuse to forward inbound traffic that does not correspond to a recent matching outbound request. This is not primarily an implementation issue: if there are multiple hosts in the private network, the NAT is likely unable to tell which host is the intended recipient. Configuration of the NAT is not always an alternative; problems range from end-user convenience and capabilities of the specific NAT implementation to administrative policies that may prohibit changes to the NAT configuration (for example, due to security concerns).

[1] `http://www.defcon.org/html/defcon-10/defcon-10-speakers.html#dankaminsky`

Since NAT systems prohibit inbound requests that do not match a previous outbound request, all existing NAT traversal techniques (aside from those changing the configuration of the NAT system) that we are aware of require some amount of active facilitation by a third party [J. 08, RMM10]. The basic approach in most of these cases is that the server in the private network behind the NAT is notified by the third party that the client would like to establish a connection. The server then initiates the connection to the client. This requires that the server maintains a connection to a third party, that the client is able to locate the responsible third party and that the third party acts according to a specific protocol.

The goal of this chapter is autonomous NAT traversal, meaning NAT traversal without a third party. Using third parties increases the complexity of the software and potentially introduces new vulnerabilities. For example, if anonymizing peer-to-peer networks (such as GNUnet or Tor [DMS04b]) used third parties for NAT traversal, an attacker may be able to monitor connections or even traffic volumes of peers behind NATs which in turn might enable de-anonymization attacks [MD05, EDG09]. Another problem is that the decrease in available globally routable IPv4 addresses [Hus10] will in the near future sharply reduce the fraction of hosts that would be able to facilitate NAT traversal.

4.5.1 Technical Approach

The proposed technique assumes that the client has somehow learned the current external (globally routable) IP address of the server's NAT. This could be due to a previous connection between the two systems or a third party having provided the IP address in a previous exchange. Note that we specifically assume that no third party is available at the time when the client attempts to connect to the server behind the NAT.

The first goal of the presented NAT traversal method is to communicate the public IP address of a client that wants to connect to the server behind the NAT. After the server is aware of the IP address of the client, it connects to the client (similar to NAT traversal methods that involve a third party).

The key idea for enabling the server to learn the client's IP address is for the server to periodically send a message to a fixed, known IP address. The simplest approach uses ICMP ECHO REQUEST messages to an unallocated IP address, such as 1.2.3.4. Since 1.2.3.4 is not allocated, the ICMP REQUEST will not be routed by routers without a default route; ICMP DESTINATION UNREACHABLE messages that may be created by those routers can just be ignored by the server.

As a result of the messages sent to 1.2.3.4, the NAT will enable routing of replies in response to this request. The connecting client will then fake such a reply. Specifically, the client will transmit an ICMP message indicating `TTL_EXPIRED` (Figure 4.2). Such a message could legitimately be transmitted by any Internet router and the sender address would not be expected to match the server's target IP.

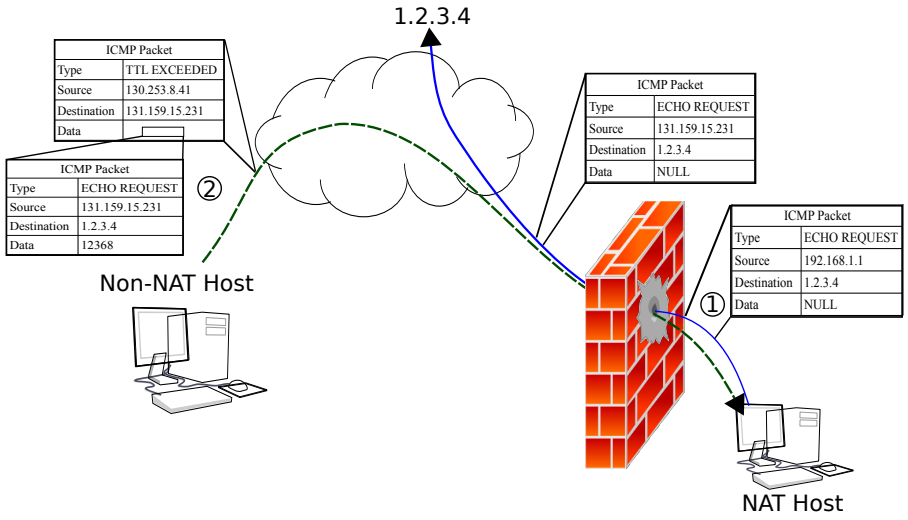


Figure 4.2: This figure diagrams the process of sending and receiving the fake ICMP messages for the server and client. In step 1, the server sends a fake ICMP request to 1.2.3.4 and in step 2 the client sends the matching reply. Note that this is a fake reply since the client *never* receives the ICMP request sent to 1.2.3.4 by the server. The important information contained in the actual packets is displayed for each step. The blue (solid) line shows the ICMP request path and the dashed (green) line shows the ICMP reply path.

The server listens for (fake) ICMP replies and upon receipt initiates a connection to the sender IP specified in the ICMP reply. If the client is using a globally routable IP address, this is entirely unproblematic and both TCP or UDP can be used to establish a bi-directional connection if the client listens on a pre-agreed port. In cases where there is no pre-agreed port, a port number can in most cases be communicated as part of the payload of the ICMP ECHO RESPONSE, which is typically not checked against the payload of the corresponding ICMP ECHO REQUEST by NAT implementations.

NAT-to-NAT Communication

Further complications arise if both the client and the server are behind NAT. In this case, often the client will be unable to transmit a fake ICMP response to the server due to restrictions imposed by the NAT implementation of the client. One possible idea for circumventing this problem is for the client to send the same message that the server is sending except with TTL 1 to its NAT. If the NAT accepts the packet despite the forged sender IP address it might theoretically generate the desired ICMP response and forward it to the external network. However, in practice we did not find NATs where generating the necessary ICMP message using a TTL of 1 works.

Even if the client is able to transmit the fake ICMP response, the next step — in which both the client and server are aware of the others IP address and now intend to establish a TCP or UDP connection — can still be complicated. The reason is that NAT systems can change the source port numbers of outbound messages. Without a third party, both client and server would have to guess matching source and destination port numbers as chosen (possibly at random) by their respective NAT implementations. Depending on the type of the NAT implementations (Full cone, restricted cone, port-restricted, symmetric), finding the correct port may take several messages. Client and server can reduce the total number of messages required by transmitting and listening on multiple ports in this phase.

Using UDP packets instead of ICMP ECHO REQUESTs

A possible alternative to having the sender transmit ICMP ECHO REQUESTs to a fixed, known IP address is having the sender transmit UDP packets to a fixed, known IP address and port. In this case, the client would again forge an ICMP TTL_EXPIRED message, only this time using the UDP format. The main disadvantage of this variation is that the sender has to guess the external UDP sender port number when faking the ICMP response. Since some NAT implementations randomly change those port numbers, the server might have to send UDP packets using multiple sender ports in order to give the client a sufficient chance at guessing correctly.

The main advantage of this technique is that the server no longer needs to send using RAW sockets, which may reduce the privileges required for the server. Note that the server still needs to be able to listen for the ICMP reply,

Table 4.1: Experimental evaluation of autonomous NAT traversal. "Echo-Server" lists the number of NAT implementations that allows (faked) ICMP TTL_EXPIRED replies to traverse the NAT in response to ICMP ECHO REQUEST messages. "Echo-Client" lists the number of NAT implementations that allow clients to transmit (faked) ICMP TTL_EXPIRED messages. "UDP-Server" and "ICMP-UDP-Client" give the same numbers when using UDP packets instead of ICMP ECHO REQUESTs. "Preserves Ports" indicates the number of implementations that preserve the sender's local port as the external port if possible. "Any server" lists the number of NATs where either the ECHO-Server or the UDP-Server work. Finally, "Two-Message Success" lists the number of NATs where autonomous NAT traversal (as a server) succeeds either with Echo-Server or with UDP with port preservation and hence only two messages are necessary to reach the server.
The totals differ as not all of the tests were run against all of the devices by the volunteers.

	Echo-Server	Echo-Client	UDP-Server	ICMP-UDP-Cl.
Full cone	0/4	1/4	1/4	1/4
Restricted cone	9/31	5/34	26/40	5/34
Port-restricted	37/56	2/71	82/91	2/71
Symmetric	2/3	2/5	3/5	2/5
Overall	53/103 (51%)	10/123 (8%)	121/149 (81%)	10/123 (9%)

	Preserves Ports	Any server	Two-Message Success
Full cone	0/4	1/4	0/4
Restricted cone	16/43	26/40	9/31
Port-restricted	72/98	83/91	43/56
Symmetric	6/6	3/5	2/3
Overall	100/162 (62%)	122/149 (82%)	62/103 (60%)

which requires RAW sockets on Linux. In the case of a full-cone NAT, using UDP packets instead of ICMP ECHO REQUESTs also has the advantage of establishing a port mapping which can then be used as an alternative method for contacting the peer.

Another difference between the two approaches is the possible payload that can be embedded in the response. With ICMP ECHO REQUESTs, the payload can be as big as the packet size permits and is hence only limited by the MTU of the respective physical network. Well-formed ICMP UDP TTL exceeded replies on the other hand can only contain 32 bits of payload: the ICMP TTL_EXCEEDED response contains the first 64 bits of the payload of the original IP packet. In those 64 bits, the 16-bit UDP checksum field and the 16-bit UDP packet length are unverifiable (for NATs that do not track extensive information about outgoing UDP packets) and can hence be used to transmit 32 bits of information to the server (in addition to the sender's IP address). With

our approach, either of these payload sizes is enough as we only transmit a port number in addition to the IP address.

4.5.2 Implementations

This section summarizes the three implementations of the proposed method that we have done so far. All of the presented implementations are freely available from the web pages of the respective projects.

Implementation in NAT-Tester Framework

Our implementation in the NAT-Tester framework was used to gather the data for this chapter. It transmits the various packet types (with or without payload) using raw sockets and uses `libpcap` to determine which messages were forwarded by the NAT. The client is currently available for W32 and Linux and must be run with administrator rights. This implementation is useful for researchers interested in exploring the various variations of this and other NAT traversal methods.

Implementation in `pwnat` tool

The `pwnat` tool[2] is a GNU/Linux-only stand-alone implementation of autonomous NAT traversal. After contacting the server behind the NAT, it establishes a channel with TCP semantics using UDP packets. It supports both client and server behind NAT (if one of the NATs allows the fake ICMP messages to be transmitted). This implementation targets end-users.

Implementation in the GNUnet Framework

Finally, we have created a re-usable implementation of the presented ICMP-based NAT traversal method in GNUnet. Since the use of ICMP requires the use of non-portable and often privileged system calls, this implementation is split into three main components:

ICMP server
: This component is a small program that provides the core ICMP-related functionality for the server. The code periodically generates the ICMP ECHO REQUEST message and also listens for incoming ICMP TTL_EXCEEDED responses. If such a response is received, it simply prints the IP address of the sender to `stdout`. If the ICMP also contains the 16 bit payload, it is interpreted as a port number and also printed.

ICMP client
: This component is a small binary which simply sends a single (fake) ICMP message to the IP address specified at the command-line. An additional

[2] `http://samy.pl/pwnat/`

argument can be given which will be interpreted as a port number to be transmitted in the payload of the fake ICMP response message.

Transport plugin
This component implements a GNUnet transport plugin [FGR03] and is thus specific to the GNUnet peer-to-peer framework. Depending on how the peer is configured, it controls ICMP servers or clients and ultimately establishes connections between peers.

Splitting the implementation into these three components has the advantage of minimizing the amount of code that must run with super-user privileges on POSIX systems (by installing the ICMP server and client with the SUID bit set). Furthermore, since the ICMP code is platform-specific, this makes it easier to manage this part of the code. Finally, this split makes it easy to share the platform-specific — but peer-to-peer network agnostic — ICMP code so that it can be used with other peer-to-peer applications. The implementation is thus suitable as a starting point for developers of any P2P network.

4.5.3 Experimental Results

We have evaluated the proposed autonomous NAT traversal techniques on a large number of NAT implementations. For this, we used TUM's NAT-Tester framework [MKC08a, MKC08b]. The framework consists of a public client that volunteers download and execute. The client then performs various tests against the local NAT implementation and reports the results back to the NAT-Tester server. This enables us to evaluate NAT traversal strategies against a wide range of NAT implementations. Detailed results are made public on the NAT-Tester web page.[3] In this section we will summarize the results based on the data available so far.

Table 4.1 summarizes which fractions of the NAT implementations evaluated so far support the proposed method for autonomous NAT traversal. We distinguish between behavior relevant for using autonomous NAT traversal from the point of view of both clients and servers behind NAT. We consider two cases: the case where the server uses ICMP ECHO REQUESTs and the case where the server transmits UDP packets. We also consider the extent of UDP port randomization which determines how efficient the second stage in the case of NAT-to-NAT communication would be. NAT implementations are categorized into the typical four types (full cone, restricted cone, port-restricted, symmetric) in cases where NAT-Tester is able to determine the type. NAT implementations that do not seem to fall into any of these categories are only included in the total.

The data shows that in virtually all cases NATs forward the faked ICMP messages for UDP (UDP-Server), but only in about half the cases for ICMP ECHO REQUESTs (Echo-Server). Furthermore, a significant majority of all

[3] `http://nattest.net.in.tum.de/`

NATs also preserve the source port (when possible), so the additional requirement of guessing the port for faking the ICMP response for a UDP message does not change the overall cost of the approach. Finally, NATs virtually always prevent their clients from transmitting the fake ICMP messages used by our clients (Echo-Client, ICMP-UDP-Client). Based on what we have seen from inspecting NAT configurations directly, the reason seems to be that NAT rules typically only allow ICMP packets for the states "NEW" and "ESTABLISHED" in the state machine [Pur04] — and the fake response falls into neither category.

4.5.4 Discussion

The proposed method of autonomous NAT traversal works well in the case of an unrestricted client attempting to initiate a connection to a server behind NAT. Here, in virtually all cases a single ICMP message by the client would be followed by traditional connection reversal [SFK08] which then reliably creates a UDP or TCP connection. In other words, there is no need for third parties to help initiate connections to NATed servers in this case.

On the other hand, if both systems are behind NAT, the proposed method rarely works and a third party is required. Assuming 70% of the peers in a network are behind NAT, this means that roughly 50% of all possible connections can be established using autonomous NAT traversal. However, even in the case where both systems are behind NAT a possible advantage of the proposed method remains; it is easy to create a simple, generic and fully stateless service that receives requests from NATed peers and generates fake ICMP replies to notify the server behind NAT. In this case, the payload of the ICMP reply would need to contain the original IP address (and likely source port number) of the client since the IP header of the faked ICMP response would now contain the IP address of the service.

4.6 The transport selection and resource allocation problem

Existing P2P networks like I2P [The13], GNUnet, SpovNet [BHMW08] and Tor [AM14] enable the use of multiple transport mechanisms, but so far they use rather simplistic processes like heuristics to decide which mechanism to use. This is problematic, as this decision can clearly have a significant impact on the quality of communication an application can provide.

In a P2P network, each peer communicates with a set of communication partners. The network defines a set of *transports* (e.g. TCP and UDP), peers can use to communicate with each other, but not every peer must support every transport. Transports provide *addresses* indicating how to connect to a peer. When a peer supports multiple transports, multiple addresses may be available to connect to this peer and even a single transport can provide multiple addresses (e.g. IPv4 and IPv6). Addresses can be located in different *network scopes* (e.g. LAN and WAN) with different *resource restrictions*. Resources available

on a system have to be distributed among communication partners. To not cannibalize other applications running on the same system, *quotas* may restrict the amount of resources available to the application. Different addresses may have different *properties* (e.g. delay and loss rate) based on the transport and the network scope of the address. Therefore metrics are required to compare and select the "best" address. Applications using the transport underlay to communicate with other peers have to specify *preferences* (e.g. low latency and goodput) to express which properties are important for good performance and which peers are important to communicate with. Applications should also be enabled to provide positive or negative *feedback* to indicate how satisfied they are with regard to the current performance.

A P2P network using a multi-transport approach should automatically select the "best" transport available for each communication partner, and continuously evaluate the performance of the chosen transport. The transport selection operation performed by each participant in the network has to (1) decide on a set of peers with which it will maintain connectivity, (2) choose a single address for each peer from the set of available addresses, and (3) allocate a certain amount of the resources while satisfying the resource constraints. Due to peers joining and leaving the network, inputs to the problem may change frequently. We expect peers to join and leave the network at frequencies in the range of seconds, whereas address properties may change within milliseconds. Whenever inputs change, the transport selection process may want to adjust the solution. The output of the address selection and resource allocation process is the set of addresses to use to communicate with other peers, containing a single address for each remote peer together with the resources assigned to each address.

4.6.1 Objectives for transport selection

In addition to considering application preferences and address properties, the selection algorithm should consider additional high-level objectives that transcend the preferences of an individual application. To provide a useful communication between participants, a minimum amount of resources is required for each connection. Thus, if an address is selected at all, at least a certain minimum amount of resources has to be assigned (**Usability**). Communicating with a larger number of participants increases the resilience of the P2P network. Therefore, the result should distribute resources over a range of peers instead of preferring communication with a tiny number of peers (**Diversity**). Resources should be allocated to peers according to their relative importance in the communication as expressed by the applications' preferences. So if a peer is valuable, it should get more resources assigned than a peer that does not contribute (**Relativity**).[4] Available resources should be fully allocated to allow participants to achieve maximum utilization when communicating (**Utilization**). Transports with high overhead should be avoided to minimize useless resource consumption and maximize application performance (**Austerity**). Allocations should exhibit some

[4]Naturally, the transport cannot tell if a peer is valuable, this is something applications have to determine, ideally in a way that is difficult for an adversary to game.

stability to minimize transport initialization overheads and provide predictable performance to applications (**Stability**). We assume that the P2P framework assigns specific weights for the sub-objectives to evaluate the overall quality of the peer selection, address selection and resource allocation.

4.6.2 Scope

For the approach presented in this chapter, we assume that only a single address per (selected) peer is to be determined, based on the idea that each connection creates inherent overheads (handshake, socket, buffers) and that establishing multiple parallel connections is thus inherently wasteful. This restriction is not a fundamental limitation since our solutions can be easily extended to permit this behavior. We further assume that the process of deciding which peers to communicate with is partially covered by the application, and the transport selection algorithm then decides on the address to use; the address must be initially provisioned with some minimum amount of resources, but in later rounds the transport selection algorithm may terminate the connection with the peer.

4.7 Design for Transport Selection and Resource Allocation

A component for automatic transport selection and resource allocation has to interact with both the underlying transport infrastructure and the applications using this transport infrastructure to communicate. We will now sketch three different solutions to find an "optimal" set of addresses and resource allocation with respect to the inputs provided by the transport underlay and higher layer applications and the user defined resource constraints. Each solver satisfies the requirements of this problem and has distinct advantages and disadvantages. A detailed analysis about the design and implementation of the solvers can be found in the thesis of Matthias Wachs [Wac15].

4.7.1 The heuristic solver

The first approach is a fast heuristic based on the idea to distribute resources roughly proportional to the importance a communication partner has for the high layer applications. The heuristic solver views the different network scopes as *buckets* of bandwidth and distributes the bandwidth in each bucket to peers in *relation* to how important this peer is for the applications.

The heuristic selects the "best" address available for a peer by comparing the *performance* properties; the focus here is on latency, but the stability of the choice is also considered. To ensure *usable* connections, the heuristic activates an address only if a minimum amount of bandwidth for all active addresses in this scope can be provided. Resources in the respective scope are distributed among the selected addresses by first assigning every address a minimum amount

of bandwidth to ensure *diversity* of connections and then distributing the remaining bandwidth among all addresses *relative* to the preferences the higher layer applications have specified (with respect to bandwidth) for peers. If not enough resources can be provided to maintain a connection, the heuristic may tell the underlay to disconnect from a peer.

4.7.2 The linear optimisation solver

To combine address selection and resources allocation in an integrated approach, the linear optimisation solver views the problem as a mathematical optimisation problem. In linear optimisation, problems are defined using a (linear) objective function to be maximized under a set of objectives, formulated as linear (in)equations [Kar72]. The address selection and resource allocation problem becomes a mixed integer linear problem (MILP) because a binary output is required to indicate if an address was selected (1) or not (0).

To formulate the problem as a MILP, one has to carefully formulate a set of linear constraints which ensure that the solution satisfies resource constraints. In our formulation, we distinguish between *feasibility constraints* ensuring that the solution is valid in the given domain and *optimality constraints* driving the solution towards the system objectives. As feasibility constraints we define constraints enforcing *diversity* (maintain a minimum number of connections), *usability* (minimum resources for active addresses), *scope* (one address per peer), *finite solution* (prevent unbounded solutions, quotas must be finite). To obtain a solution optimal with respect to the objectives defined in 4.6.1, we add optimality constraints optimizing for *utilization* (use resources available in network scopes), *austerity* (prefer transports with smaller overhead), *diversity* (establish connections with a larger number of peers), and *relativity* (distribute according to application preferences). The solver's running time can be reduced by exploiting the fact that the Simplex algorithm used to solve the problem can re-use an existing solution as a starting point if only the coefficients in the problem changed. As an output, the optimisation algorithm provides for each address a binary variable that models the selection of the address, and another with the amount of inbound and outbound bandwidth that was assigned.

4.7.3 The machine learning solver

The machine learning solver uses reinforcement learning [KLM96] to learn *good* address selection and bandwidth allocation strategies. The reinforcement learning (RIL) solver uses an autonomous *agent* per requested peer performing *actions* to learn or exploit the allocation strategy. To perform actions, the agent can increase and decrease bandwidth assigned to an address, switch to a different address or decide to do nothing. Based on the impact actions have with respect to the objectives defined in 4.6.1 and feedback received from applications, the agent receives a *reward* indicating if previously taken actions have improved the allocation or not. Based on this reward, the agent updates his allocation strategy. To achieve a global optimal solution for all peers, the solver uses a social

welfare algorithm to achieve good allocations for all peers. Contrary to previous solvers, this approach also supports *over-allocation* of the available resources. However, *allocated* resources might then ultimately not be *used* as applications may not generate enough traffic to fully utilize the allocations. Thus, over-allocation can be useful even if *over-utilization* creates significant penalties.

4.8 Implementation

To ensure the applicability of our proposed design for transport selection and resource allocation in practice, and to validate the design, scalability and performance of all proposed solution approaches, we implemented and experimentally compared the three solvers. The source code and the evaluation tools are available on our website[5].

We implemented the algorithms to execute independently from the transport underlay and the higher-layer applications. Specifically, the solvers run as separate (operating system) processes and both the transport underlay as well as the applications can interact with the solver component in a non-blocking way. This ensures that the rest of the P2P framework can operate independently from our component without having to worry about blocking operations or shared resources. This also facilitates the integration of the implementations into different P2P applications. The MILP solver relies on the GNU Linear Programming Kit (GLPK)[6], a Free Software package intended for solving linear optimisation problems.

4.9 Evaluation

To evaluate the proposed solvers, we used the production code in a simulation environment to evaluate our proposed design under controlled circumstances.

To evaluate the scalability of the solvers, we measured the running time and memory consumption of each respective solver by incrementally adding peers and addresses to the problem. Each time a new peer and addresses are added, our benchmarking tool requests the solver to find an allocation. In addition to changing the problem size by adding new peers and addresses, the tool requests *incremental* solutions after updating properties and preferences for peers and addresses already existing in the problem. Incremental solutions are particularly interesting for the linear optimisation solver since the solver can re-use an existing solution of the problem whenever the problem size does not change. The results of the solver scalability evaluation are presented in Section 4.9.1.

To evaluate the quality of the solutions provided by the solvers, we analyzed the quality of the solutions provided by the three different solver approaches.

[5] https://gnunet.org/git/

[6] `https://www.gnu.org/software/glpk/`

We designed multiple scenarios that peers might experience including fluctuating address properties and application preferences and evaluated the quality of the address selection and resource allocations produced by the solvers. To ensure that the heuristics were not somehow tuned specifically to the evaluation scenarios, we did not perform performance tuning of the solvers on the scenarios used for the evaluation. The results of the solver quality evaluation are presented in Section 4.9.2.

We used a desktop PC with an Intel Xeon W3520 quad core CPU at 2.67 Ghz and 24 GiB of memory running Ubuntu 14.04-AMD64 for all of the measurements. GLPK version 4.54 was used to solve the linear optimisation problems.

4.9.1 Solver scalability evaluation

The scalability of the different solvers largely depends on the number of peers, addresses and network scopes. Asking the MILP solver to produce optimal solutions for larger problems can be problematic — even for problems with just a few dozen peers and addresses the solver can take gigabytes of memory and hours of running time to prove the optimality of the solution. We thus allowed the MILP solver to terminate with an approximate solution of guaranteed quality (within 2.5% of the optimal solution). We also used a 10 second timeout; however, that timeout was then never reached in practice. With these restrictions, memory consumption for all approaches was relatively small (at the order of a few megabytes).

Figures 4.3, 4.4, 4.5 and 4.6 show the execution time in relation to the number of peers in the problem. Each peer always provides ten different addresses, equally distributed over five different network scopes. Properties of new addresses are initialized with random values. To evaluate the performance to solve an updated problem, the properties of 10% of all addresses currently in the problem are updated.

4.9.2 Solver quality evaluation

To evaluate quality of the solvers we used three different scenarios, modelled to represent the behavior of a file sharing application, a telephony application, and the case where both file sharing and telephony execute together. To evaluate the quality of the solutions provided by the different solvers the simulator collects information about the selected addresses and allocated bandwidth as well as the current properties and preferences as specified by the scenario generator. These inputs are then used to evaluate the quality of the allocation using a goal function similar to the objective function of the MILP solver. This goal function includes the utility of the current allocation, if bandwidth is assigned according to preferences specified by the applications (relativity), if connections to a larger number of peers are established (diversity) and if the addresses were selected according to the properties and preferences for these properties. In addition, the goal function includes a penalty if resource constraints are violated to penalize the reinforcement learning (RIL) solver for over-allocation.

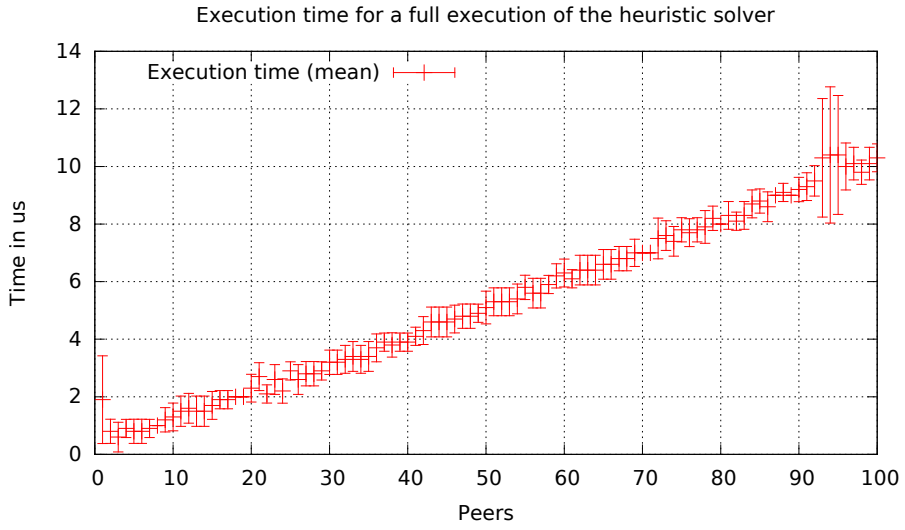


Figure 4.3: Execution time for the heuristic solver in relation to the number of peers.

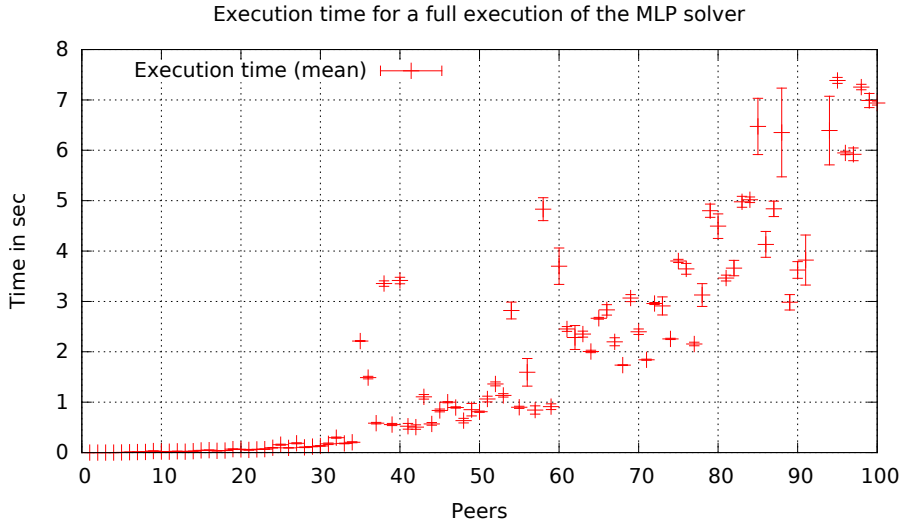


Figure 4.4: Execution time for the MILP solver to solve the problem from scratch in relation to the number of peers.

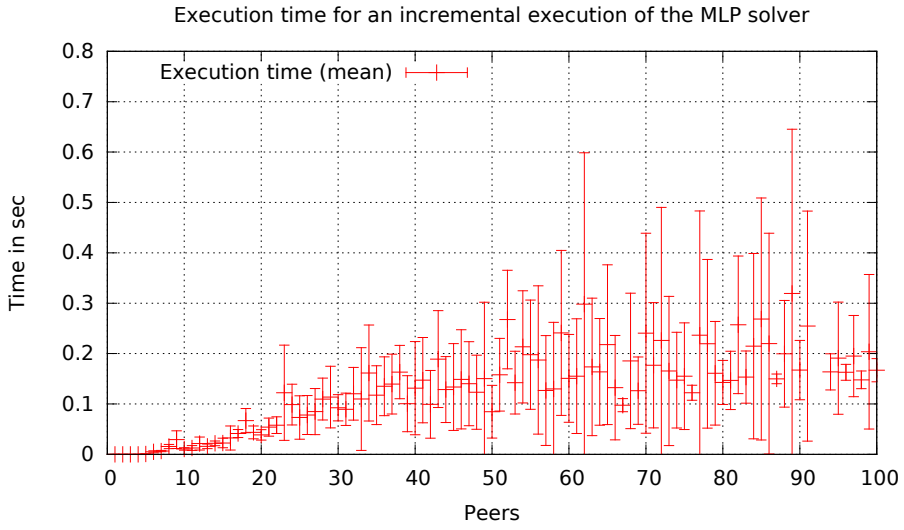


Figure 4.5: Execution time for the MILP solver to incrementally solve problem in relation to the number of peers.

Figure 4.6: Execution time for the reinforcement learning solver in relation to the number of peers.

To evaluate quality of the solvers we used three different scenarios, with two different durations (to observe the effect of learning). We run the simulations for 10 seconds in the *short* variant and 20 seconds in the *long* variant. We use two network scopes: scope n_0 with a large amount of bandwidth available, and scope n_1 providing only half the bandwidth. We have two neighbors p_0 and p_1, each with one address in each of the network scopes.

In the *throughput* scenario, we emulate an application trying to achieve a high throughput to one peer, and not caring about other peers. The application wants to maximize throughput to peer p_0, and has no concern for latency at all. We generate delay values for the addresses in n_1 to provide better latency properties than for the addresses in scope n_0, which provides more bandwidth but worse delay properties. After this setup we then begin to issue preferences in regular intervals of 500 ms with respect to bandwidth for p_0 with linearly increasing values and for p_1 with a constant low value to indicate our disinterest in this peer.

In the *latency* scenarios, we emulate an application requiring low latency values to communicate with both p_0 and p_1. We generate delay values for the addresses of neighbor p_0 and p_1 located in n_0 with random values between 20 and 25 ms and better delay values for addresses in scope n_1 with delay values between 10 and 15 ms for p_0 and 1 and 30 ms for p_1. We then begin to issue preferences for both peers with respect to latency, linear increasing values for p_0 and sinusoidal for p_1.

In the *mixed* scenario, we simulate two applications issuing conflicting preferences. For the addresses located in n_0, we create delay properties for both addresses with values between 20 and 25 ms, whereas for addresses in scope n_1, we create delay values between 10 and 15 ms for p_0 and values linear increasing between 1 and 30 ms for p_1. The values for p_1's address in n_1 were particularly chosen to make the solver switch to p_1's address in scope n_0. The application begins to generate preferences for p_0 to prefer maximize throughput and for peer p_1 to minimize delay.

Table 4.2 gives the results for the goal function for the different scenarios. The values are normalized in relation to the quality of the solutions produced by the MILP solver. The results show that learning is effective as the RIL solver's solution improves for longer runs, and it outperforms the heuristic for most scenarios.

4.10 Discussion

Our heuristic can compute the address selection and the resource allocation very fast, which is beneficial given frequent changes in the problem due to peers joining and leaving and updated address properties and application preferences; however, the greedy nature limits the quality of the solution and the heuristic does not benefit from the requirement to solve the problem repeatedly.

Treating the address selection and resource allocation process as an optimisation problem and using optimisation techniques has the advantage that the

solution found is always an optimal solution and objectives can be weighted according to the application's needs by adapting coefficients within the objective function. However, having to formulate the problem as MILP requires a careful design to formulate all constraints and the object function as linear equations. Requiring the output to contain binary variables makes solving the problem significantly more expensive since solving a mixed integer problem is NP-hard, while for linear programming polynomial time algorithms exist. For reasonable performance, it is important to keep in mind that Simplex typically produces feasible but suboptimal solutions quickly; thus it is important to bound CPU time with timeouts or reduce CPU consumption by allowing the MILP solver to terminate with an approximate solution of guaranteed quality.

An apriori definition of the objectives for address selection and resource allocation is a difficult task, especially as the requirements of applications may change over time. Furthermore, some of the constraints that were formulated are rarely "hard" constraints — an application that sometimes slightly overshoots bandwidth targets might be more desirable than an application that sticks to constraints and fails to deliver performance when it is critical. These challenges can be addressed using reinforcement learning which may predict future developments. In particular, a learning algorithm has the chance to adapt to the current observed utilization behavior of the application and can adjust its allocations accordingly. This can then reduce the amount of allocated but unused resources. However, reinforcement learning takes time for the adaptation, and thus naturally performs worse if evaluated under the same goal function as the MILP.

4.11 Conclusion

We have presented the design of a transport abstraction for peer-to-peer systems. The abstraction can support a wide range of underlying transport mechanisms and we have implemented service modules for UDP, TCP, HTTP, HTTPS, WLAN and Bluetooth, and historically also supported SMTP. While the benchmarks clearly show that SMTP is significantly worse in terms of performance, the service can still be useful to initiate connections and negotiate the use of

Table 4.2: Normalized quality of the solutions produced by the solvers.

Scenario	Heuristic	MILP	RIL
throughput short	0.905	1.00	0.513
throughput long	0.949	1.00	0.690
latency short	0.510	1.00	0.692
latency long	0.506	1.00	0.803
mixed short	0.547	1.00	0.367
mixed long	0.552	1.00	0.969

a cheaper service. We have addressed security concerns that arise with the use of SMTP and argued why a peer-to-peer transport abstraction should have unreliable semantics.

Even with pluggable transports, NAT traversal can remain an issue. We have shown how fake replies can enable autonomous NAT traversal in a number of cases. As with most NAT traversal techniques, this approach does not work for all installations. What is unusual about the presented method is that it works extremely well if only one peer is behind NAT and virtually never if both peers are behind NAT. Systems that require high NAT traversal success rates typically implement a number of traversal techniques and the presented approach extends the set of available methods by one that, if applicable, is cheaper and simpler than most of the existing techniques.

Based on an analysis of the challenges arising from the support of multiple transports under resource constraints, we presented three methods for transport selection and resource allocation for decentralized P2P networks supporting multiple transport protocols. We demonstrate that both reinforcement learning and constraint solving methods can deliver significant performance benefits over ad-hoc heuristics.

Chapter 5

Secure routing

This chapter is based on [EPG12, EG11b, PG14]. These papers were co-authored with Nathan Evans and Bartlomiej Polot.

5.1 Introduction

Existing IP networks are not suitable for secure, decentralized ad-hoc networking applications. IP routing requires trusted routers to assign structured addresses to their clients, and at higher levels BGP requires business contracts to negotiate peering relationships. Ad-hoc community networks require protocols that avoid the resulting overheads in planning and business negotiation, and eliminate the insecurities resulting from the dependency of network users on network operators.

This chapter presents *CADET*, a new algorithm for establishing robust end-to-end transport-layer connections in completely self-organized networks without a central authority. Starting with an arbitrary network topology (such as a wireless mesh, a physical LAN or even the peering relationships between autonomous systems), the algorithm uses the R^5N distributed hash table (DHT) to discover a redundant set of available paths, creates multiple, switched *connections* between the endpoints and then uses those to create a robust *tunnel* for secure, authenticated communication. Multiple applications can then multiplex TCP- or UDP-like *channels* over the tunnel.

A fundamental design principle of CADET is key-based routing (KBR). Addressing systems by their public key eliminates the need to use a network protocol to obtain a network address, and thus eliminates the use of insecure protocols like RARP or DHCP, or manual management processes by which users are assigned addresses out-of-band. A fundamental difference is that a system can then cryptographically prove its ownership of an address, as only it has knowledge of the respective private key. However, using public keys as addresses creates the problem of routing messages to the respective address, as public keys have no structure.

Many existing overlay networks use distributed hash tables (DHTs) to locate values by hash. While traditional DHTs assumed an underlying (IP-based) routing layer, more recent designs operate over an arbitrary mesh topology. CADET uses the R^5N randomized friend-to-friend DHT to discover paths to the target system. This DHT in turn requires a mechanism for estimating the size of the network, which we will present first.

The result can be viewed as a peer-to-peer implementation of the TCP/IP protocol: it allows a node in a decentralized and unstructured P2P network to find and connect to any other peer in the network, and to confidentially exchange authenticated data. Like SCTP, CADET supports transmitting multiple reliable and unreliable channels over a single connection.

5.2 Secure network size estimation

Individual peers participating in unstructured networks, such as Peer-to-Peer (P2P) networks, ad-hoc wireless networks and sensor networks, can benefit from knowing the size (total number of participants) of the network. Peers in unstructured P2P networks which know the network size can make intelligent decisions with respect to content replication, message routing and forwarding and the overall cost of operations. Additionally, nodes in a sensor network can use such data to gauge the health of the overall network, calculate on/off time to save energy, selectively route messages, and generate alerts.

This section describes the design, implementation and experimental results of a protocol that provides all peers in a structured or unstructured P2P network with an accurate estimate of the total number of peers in the network. The primary motivation for our work is the R^5N routing protocol presented later in this chapter; however, there are other P2P routing protocols which also explicitly require a network size estimate to tune parameters [EGH+03, MNR02]. Also, studies about deployed P2P networks [CCF10] could benefit greatly from knowing a good estimate of the analyzed network.

The focus of our design is to provide security in the context of an open and completely decentralized network architecture. While it would be possible to strengthen the security of our design with trusted centralized services — for example by preventing a Sybil-attack with a centralized registration requirement — our design does not require a centralized authority and is intended to provide security in the presence of actively participating adversaries. A key difference to existing proposals is that in our design there are no peers with special roles in the process; this eliminates the possibility of malicious peers abusing such roles.

A central goal of our design — which is generally not satisfied by many other network size estimation algorithms [BJBS+08, KPG+05, MLMKG06] — is that all peers are supposed to participate in calculating a network size estimate at roughly the same time and obtain the same result. This is achieved by a controlled flood of the network with the size estimation information, costing $O(|E|)$ messages per round. In practice, the constant factor is typically between

one and two; in other words, the algorithm can be expected to only generate $|E|$ messages per round.

The basic idea behind our algorithm is to flood the network with the identity of the peer whose identity is closest to a particular key T. Each peer's identity is generated when the peer starts the first time. The key T is not chosen by any peer but instead is generated from the start time S of the current round. Despite using time, we specifically do not assume that the clocks in the network are closely synchronized; our protocol ensures that in the worst case individual peers with significant clock skew only cause a bounded amount of additional network traffic.

Our protocol considers many other important networking issues as well. The protocol is very efficient as it requires only $O(1)$ state per peer and does not require peers to establish new connections (we assume the network graph is connected). The amount of work required by each node is based only on the number of edges of the node, so the load between peers is typically balanced. Given that our protocol floods the network with size estimation information, peers randomly delay messages to avoid spikes in network traffic. Finally, our design handles network churn well, and allows the system designer to trade-off computational efficiency for security and bandwidth for accuracy.

5.2.1 Related Work

Algorithms for estimating the size of a P2P network can be categorized into algorithms for structured overlays, which typically exploit statistical properties of an existing routing table from a DHT, and algorithms for unstructured overlays, which make no assumptions about the structure of the underlying network.

Network Size Estimation for Structured Overlays

Structured overlays construct routing tables at each peer according to particular rules that enable efficient routing of messages to the peer with the "closest" identifier with respect to a given key [MM02, RD01]. In these structured overlays, the distance to the nearest neighbors in the routing table can be used as a first network size estimate as it correlates with the network size [Pol10].

As node identifiers are often not perfectly uniform, searching the structured overlay for the closest node to various randomly selected keys can be used to get accurate network size estimates [Pol10]. Given a DHT routing algorithm with a typical cost of $O(\log n)$, network size estimation for all nodes using this method would be $O(n \log n)$. When compared to the method presented in this chapter, a key disadvantage of existing methods for structured networks is that they rely on the security of the underlying routing algorithm; actively participating malicious nodes have thus the potential to significantly skew the network size estimate. Furthermore, for any of the structured methods that we are aware of, different nodes will virtually always compute somewhat different network size estimates.

Network Size Estimation for Unstructured Overlays

Several algorithms for unstructured overlays are based on sampling. Examples include Sample & Collide, Random Tour and Hops Sampling. In Sample & Collide [MLMKG06], each peer starts bounded random walks to sample random peers in the network and uses the collision information and the birthday paradox to estimate the size of the network. In Random Tours [MLMKG06], a message tours the network until it reaches the initiator; the size estimate is then computed based on a counter in the message that was incremented by each peer on the tour. Hops Sampling [KPG+05] works by flooding the network with a message containing a hop count. Peers report back to the initiator with a probability inverse to the hop count they received. The network size estimate is then the sum over all distances of the number of replies received for a particular distance divided by the reply-probability for that distance.

As described, these methods generate results for just one peer in the network, resulting in a high amount of bandwidth used overall (assuming each peer requires an estimate). Also, different peers will have potentially significantly different network size estimates. While these approaches do not assume a particular network structure for routing, they do still make implicit assumptions about the structure of the overlay topology and may significantly underestimate the network size if the overlay topology happens to have a structure that is unfavorable to the algorithm. For example, a circular topology would result in a network size estimate of n^2 for Sample & Collide.

Other algorithms, such as Gossip-based Aggregation [JMB05], achieve a somewhat more uniform estimate for all participating nodes at the cost of sensitivity to node failures. Gossip-based Aggregation starts with one peer setting a local state to 1 while all other peers set their local state to 0. Peers continuously connect to randomly selected peers, and exchange states in pairs. Each peer then replaces its state with the average of both values. After a predefined number of iterations, all peers are supposed to end up with a value close to $1/n$ where n is the size of the network. A method that addresses the problem of who should set the state to 1 has been proposed [SGH08], but only works in certain structured networks and retains other shortcomings of this approach, including high vulnerability to malicious peers. In [vdBKM12] a much more efficient gossip-based method is introduced which uses aggregation and beacons to achieve fast convergence, high precision and is able to handle churn; however, it still offers no security.

A special case is the method proposed in [BJBS+08] which attempts to produce a network size estimate using only "local" information. The idea behind this algorithm is to observe the number of new neighbors discovered in a breadth-first search of the network and estimate the network size based on the growth of this function. The authors claim to obtain accurate results with a breadth-first search of depth three, which makes this a "local" method. However, the way they constructed the topologies for their experiments does not seem to properly model the structure of actual networks. We were unable to reproduce their results on other network topologies.

The accuracy and performance of various size estimation methods for unstructured networks are compared in [MKM06] using simulation. The authors identify the Sample & Collide method as the strongest algorithm and state that it requires about 50 million messages in a random-graph topology of 100,000 nodes for an accuracy of ±4%. It should be noted that this is the overhead for an individual node to obtain an estimate; if each of the 100,000 nodes were to run the Sample & Collide protocol, it would take 5 *trillion* messages to achieve this degree of accuracy.

None of these papers mention concrete implementations or discuss security concerns. Furthermore, all of them are clearly vulnerable to malicious participants. For example, in the case of sampling-based algorithms, malicious participants can manipulate walks that pass through them (allowing virtually unbounded manipulation of the network size estimates) or achieve a significant multiplier ($O(\sqrt{n})$) to their network bandwidth in a denial-of-service attack by continuously initiating size estimation requests. Similarly, an active adversary can manipulate the exchanged values in gossip-based methods to change the size estimate in any direction.

5.2.2 Our Approach

We generate node identifiers by hashing the public key of the respective node. We will assume that the hash is large enough (say 512 bits) for its finite length to be practically infinite for the purpose of the protocol. Node identifiers for benign nodes should therefore be statistically equivalent to random numbers from a uniform distribution. Furthermore, nodes are able to cryptographically sign messages using their respective private key.

Similar to the network size estimation algorithms for structured overlays, our network size estimation approach is based on the largest number of leading overlapping bits between any node identifier and a random key:

Theorem 1. *Let $\overline{p}$ be the expected maximum number of leading overlapping bits between all n random node identifiers in the network and a random key. Then the network size n is approximately $2^{\overline{p}-0.332747}$.*

Proof. Let X be the random variable for all n identifiers and let X_i be the number of overlapping bits for an individual random node identifier i.

The probability that a single random node identifier i overlaps with at least α bits with a random key is

$$P(X_i \geq \alpha) = 2^{-\alpha}. \tag{5.1}$$

Then, the probability that a single random node identifier overlaps with less than α bits with a random key is

$$P(X_i < \alpha) = 1 - 2^{-\alpha}. \tag{5.2}$$

The probability that the maximum number of leading overlapping bits for all n random nodes is strictly less than α is

$$P_n(X < \alpha) :=: P\left(\bigwedge_i X_i < \alpha\right) = (P(X_i < \alpha))^n = \left(1 - 2^{-\alpha}\right)^n. \tag{5.3}$$

Then $E_n(X)$, the expected maximum number of leading overlapping bits between n random node identifiers in the network is:

$$\begin{aligned}
E_n(X) :=: \sum_{\alpha=0}^{\infty} \alpha \cdot P_n(X = \alpha) &= \sum_{\alpha=1}^{\infty} P_n(X \geq \alpha) \\
&= \sum_{\alpha=1}^{\infty} (1 - P_n(X < \alpha)) = \sum_{\alpha=1}^{\infty} \left(1 - \left(1 - 2^{-\alpha}\right)^n\right) \\
&= \sum_{\alpha=1}^{\log_2 n} \left(1 - \left(1 - 2^{-\alpha}\right)^n\right) + \sum_{\alpha=\log_2 n+1}^{\infty} \left(1 - \left(1 - 2^{-\alpha}\right)^n\right)
\end{aligned}$$

Suppose n is sufficiently large such that we can use $\lim_{n\to\infty}(1 - \frac{x}{n})^n = e^{-x}$. By substituting $\beta := \alpha - \log_2 n$ and $\gamma := \log_2 n - \alpha$ we then get:

$$\begin{aligned}
E_n(X) &= \log_2 n - \sum_{\gamma=0}^{\log_2 n - 1} \left(1 - 2^{\gamma - \log_2 n}\right)^n + \sum_{\beta=1}^{\infty} \left(1 - \left(1 - 2^{-(\beta + \log_2 n)}\right)^n\right) \\
&= \log_2 n - \sum_{\gamma=0}^{\log_2 n - 1} \left(1 - \frac{2^\gamma}{n}\right)^n + \sum_{\beta=1}^{\infty} \left(1 - \left(1 - \frac{2^{-\beta}}{n}\right)^n\right) \\
&\approx \log_2 n - \sum_{\gamma=0}^{\log_2 n - 1} e^{-2^\gamma} + \sum_{\beta=1}^{\infty} \left(1 - e^{2^{-\beta}}\right) \\
&\approx \log_2 n - 0.521865 + 0.854613 = \log_2 n + 0.332747
\end{aligned}$$

Thus, for sufficiently large values of n,

$$E_n(X) \approx \log_2 n + 0.332747. \tag{5.4}$$

□ □

Given Theorem 1, the key remaining challenge is thus to efficiently and securely find a closest node identifier (with distance measured in terms of leading overlapping bits) to a random key in an unstructured network.

In our design, all nodes in the network periodically participate in a global network size estimation operation at a frequency of f. Each round results in all peers learning a discrete approximation p (the number of overlapping leading bits for a particular random key) for $\overline{p}$ (the theoretically expected number

of overlapping leading bits). The specific frequency f is chosen based on the expected level of network churn and the desired accuracy. f is a design parameter and fixed in the implementation. The results from the last k iterations are averaged locally by each peer to obtain an approximation $\tilde{p}$ for $\overline{p}$. A standard deviation can also be computed if an estimate for the error of the size estimate is desired. Furthermore, the current p value is used by the protocol as a parameter to (slightly) improve the performance for the next round. We will refer to the number of overlapping leading bits from the previous round as p'.

Generating a random "key"

Given a frequency f, the random target key T for each round is generated by hashing the start time S, which is the absolute UTC time at times that are zero modulo f. For example, if $f = 1h$, then a fresh key could be generated every hour by hashing "DD-MM-YYYY HH:00:00". Using this method, all peers will generate exactly the same key at (roughly) the same time. Generating the key this way has the advantage that it will be known to all peers without communication and that malicious participants cannot influence the process. However, it should be noted that while the keys satisfy the statistical properties of being random and uniform, it is trivial to compute them in advance.

Our method requires all peers to calculate the current key T at the respective start time S. The network size estimation protocol's goal is to communicate to all peers an identity I_T of a peer with the largest proximity p with respect to T. More specifically, all peers are supposed to learn one of the closest peer identities I_T between time S and time $S + f$. Given I_T, each peer can then calculate p, the average $\tilde{p}$ of the p-values from the last k rounds and finally the current network size estimate $2^{\tilde{p}-0.332747}$.

Note that p is a discrete value representing the number of leading matching bits between the key T and a peer's identity. As such, it is quite likely that many peers have identities with the same number of leading matching bits and hence the same proximity p. Our protocol deliberately ignores all bits after the first mismatch to improve performance; if multiple peers have the same proximity score, it does not matter which of these equivalent identities is propagated as they will all ultimately result in the same proximity estimate p.

Starting the Flood

Our protocol essentially floods the network with the identity of a closest peer I_T. If only the identities of closest peers are propagated, this operation would create less than $2|E|$ messages (up to two per edge in the network). The challenge is to avoid creating significantly more than $2|E|$ messages, which is difficult since in an unstructured network a peer with the closest identity I_T cannot be certain that there is no other peer that is closer to T. We address this problem by delaying the flood based on proximity.

First, each peer evaluates its own proximity x with respect to T. How close the peer is to T is then used to determine how soon the peer will initiate the

flood of the network with a message claiming that he is the closest peer. We use the previous network size estimate as a guide to time the release. Specifically, given a proximity of x leading overlapping bits and a network size estimate p' from the previous round, we use the following function to determine the time when a peer starts to flood the network:

$$r(x) := S + \frac{f}{2} - \frac{f}{\pi} \cdot \arctan\left(x - p'\right) \tag{5.5}$$

Using this function, if the peer's proximity x is equal to the proximity of the last iteration, the peer floods the network at time $S + \frac{f}{2}$. If the peer's proximity is 0 bits, the peer floods the network close to time $S+f$, which is at the end of the time interval for the current round; if the peer's proximity is significantly higher than p', the peer floods at the beginning of the round, which is close to time S. It should be noted that since $\lim_{x \to p'} \frac{\partial r}{\partial x}(x) = 1$, Equation 5.5 maximizes the difference between release times for nodes with proximities that are close to the previous number of overlapping bits p' and as such minimizes the chance of two peers releasing floods for different network size estimates around the same time — assuming the network size estimate did not change significantly.

Processing the Flood

Peers that receive the resulting network size estimation messages first perform a series of validation steps before continuing to forward the message. First, each peer checks if a notification from a closer peer has already been received for round S. Messages with proximity scores equal to the currently known best score for the current round are simply discarded. Messages with lower proximity scores should only occur if there is significant clock skew, and are answered immediately with a message indicating the higher proximity score. If the message contains a higher proximity than what was previously known for the current round, the peer checks if the proximity p of the given message justifies receiving it at the current time. If not, further processing is delayed until the local peer's time is past $r(p)$. Finally, before forwarding and further processing, the format of the message is validated (this is discussed in more detail in Section 5.2.2).

Assuming the message validates, the peer then proceeds to forward it to all of its neighbors. For each neighbor, the message is forwarded with a peer-specific random delay. If a peer receives a message with an equivalent proximity score during the delay, the transmission is canceled. As a consequence, the delay helps to both avoid an explosion of messages on the network in a tiny amount of time, and to improve the chances of traversing each edge in only one direction per link (as it decreases the chances of equivalent messages being sent in both directions at the same time).

The permissible delay L is calculated using the time difference between $r(p)$ and $r(p-1)$ divided by an estimate of the network's diameter. The network diameter D is estimated using the maximum of the hop counters in the network size estimation messages from the previous k iterations. For each neighbor, the

peer then applies a delay chosen uniformly at random from the interval $[0, L)$ where L is defined as

$$L := \frac{r(p-1) - r(p)}{D}. \tag{5.6}$$

As a result, each peer in the network is expected to receive a proximity notification with proximity p before any peer with notifications for proximity $p-1$ would even begin to flood the network. Naturally, there are various causes that could increase the number of messages above $|E|$ in a real-world network; for example, different system times between peers, high network latencies, and situations in which all peers have unusually high distances to I_T can increase the total number of messages. However, all benign peers that form a connected component are guaranteed to eventually receive I_T. Furthermore, given that the number of bits in the key is a small constant, the total number of messages can never exceed $O(|E|)$ per round.

Joining the Network

A peer that is freshly joining the network lacks results from previous rounds for network size estimation. In order to bootstrap the protocol, each peer starts with a network size estimate based on its own key in relation to the key T from the previous round, and a network diameter estimate of one. Whenever a connection between two peers is established, they exchange the network size estimation result from the previous round (and the current round if their local time is past $f(p)$). As a result, all nodes can always be expected to use the same value for p' in Equation (5.5).

If, as a result of this exchange, one side has to increase its network size estimate for the previous round, it floods its neighbors with the result from that round as well. If information about the previous round is flooded in this fashion, the artificial delay limit L is set to an implementation-defined small constant (we use 50ms). Note that only flooding of information about the current round S and the previous round $S - f$ is permitted. As a result, all nodes can always be expected to use the same value for p' in Equation (5.5).

It is not true that all peers will calculate the same value for L based on Equation 5.6. The hop counters in the flooded messages can clearly differ between peers, potentially resulting in a different estimate for D. Furthermore, since the network size estimate given to applications is based on the last k values, application-level network size estimates may differ between established and recently joined peers as well. If the latter is unacceptable, peers establishing connections could propagate the last k network size estimates.

Proof of Work

The presented design is vulnerable to an adversary that creates fake identities (Sybil attack [Dou02]). Such an adversary could create identities that are "close" to the respective key for each time $S + \mathbb{Z}r$. By flooding the network with the

respective messages at the right time, the adversary can then make the network appear to be larger than it is.

Our design defends against this attack by requiring a proof of work [SL03] for the identity of the peer as part of the network size estimation message. Specifically, we require the originator to produce a value with a W-bit hash collision with the peer's identifier, and a cryptographic signature to demonstrate that the identifier was derived from a valid public-private key pair.

The complete message format for the network size estimation messages is described in Figure 5.1.

Offset	Contents
0	Message header magic code
4	Hop-Count (updated at each peer)
8	Signed data header magic code
16	Time S of the round
24	Proximity p in bits
28	Public key (256 bit EdDSA)
60	Proof-of-work
68	64-byte EdDSA signature (signing bytes 8–67)

Figure 5.1: Message format for the network size estimation messages. The proof-of-work is a 64-bit number such that the hash of the concatenation of the public key and the proof ends with W bits of zeros. The claimed proximity p is redundant (it could be calculated from hashing the public key and S). However, by including p the moderately expensive cryptographic hash calculation of checking the proof of work repeatedly can be avoided if p is not larger than the current local estimate already known to the peer for time S. A signature is included to ensure that the public key itself is well-formed and derived from a private key.

5.2.3 Security Analysis

For our security analysis, we assume that an active adversary is participating in the P2P network. The adversary is allowed to control a certain percentage of colluding malicious nodes in the network. Individual malicious and benign nodes are assumed to have the same amount of computational resources; all nodes are assumed to have sufficient bandwidth to participate in the protocol in the absence of an attack.

We can imagine three different high-level goals an adversary may pursue with an attack. First, an adversary may try to cause nodes to significantly underestimate the size of the network. Second, an adversary may try to cause nodes to significantly overestimate the size of the network. Finally, an adversary may want to use the protocol for a denial-of-service attack where the P2P network

uses significantly more traffic for network size estimation, possibly causing other components of the system to be left with insufficient bandwidth.

The best method for an adversary to cause peers to underestimate the size of the network is to not participate in the protocol. If the adversary controls $X\%$ of the network, that will cause the protocol to underestimate the size of the network by $X\%$. Furthermore, if the adversary is able to control an ϵ-separator of the network graph [KL70, KSS08, Mar06] (removing an ϵ-separator from a graph reduces the size of the largest remaining connected component to ϵn), then the overall network size estimate would be reduced to less than ϵn for all nodes in the network. Given that in all of these cases the network size estimate would correspond to the size of the network after the removal of the adversaries' nodes, this attack is not particularly disruptive in relation to the strength of the adversary. Thus, an adversary cannot make the network appear significantly smaller than it is.

If the adversary wants to make the network look larger by M nodes, it needs to first compute (and store) M public-private key pairs. Then, at every time interval f, the adversary needs to compute collisions costing an additional $O(2^W)$ to generate the required W-bit collision. Actually joining the network with M "fake" peers is not required. If W is chosen so large that the adversary cannot solve the problem at frequency f, it is still possible for the adversary to cause an increase in the network size estimate by solving the problem every $c \cdot f$ (for an appropriate choice of c based on the adversaries computational resources), which would still affect the computed medium-term averages computed for subsequent intervals. Using such an attack, an adversary can make the network appear significantly larger than it is, as long as the adversary has access to sufficient computational resources.

Finally, for a denial-of-service attack, an adversary would first generate additional identities and generally perform the same steps as for increasing the estimated network size. Now, suppose the adversary has created m identities that are closer to the current key than the closest actual peer in the network by $1 \ldots m$-bits respectively. Then, just after the identity of the peer that is actually closest to the key has been broadcast to the network, the adversary can cause m additional broadcasts by transmitting its m "fake" identities in order of increasing proximity to the key. Each time, the network will presume that a closer peer was "late" with its transmission (for example, due to clock-skew or network latency) and broadcast the update. If the network is already of size n, the expected one-time cost for the adversary to create m such identities is $O(n2^m)$; the attack then requires an additional $O(m2^W)$ operations for the hash collisions at frequency f. Therefore, if we neglect the high one-time cost of computing identities, the adversary can cause $|E|$ traffic on the network at the cost of $O(2^W)$ computations.

Analytical Worst-Case Analysis

The following scenario describes the theoretical worst case in terms of bandwidth consumption by the protocol. Without loss of generality, suppose a 512-bit hash

function is being used. Then, the worst-case network would for $\mu \in [1, 512]$ have exactly one peer with μ matching bits with the target key T (in each round); all other peers in the network would have zero matching bits. The peers that do have matching bits should be connected to the main network via a long chain (with larger distances for peers with larger μ), causing the network diameter D to be large. (As a result, the algorithm will calculate $L \approx 0$.) Peers with μ matching bits should furthermore have (or pretend to have) a late system time that causes them to transmit effectively at time $S + \mu\epsilon$; all other peers have fast clocks that cause them to accept any message at any time, causing 512 network-wide floods per round and creating a total of $1024 \cdot |E| \in O(|E|)$ transmissions. Note that this scenario covers the worst-case and includes an adversary with infinite computing power and full control over the network topology.

5.2.4 Experimental Results

We have implemented the presented protocol in the GNUnet P2P framework[1], and evaluated the behavior of the proposed protocol using large-scale emulation [EG11a].

Adaptivity to Churn

To evaluate the network size estimation quality under churn, we show the network size estimate based on an average of the previous 64 rounds. Figure 5.2 shows the evolution of the network size estimate for a random graph topology [ER59, EG11a] with approximately 10 edges per peer. The estimate is calculated with a weighted average over the last 64 readings. It should be noted that the shape of the network topology has no impact on the size estimate. The experiment was started with an initial network size of 4,000 nodes for 640 rounds. Then, we decreased the network size to 1,000 nodes for 640 rounds and finally increased it to 2,000 nodes for another 640 rounds.

Precision vs. Number of Iterations

The number of rounds used to calculate the result has an impact on the precision of the estimate. The trade-off between more measurements and the resulting precision is plotted in Figure 5.3. Precision is measured as $|\tilde{p} - \overline{p}|$. Averaging over four rounds gives results with a standard deviation of one. As the network size is calculated as $2^{\overline{p}-0.332747}$ (Theorem 1), a standard deviation of one means that the network size estimate is in an interval between half and double the actual network size 68% of the time and between a quarter and four times the actual network size 95% of the time. The 64 rounds we used for Figure 5.2 correspond to a standard deviation of under 0.3. This means that 95% of the time the network size estimate is accurate up to a factor of ≈ 1.5.

[1] `https://gnunet.org/svn/gnunet/src/nse/`

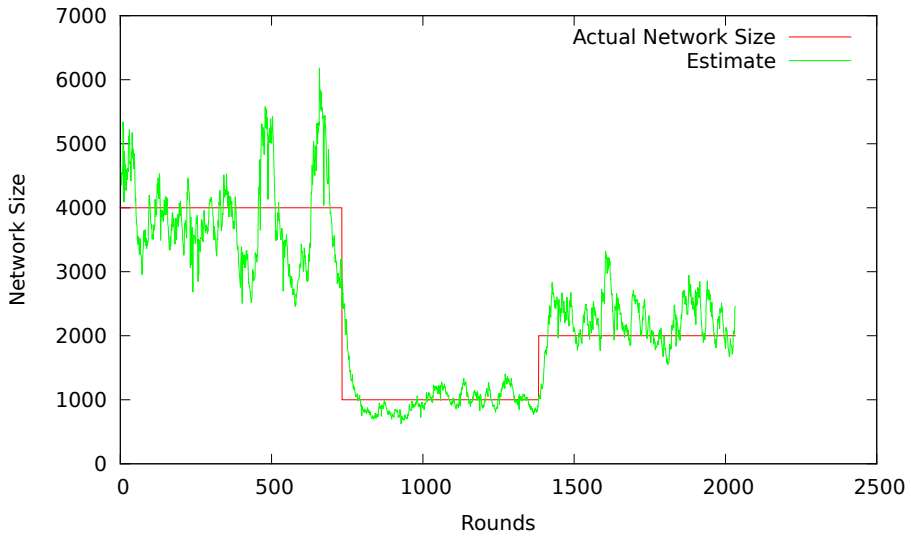


Figure 5.2: Network size estimates as actually observed by a node in relation to the (changing) total network size over time for a random graph topology. All nodes arrive at the same estimate, except for nodes that recently joined the network.

Impact of the Network Topology

Figure 5.4 shows the evolution of the network size estimate for various topologies in relation to the actual network size for a small-world topology [EG11a, Kle00] (Figure 5.4a) and a random graph topology [ER59, EG11a] (Figure 5.4b). Our experiments show that the topology has no significant impact on the result. As we are only using $k = 8$ rounds for the averaging, the results between rounds still differ widely. This is the natural cause of using the counter of a discrete number of matching bits in the exponent: while the standard deviation is typically about 1 bit, this translates to a factor of two for the range of the 68%-confidence interval for the network size estimate.

Impact of Clock Skew

Figure 5.5 compares network size estimates from a network of 1,000 peers with and without clock skew. The clock skew can affect the protocol if the closest peer to the key has a clock so far back that other peers discard that information as too old. This would cause an underestimate of the size of the network. For these tests, we created a small-world network with approximately 5,000 total edges. We used a network size estimate interval of 30 seconds and ran the test for 96 minutes, averaging over 64 measurements. We skewed the clocks of peers by ± 30 seconds, which is a significant clock skew in light of the short network

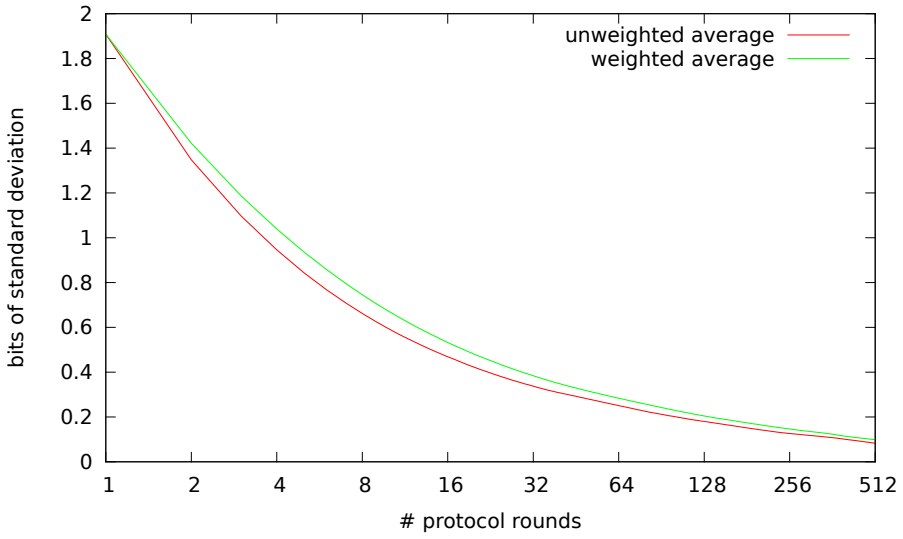


Figure 5.3: Trade-off between the precision of the network size estimate vs the number of rounds used to calculate $\tilde{p}$. Naturally, this plot assumes that the network size does not change during the measurement.

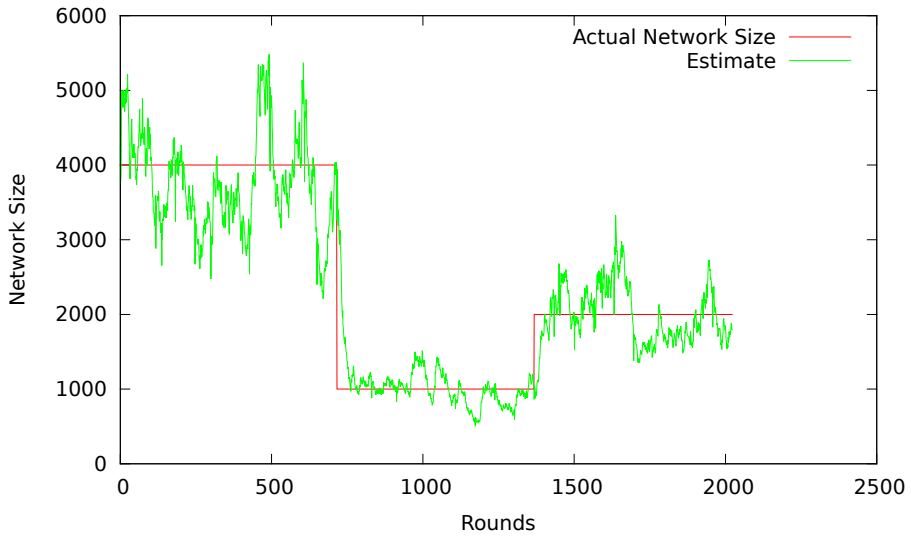
size estimation interval of $f = 30s$. The results from Figure 5.5b generally show similar results in the size estimate, with minimum estimates around 600 peers. The experiment with skew has more data points. This is due to the multiple notifications per round that happen when peers receive delayed messages.

Network-Wide Agreement (under Churn)

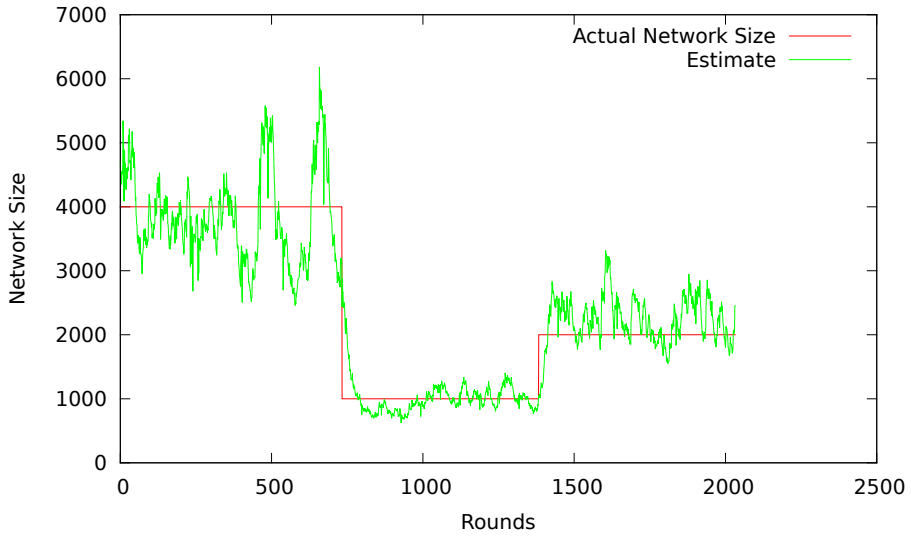
Figure 5.6 plots each of the individual estimates calculated by the various peers in an experiment with clock skew and churn. While the protocol guarantees that all peers within the same connected component will converge to the same network size estimate, this is not true for peers that just joined the network, thus lack some of the previous k proximity values and hence will arrive at a different average. Furthermore, different proximities and clock skew can cause some disagreement between peers, especially early in the experiment where the diameter has not yet been established.

Traffic Cost

Figure 5.7 shows the amount of network traffic generated by the protocol for each round in relation to the number of edges in the network. As before, the network size is cut to a fourth in round 10 and is doubled in round 20. Given the small amount of bandwidth required, it is clearly possible to run this protocol



(a) small-world Topology

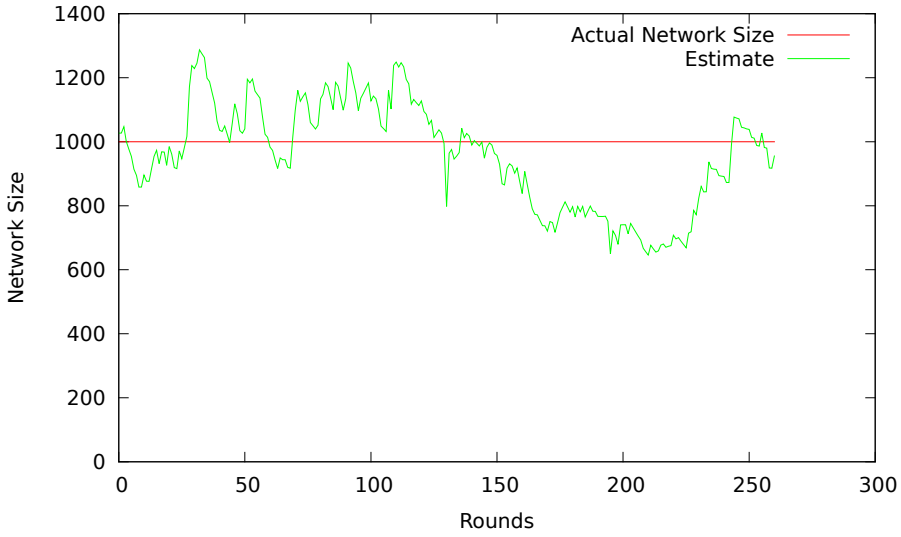


(b) Random Graph Topology

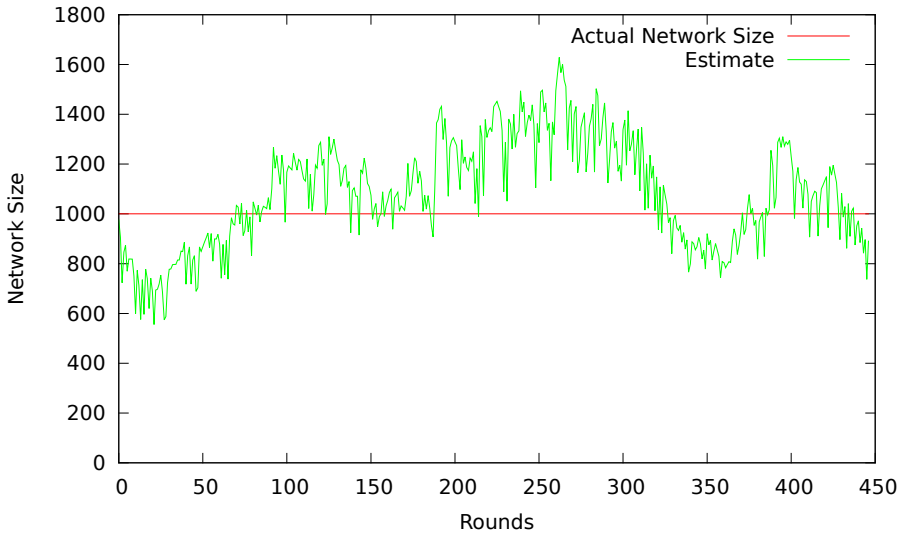
Figure 5.4: Average network size estimate in relation to actual network size over time for different topologies.

with small values of f and large values of k in cases where precise network size estimates are required.

A second round of traffic measurements was performed to demonstrate the



(a) Static small-world - No Skew



(b) Static small-world Topology - Skew

Figure 5.5: Data showing effect of clock skew on average network size estimates for a static network of 1,000 peers. The differences between Figure 5.5a (without skew) and Figure 5.5b (with skew) are, as expected, small.

impact of clock skew on network bandwidth. For this measurement, the local times of the different peers were desynchronized; specifically, the local times at

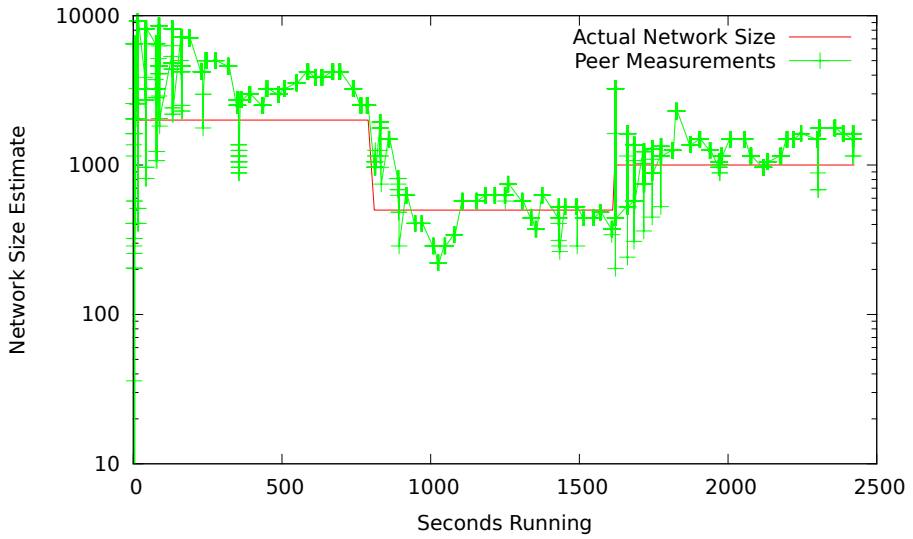


Figure 5.6: This figure plots the results of all peer's network size estimates during the course of a single experiment. While clock skew causes more estimates, disagreement among peers is minimal.

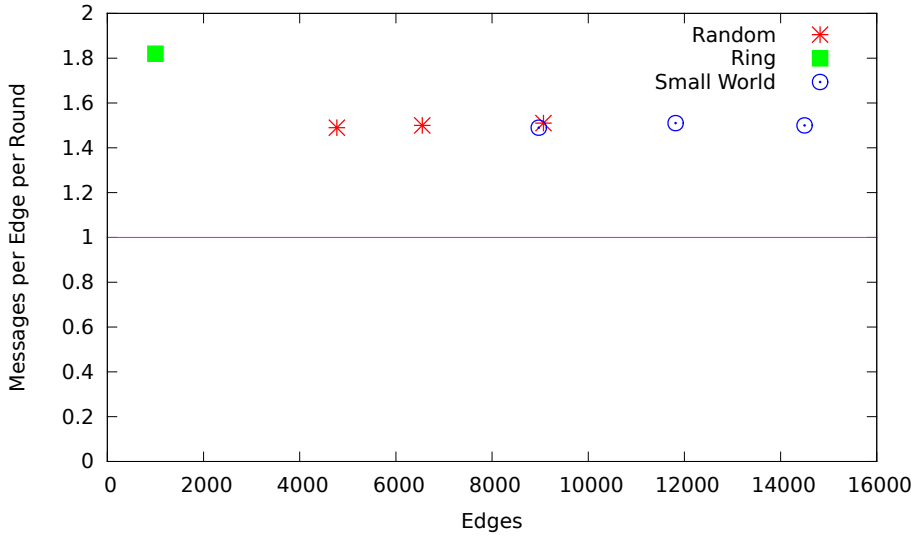


Figure 5.7: Number of messages exchanged in relation to the number of edges in the network.

the different peers were offset from the actual time using a triangular distribution with a maximum deviation of one minute.

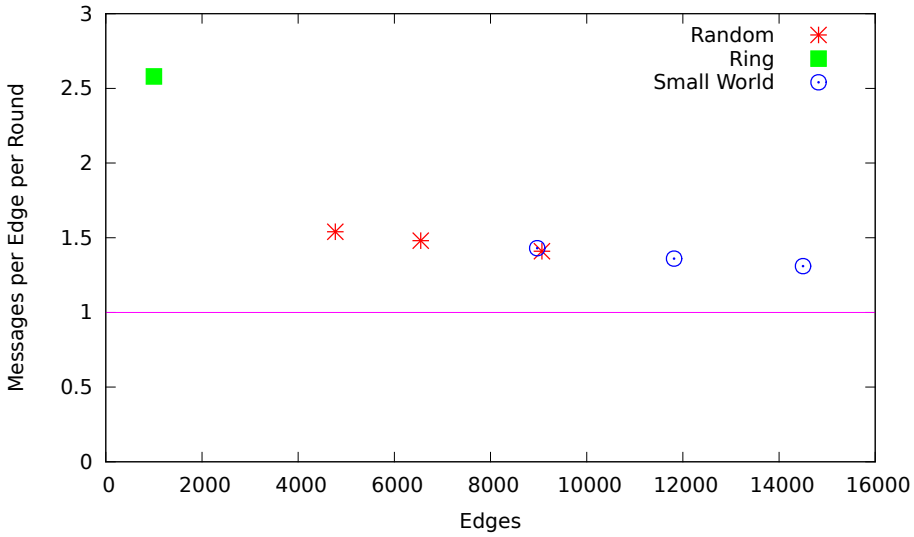


Figure 5.8: Number of messages exchanged in relation to the number of edges in the network if the peers are suffering from clock-skew.

Finally, we evaluated the effect of randomly delaying messages. If we disabled random delays, the overall number of messages exchanged in the network increased by 25% (in a network with 1,000 peers in a small-world topology with 18,000 edges). This demonstrates that the random delays reduce the number of cases where an edge is traversed in both directions at the same time. Globally, the protocol also ensures that the network load is reasonably spread out over time. Figure 5.9 shows the maximum number of messages generated by the entire network in 1ms intervals during the experiment. The shape of the first spikes is different because the network was just started and peers begin their transmissions starting from very diverse initial network size estimates. The second spike is significantly offset from the typical period because the initial estimate (based only on the first round) is far from the typical average for the network. Without the random delays, spikes in traffic for this network could theoretically increase by a factor of 180.

Accuracy of Approximations in Theorem 1

In the proof for Theorem 1 we made an approximation that is valid if "n is sufficiently large". However, what constitutes a sufficiently large n in practice is not obvious. Figure 5.10 shows the results of a simulation that determined $\tilde{p}$ from 50,000 rounds for networks of size $n \in [1, 2^{24}]$. The difference $\tilde{p} - \log_2 n$ quickly converges to the constant calculated in Theorem 1 (0.332747). It should be noted that even with 50,000 rounds the values for $\tilde{p}$ still exhibit some visible fluctuation. Figure 5.10 shows that for a reasonable number of rounds of

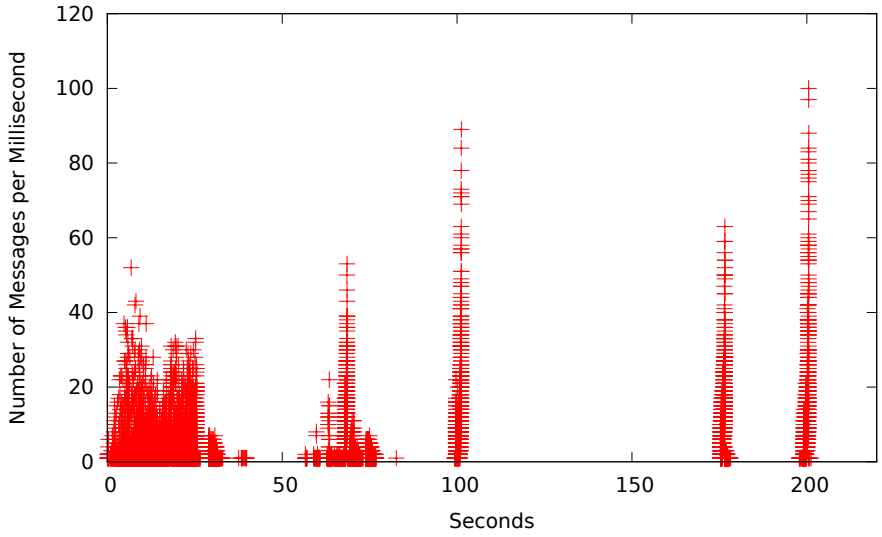


Figure 5.9: Maximum number of messages received per millisecond globally in a network with 1,000 nodes and 18,000 edges. The result seen here is that while spikes exist as the nearest peer in a round sends a message, the total number of messages received is spread out over time due to the built-in message delays. The peaks are somewhat lower initially (while at the same time generating more traffic overall) due to the lack of previous network size and diameter estimates.

measurement ($\leq 50,000$), the "sufficiently large values of n" are values larger than 2^5.

Comparison with other Methods

A naive version of gossiping using randomly chosen edges with 1,000 peers in ring topology takes about 70 million interactions to get all 1,000 peers within a factor of two of the real size (compared to about 4 rounds with 1,000 interactions each for our protocol). For a 2D torus, naive gossiping still took 180,000 interactions. For a random graph, small world or clique, only about 10–20 thousand interactions are required (comparable to the protocol presented in this chapter); thus Gossip efficiency is somewhat dependent on topology and not as efficient in some cases. A much more efficient and precise gossiping protocol is presented in [vdBKM12], offering high precision at a cost comparable to the approach presented in this chapter. However, their protocol is more complex, offers no security and has not been implemented.

Sample & collide showed to be quite dependent from the topology in our simulations, as the algorithm is very dependent on a good sampling method and the topology affects the sampling heavily. A clique of 1,000 nodes provided good estimations, although at a cost of 40,000 messages per round per peer with

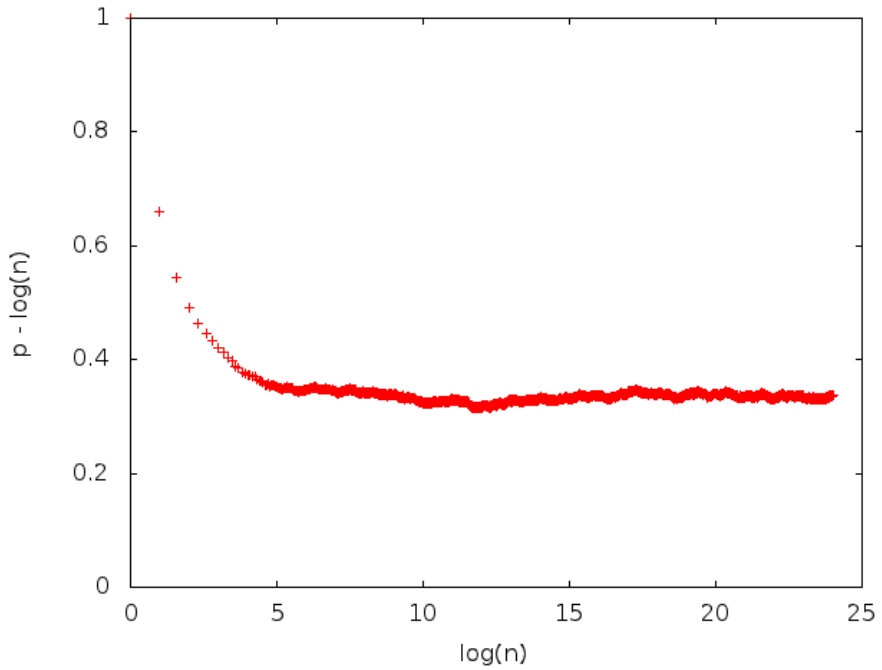


Figure 5.10: Differences observed between $\log_2 n$ and the average observed value for p over 50,000 iterations in relation to the network size. The average difference was $0.33 \approx 1/3$. Since peer-to-peer networks smaller than $2^5 = 32$ peers are not really relevant for network size estimation techniques, we use a uniform correction of $1/3$ to compensate for the observed difference when estimating n from $\overline{p}$.

a parameter $T = 1$ (the higher the degree of the nodes, the longer each sample message has to travel). After 16 rounds the results are within a factor of 2 of the real size. If every peer in the network would be to obtain an estimate, the traffic would amount to 64 million messages, compared to 64,000 messages with our algorithm.

On more restricted topologies the precision was not as good, although the traffic was also lower, due to the lower average degree of the nodes. On a 1000 nodes random topology with 5000 edges the algorithm showed a tendency to underestimate the size, converging to an estimate of under 900 even averaging over 100,000 rounds. The traffic generated was 1,600 messages per peer per round. To obtain an estimate for every peer would require to transmit 2,5 million messages, compared to 320,000 using our approach.

In our experiments, hops sampling gives an accurate (within 10%) result with a single round (for random graph with 1,000 nodes and 10,000 edges). However, already for this small graph hops sampling takes always significantly

more than 10 million messages regardless of parameter choices and, compared to the method proposed in this chapter, also requires more state to be kept by all participating nodes. Given that for a 1,000 node network hops sampling would thus have to be compared in precision and cost to the algorithm presented in this chapter averaged over 1,000 rounds, it is still not competitive in performance. Naturally, compared to hops sampling the network traffic of the protocol described in this chapter is also more evenly distributed. Hops sampling is also much more vulnerable to message loss compared to other methods.

Finally, methods focused on individual estimates are prone to be abused to perform DoS attacks on the network, allowing a malicious participant to multiply its bandwidth by several orders of magnitude.

5.2.5 Discussion

The security of the presented scheme is partially based on the assumption that the adversary cannot calculate W-bit collisions frequently. However, in practice, we must expect that a dedicated adversary may use specialized hardware such as GPUs to achieve significantly higher computational power than normal nodes [LNOM08]. Using a value of W that requires GPUs would exclude many "normal" users (which may not have access to properly configured modern GPUs) from participation. So instead of using bit collisions in cryptographic hash functions — the search for which is easily accelerated with GPUs — one might want to use computational puzzles that are memory-bound [ABMW05] and hence less suitable for GPU-based acceleration.

Another key issue is that the adversary can pre-compute the solution to the next round ahead of time, which makes it trivial for an attacker to parallelize the computation across multiple systems. Such pre-computations could be prevented if the key of future rounds was unpredictable. This would effectively force the adversary to either calculate the proof-of-work in an unrealistically short period of time or to pre-calculate proofs of work for each of the M "fake" peer identities. Consequently, unpredictable keys would seriously limit the effectiveness of such an attack.

However, creating and distributing an unpredictable key for the next round cheaply and securely is difficult. One approach for making the key harder to predict is to tie the key for the next round to the result from the previous round. This would require changing the protocol to propagate a unique "closest" peer identity to all peers. The protocol described herein allows for the possibility that different areas of the network determine different "closest" peers — as described, propagation stops if the number of leading matching bits is identical to that from a previously received message for the current round.

Changing the implementation to flood the entire network with the globally closest peer would be trivial; however, this has other disadvantages. Without adversaries, this change increases the expected overall communication cost since multiple peers with small differences in proximity may start to flood the network at the same time, ultimately causing many edges to be traversed many times. Furthermore, adversaries that are able to predict the target key for the

next round could then efficiently generate many more "closer" node identifiers, increasing the effectiveness of denial-of-service attacks: since proximity changes would no longer be counted in leading bits, causing m rounds of broadcasts would require pre-computing m closer identifiers (at a cost of $O(nm)$). In our current implementation, for m rounds of broadcasts, $O(2^m)$ closer identifiers would have to be pre-computed (at a cost of $O(n2^m)$). Another drawback to such a design change is that an adversary with "unlimited" computational power could cause virtually unlimited rounds of broadcasts, whereas the described method only allows a small constant number of rounds.

While this change would make it much more difficult for the adversary to predict the key, it is still conceivable that the adversary may at some point successfully predict the closest key. At that point, the adversary might be able to pre-compute proofs-of-work in parallel into the future, dominate the closest peer estimate for some amount of time, and during that time have the ability to perform an exponentially more effective denial-of-service attack (due to the required global consensus on the result of the previous round).

For our implementation, we felt that this choice — making it initially harder for the adversary to make the network seem bigger vs. reduced traffic costs in normal operation and an exponentially less effective denial-of-service attack — should be made in favor of the method described in Section 5.2.2.

Naturally, in a setting that does not have to deal with the possibility of malicious participants trying to drive up the network size estimate, the entire proof-of-work, the cryptographic signature, and the public key would not be needed at all. This might, for example, be the case where developers try to get approximate usage metrics but are at the same time too concerned about leaking too much information and are thus unwilling to keep detailed centralized records. The Tor Metrics Portal [2] is an example of such a service where approximate usage counts are actually desired and where it might be unlikely that participants would deliberately provide false information. Calculating these metrics based on Theorem 1 might thus serve the privacy-sensitive nature of the project.

5.3 R^5N: Randomized Recursive Routing for Restricted-Route Networks

Distributed Hash Tables (DHTs) [RFH+01, RD01, SMK+01] are a key data structure for the construction of completely decentralized applications. DHTs are important because they generally provide a robust and efficient means to distribute the storage and retrieval of key-value pairs.

In recent years, DHT designs have become increasingly efficient and robust under churn [LSG+04, OHKY10, RGRK04, SR06] and Sybil attacks [Dou02, LMSW10, SEnB07a, YKGF06]. Other research has addressed implementation concerns, such as optimizing network performance. In practice, modern

[2] `http://metrics.torproject.org/`

DHTs are often not deployed over an entire P2P network and are instead limited in scope to so-called super-nodes. The primary reason for this is that virtually all previous DHT routing algorithms (with the notable exception of Freenet [San06]) are based on the fundamental assumption of universal connectivity between all participating nodes (or rely on unstable NAT traversal).

This assumption means that modern DHTs cannot function properly in networks with limited connectivity (mobile, ad-hoc wireless, sensor, friend-to-friend, etc.). Following [San06], we refer to these networks where peers are not free to directly connect to arbitrary other peers (and therefore route in the DHT) as *restricted-route* networks. We need to distinguish between the network topology created by a peer-to-peer overlay and the underlying network infrastructure, so we use the term restricted-route underlay topology to describe the resultant restrictions imposed on the overlay routing algorithm.

This section introduces a new randomized DHT routing algorithm, R^5N, which enables our DHT to operate effectively over restricted-route networks and also increases security and resilience to various attacks compared to existing algorithms. R^5N only assumes that the topology is connected and, in particular, does not require or use a coordinate system for organizing peers. A primary goal of R^5N is providing an open network where users can join or leave at any time without approval by a certificate authority or other trusted entity.

The R^5N design itself is relatively simple, essentially combining a random walk with recursive Kademlia-style [MM02] routing. Our design also includes topology augmentation using a combination of distance-vector and onion-routing, a novel replication strategy and an API to verify content integrity. Using distributed emulation, we demonstrate that this new algorithm has performance comparable to Kademlia if the underlay is unrestricted, and outperforms Kademlia and random walks for various restricted-route topologies. We also show that our algorithm has advantages in terms of availability and fault-tolerance, especially in the presence of malicious participants. Compared to Kademlia, we generally see a larger number of replicas and higher success rates for data retrieval.

5.3.1 Related Work

A DHT imposes a structure upon the network underlay by connecting peers to a certain subset of all nodes in the network. The size and method of construction of the routing table is one of the key design choices that distinguish DHTs. For example, Kademlia [MM02] has routing tables of size $O(\log n)$ and can route requests to the proper destination with $O(\log n)$ steps.

Another key design choice for a DHT is the routing or lookup behavior, which is categorized either as iterative or recursive [HC07]. In iterative routing, the initiator directly connects to each hop and retrieves information about the next hop until the initiator has a direct connection to the final destination. As a result, the initiator of a request has full control over which node(s) the request is forwarded to at each step — and can possibly tackle problems (such as node failures or malicious participants) during the propagation (for example,

by choosing alternative paths).

With recursive routing, the request is forwarded through the network from the first hop onwards according to the routing algorithm and the initiator is only involved again as the final destination of the response, if there is any. Recursive routing is generally faster than iterative routing since fewer connections need to be established and thus the number of round-trip-times is significantly smaller. Another key benefit of recursive routing is that the initiator does not have to be able to connect to each peer that participates in request routing. However, recursive routing is also less fault-tolerant due to the initiator's lack of control.

Kademlia

We use a modified version of Kademlia [MM02] as the basis of our routing algorithm. The Kademlia algorithm has been shown to work well in networks with common rates of churn [OHKY10] and has, in practice, proven capable of handling millions of peers [SENB07b]. Kademlia uses XOR to determine the distance between elements in the key space.

Kademlia's routing table is structured as an array of k-buckets. Kademlia uses as many k-buckets as there are bits in the address space. Each k-bucket can hold up to k peers. The i-th k-bucket stores up to k peers whose identifiers are between distance 2^i and 2^{i+1} from the local peer. Routing in Kademlia is iterative; at each step the initiating node picks r closest peers for the next step. Those r peers are queried and return a set of peers closer to the key, and routing continues in this fashion until no closer peers are found. Finally, Kademlia stores data at the r closest peers to the key. Kademlia achieves $O(\log n)$ routing performance: in each step the distance to the destination is at least halved (Figure 5.11).

One failing of Kademlia is that it has been shown vulnerable to numerous attacks, such as poisoning [LMSW10] and Sybil [SEnB07a] attacks. For example, an adversary may want to deny participants access to a particular key. This can be achieved by creating r peers with identifiers closer than the closest current peer to the key; afterwards, all requests will effectively end at an adversary-controlled peer. Access to the data is then under the control of the adversary.

5.3.2 Restricted-Route Topologies

We use the term "restricted-route topology" to refer to a connected underlay topology which prohibits (restricts) direct connections between some of the nodes. Common DHT routing algorithms show diminished performance or even arrant failure when operating over a restricted-route underlay. A common solution on the Internet is to restrict participation in the DHT to peers that are not encumbered by NATs or firewalls. However, this solution limits load-distribution for P2P applications on the Internet and does not work at all for physical networks or friend-to-friend networks. For these types of networks,

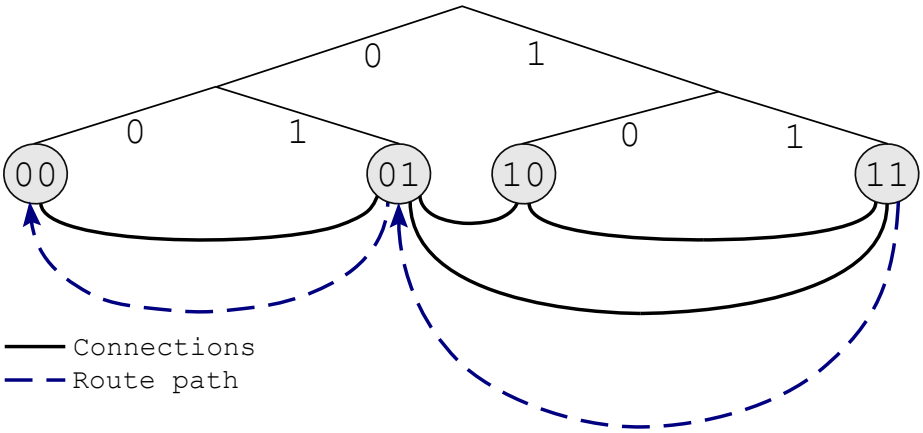


Figure 5.11: Illustration of Kademlia routing for key "00". Routing in a four-node Kademlia network ($r = 1$) with a set of connections that satisfy Kademlia's requirements. In this topology, requests for keys starting with "00" terminate at node "00".

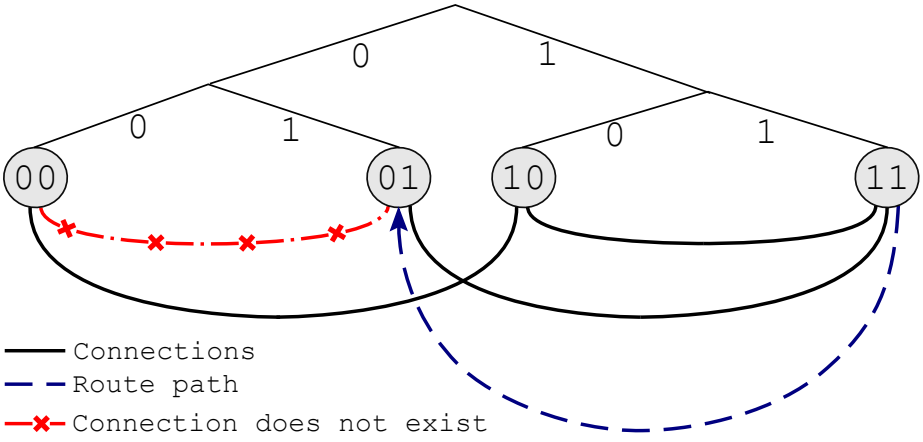


Figure 5.12: Illustration of Kademlia routing for key "00", except this time a single required link is unavailable. Here, requests started at peers "10" and "00" are routed to the correct node while those started at "01" or "11" are incorrectly routed to "01".

some other method of routing must be employed to cope with restrictions on direct communication.

Freenet

Freenet [San06] is the only efficient DHT design we are aware of which works well in restricted-route networks without coordinates. Freenet uses location

swapping between peers in order to structure the overlay topology and is able to route in $O(\log n)$ steps. The main problem with Freenet's DHT is the inherent vulnerability of the critical location swapping operation [EGG07]. This operation allows an adversary with only a few peers anywhere in the network to cause massive peer identifier clustering, leading to possible data loss and destroying the load balancing properties of the DHT. Furthermore, natural churn can cause similar problems even in the absence of malicious peers.

Randomized Designs

One interesting sub-category of restricted-route networks are sparsely connected networks, where $|E| \sim |V| \cdot \log |V|$. These can often be modeled quite well as random graphs, and it has been shown that flooding [LCC+02] or random walks [ABK+92] are the optimal method for searching such random graphs, though these unstructured techniques are costly ($O(\log n\sqrt{n})$) compared to modern structured routing algorithms [FRA+05].

Various so-called "randomized" DHT designs have recently been proposed to deal with problems associated with the deterministic nature of traditional DHTs [BBK+10, DLlKA05, MBR03, ZCSY08]. In [BBK+10, MBR03] and [ZCSY08], the overlay structure is partially randomized, using random peer sampling (RPS) to select neighbors. The randomization is used to either increase resilience to attacks or to alleviate the impact of high churn rates. In the design of [DLlKA05], routing behavior is randomized to defend against Sybil attacks, but it is costly in terms of routing table maintenance. To the best of our knowledge, none of these designs cope well with restricted-route underlay topologies, necessitating a truly randomized design such as R^5N.

5.3.3 Design of R^5N

The basic idea of R^5N is to take advantage of the limited connectivity of restricted-route networks by using the large number of peers that are closer to a key than any of their neighbors for replication. A PUT operation is used to store data at a random subset of these peers, and subsequent GETs then attempt to reach one of the replicas. PUTs are repeated at a certain frequency to refresh data. Since R^5N performs non-deterministic routing, repeated PUTs are likely to result in data being stored at different peers. Furthermore, since our design specifies that this refresh period is significantly shorter than the timeout of content at the replica nodes, this increases the chance of success for subsequent GET operations.

Naturally, a GET may still fail to find its target value. In this case, R^5N expects peers performing GET to retry a few times. Since GETs are also non-deterministic, repeating the operation has a high chance of reaching different peers and hence improves the chance of finding the data. While the design guarantees that the chance of failing to find existing data declines over time, the specifics depend on a replication parameter r, the network topology and the number and behavior of adversaries in the network.

Since both GET and PUT operations take different paths each time, an adversary has little chance to successfully place his nodes in the network to block particular key-value pairs. Depending on how the restricted-route underlay is constructed, an isolation attack on nodes may still succeed.

The remainder of this section will detail the various components required for the R^5N routing algorithm. Specifically, we will discuss routing table construction, the use of bounded onion-routing to augment the underlay, request processing, content replication and application-level requirements (specifically content validation).

The Routing Table

Routing tables in R^5N are constructed and maintained in the same manner as in Kademlia (Section 5.3.1), with the main difference being that R^5N expects that (especially higher numbered) buckets will be empty even though peers with appropriate identifiers exist in the network — direct connections were simply not possible or peers were not discovered because lookups failed (where they would have succeeded in Kademlia). As in Kademlia [MM02], this results in $O(\log n)$ connections to neighbors. It should be noted that a small difference in routing table maintenance arises indirectly because routing of FIND PEER (in both Kademlia and R^5N) is based on routing of GET and PUT requests, and R^5N's routing of GETs and PUTs is slightly different (and hence FIND PEER requests are also routed differently).

Routing

Routing in R^5N is recursive and is performed in two distinct phases. In phase one, a request for a key is routed for some number of hops using random neighbors from the routing table. In phase two, routing is deterministic using the peers from the routing table that are closest to the given target. Each request includes the number of hops h that the request has traversed so far, and each peer is supposed to increment the counter by one at each hop. Once the hop counter exceeds a threshold of $T \approx \log n$ where n is the size of the network, the request enters the second phase. The intuition behind this is that we first make the starting point in the network independent from the location of the initiator and then efficiently find a nearest peer. Because the underlay topology is supposed to be a restricted-route topology, there are many peers that are nearest to the key as far as their immediate neighborhood is concerned:

Lemma 1 (Number of Nearest Peers in a Random Graph)**.** *For a random network with n peers and c random connections per peer, the expected number of nearest peers in the network to any random key is approximately $\frac{n}{c+1}$.*

The optimal number of random hops taken is equal to the mixing time of the graph [LPW06]. The Markov mixing time for various graphs is well known. In a full clique, the optimal number of random steps to take is 1; in a completely random graph, it is closer to $O(\log^2 n)$ steps [AE07]. For small-world and social

networks, the mixing time has been shown to be around $O(\log n)$ [DaR09]. Since we expect R^5N to be used primarily in network topologies that more or less conform to small-world topologies, $T \approx \log n$ random hops should be enough to arrive at a sufficiently random point in the graph.

Each request also contains a unique identifier and a 64-bit bloom filter which are used to improve efficiency by preventing looping and limiting repeated forwarding of the same request to the same peer. The bloom filter is updated with the list of peers selected for forwarding the request to at each hop — those peers that match the bloom filter are excluded from the selection process.

Processing Requests and Replies

Each peer that receives a routing request performs the same basic sequence of operations. First, the peer determines whether it is closer to the key of the request than any of the peers in its routing table. If the current peer is a nearest peer, PUT requests are not forwarded; instead the data is stored locally. GET requests where the only possible result is found locally are also not forwarded. Otherwise, the request is forwarded to neighboring peers; these are selected from the routing table using random peer selection or the XOR distance metric depending on the current hop counter. The number of forward replicas is calculated according to the replication level, network size estimate and number of hops traversed so far as described in Section 5.3.3.

For handling replies, each peer tracks a bounded number of active requests, including the respective identity of the preceding peer. Responses are forwarded along the request paths until they reach the original peer or are discarded by a peer that lacks path information (due to memory limitations, for example). It should be noted that most other DHTs do not require this additional state since, in traditional DHTs, the normal routing mechanism can also be used to route replies. For R^5N, this is not feasible due to path randomization. Were R^5N to use randomization for replies, the success rate for replies to reach the intended initiator would be rather low. In contrast, randomization for the lookup is acceptable since many peers are expected to store the data due to replication.

Replication

In R^5N, replication is used not only to protect against node failure, but also to improve the chances of a lookup operation finding the desired datum in the absence of failures. For R^5N, the highest GET success rate would be achieved if there are $\frac{n}{c+1}$ replicas in the network (Lemma 1). We use r to describe the desired replication level and for R^5N the target value is $r \sim \sqrt{\frac{n}{c+1}}$; this choice represents a trade-off between the cost for PUTs and the performance for GETs.

If the initiator were to transmit r PUT requests to obtain r replicas, there would be a good chance of collision in the resulting paths and this might be a strong burden on the direct neighbors of the initiator, especially since in the underlay the initiator may not even have r neighbors. Instead, R^5N attempts to have (on average) $1 + \frac{(r-1)h}{T}$ PUT requests active in the network at hop h.

Lemma 2. *Let h be the number of hops in the network that the query has already traversed. If the network is large enough that r random paths of length T are unlikely to merge and if $h < T$, then the average number of peers to which a peer forwards a request to should be*

$$\Upsilon_{r,h} := 1 + \frac{(r-1)}{T + (r-1)h} \tag{5.7}$$

in order to achieve the desired replication level r at T hops.

R^5N uses a biased random selection, forwarding to either $\lfloor \Upsilon_{r,h} \rfloor$ or $\lceil \Upsilon_{r,h} \rceil$ peers to reach on average $\Upsilon_{r,h}$ peers for the next hop. We continue to forward to $\Upsilon_{r,h}$ peers for $h \leq 2 \cdot T$ (instead of just until $h < T$) to compensate for path collisions, inaccuracies in the network size prediction and not forwarding PUT requests from nearest neighbors.

Content Validation

A key concern for any DHT is the integrity of the content stored in the system. R^5N provides an application with hooks for integrity checks to detect malformed key-value pairs. Such pairs are then not forwarded or stored by well-behaved peers, reducing storage and bandwidth requirements in the presence of faulty or malicious participants and making DHT pollution more difficult.

Another possible issue is allowing multiple values to be stored under the same key. Requests in R^5N include a bloom filter which matches replies already known to the requester. While bloom filters offer a compact way to filter replies, they can also produce false positives. R^5N mitigates this problem by having the requester provide an additional 32-bit mutation value which modifies the hash function used for testing the bloom filter. By re-issuing the request with a different mutation value, these false positives can be eliminated.

5.3.4 Experimental Results

Unless stated explicitly otherwise, the presented experiments were done using a fixed replication level of $r = 10$ and a fixed network size estimate parameter $T = 4$ ensuring that only the shape of the topology and the node degree are parameters for the evaluation.

Experimental Setup

We have analyzed the expected performance of R^5N using mathematical analysis, simulation and emulation. Due to space constraints, the experimental results presented in this section are only based on our experiments using emulation. For our emulation experiments, we implemented our DHT routing algorithm on top of an existing P2P framework. The results presented in this chapter were obtained by emulating 2025 peers on a single desktop, which is close to the limits of our hardware and since $45^2 = 2025$ this number allows for the construction

of a clean 2D-torus topology as a starting point for our small-world topology construction. Our emulation does not model network latencies; however, this is not a significant problem since R^5N currently ignores link latencies in its peer selection strategy.

Underlay Topologies

For the evaluation of our routing algorithm we use a number of well known topologies:

Unrestricted All connections allowed up to some upper bound for $|E|$.

small-world Extends a 2D-torus by adding or rewiring random links. [Kle00]

InterNAT Allows a set of $p \cdot n$ unrestricted peers to connect to any peer and $(1-p) \cdot n$ peers to only connect to unrestricted peers (for $p \in [0,1]$). This models $(1-p) \cdot n$ peers behind NAT (without traversal ability).

Erdos-Renyi Peers are allowed to connect with a fixed random probability. [ER59]

R-Kademlia

We use a variant of Kademlia, which we call R-Kademlia, as a point of comparison with our own algorithm. The iterative routing in the original Kademlia design performs so badly in a restricted-route topology that it is not useful for comparison. R-Kademlia is a recursive implementation of Kademlia that is otherwise as faithful to the original design as possible.

The first — and biggest — problem with a recursive implementation of Kademlia is that r concurrent requests are meant to be kept in flight until no closer peers are found. R-Kademlia initiates r requests at the first peer. These requests terminate once a nearest peer is reached; however, the initiator has no way to guarantee this. Peers in R-Kademlia are responsible for attempting to forward requests only to peers that have not encountered the request already using a bloom filter (as explained in 5.3.3). Peers also maintain a limited store of recent requests; thus, if the same request reaches a peer twice, the bloom filters are merged. Using these techniques, we mimic the iterative routing of Kademlia, with the exception that the initiator cannot control the next-hop decisions.

Worst-Case Network Performance

For networks with few connections, the success rate of R^5N is significantly higher than it is for R-Kademlia. The worst case for R^5N when compared to R-Kademlia is hence an unrestricted underlay topology (clique). In this case, both R-Kademlia and R^5N will always find the data at the nearest peer on the first attempt, but R^5N is expected to take longer. Table 5.1 shows the average number of hops taken for the two designs in this worst-case scenario for R^5N.

Table 5.1: Average hops for R-Kademlia and R^5N in clique underlay topologies of different sizes. As expected, R^5N takes about twice as many hops as R-Kademlia.

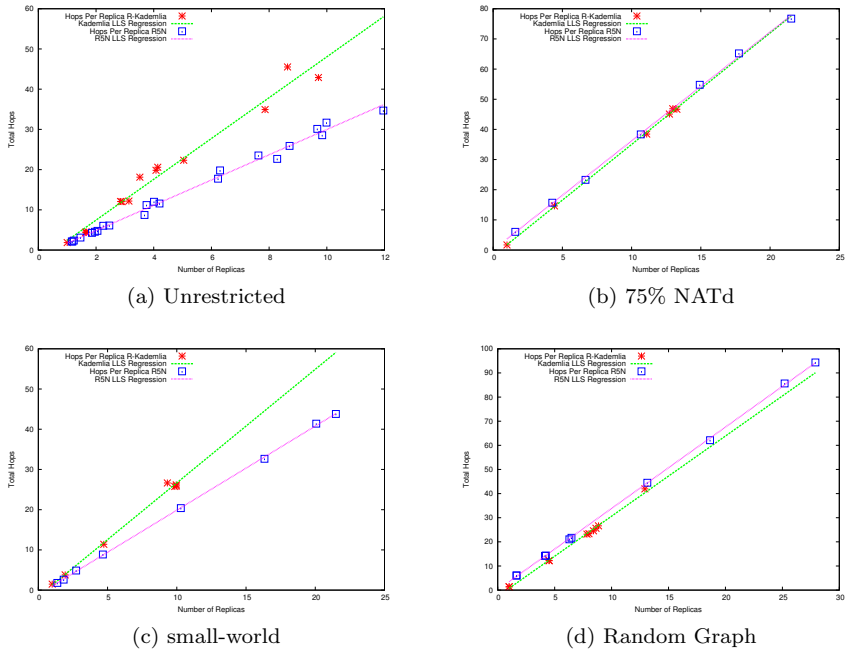
Size of network	Average hops per PUT		Average hops per GET	
	R-Kademlia	R^5N	R-Kademlia	R^5N
100	2.70 ± 0.06	3.96 ± 0.06	2.54 ± 0.03	4.63 ± 0.17
250	3.06 ± 0.10	4.26 ± 0.10	3.10 ± 0.06	5.96 ± 0.27
500	3.08 ± 0.46	4.38 ± 0.45	3.38 ± 0.06	6.17 ± 1.14
750	3.19 ± 0.74	4.37 ± 0.83	3.50 ± 0.04	6.29 ± 1.04
1000	3.63 ± 0.07	4.47 ± 0.93	3.64 ± 0.04	7.29 ± 0.95

Replication Performance

As described in Section 5.3.4, R-Kademlia attempts to achieve a certain replication level r by starting r requests in parallel from the initiating peer. In contrast, R^5N probabilistically chooses multiple peers to forward the request to at each hop. Neither approach is able to precisely hit the specified replication target; however, R^5N produces the same number of replicas with significantly fewer messages when compared to R-Kademlia (Figure 5.13) for the unrestricted and small-world topologies, and about the same number of total messages for the other two topologies. This is because sending out many parallel requests from the same initial peer increases the chance that paths will at times converge, while requests that branch at later hops are likely to be further apart in the network and carry more information about which peers have already been routed to and therefore overlap with lower probability. We limit results to 30 total replicas or less; at higher replication levels R^5N outperforms R-Kademlia in all topologies.[3]

Figure 5.14 compares the total number of replicas after several rounds of PUT operations for the same key-value pair (without churn or replica expiration). The figure shows the number of replicas that is achieved by either R-Kademlia and R^5N for the case where either the *same* peer performs the PUT operation or where the source of the PUT operation is chosen at *random* in each round. If the same peer performs the PUT operation using R-Kademlia, the PUT paths always converge at the same nearest peers and, hence, the number of replicas remains constant. In contrast, with R^5N, random peer selection achieves significantly higher levels of replication over time. If PUTs in R-Kademlia are started at a random peer, the resulting replication levels are only slightly higher, suggesting that the random phase achieves its mixing goal.

[3]Due to R-Kademlia's inability to create more replicas than connections.



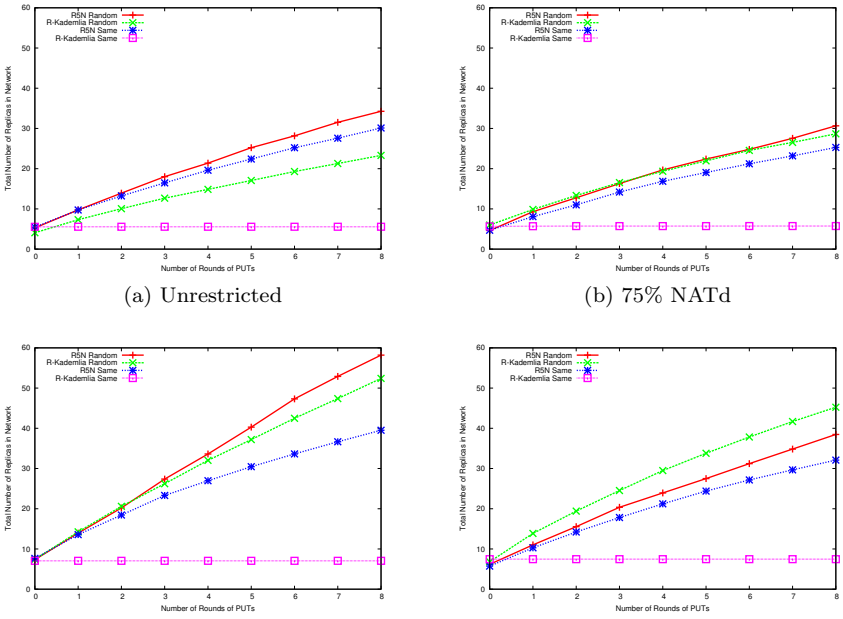
(a) Unrestricted

(b) 75% NATd

(c) small-world

(d) Random Graph

Figure 5.13: Average hops required per replica; varying replication level r.



(a) Unrestricted

(b) 75% NATd

(c) small-world

(d) Random Graph

Figure 5.14: Replication over time; same starting peer vs. randomized starting peers.

Robustness Against Active Adversaries

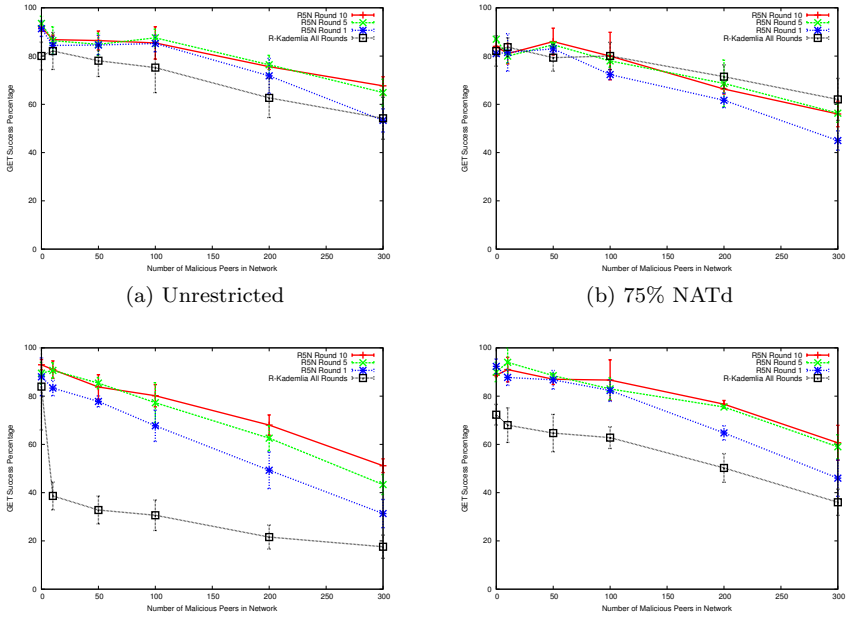
An additional goal for our routing algorithm is to perform well in the presence of malicious participants. Handling malicious participants well subsumes handling peer failure due to bugs, churn or misconfiguration. The design of R^5N already explicitly addresses malicious peers that attempt to perform denial of service (DoS) attacks on the network by bounding the resource multiplier effect of all operations. Peers cannot send requests that consume significantly more resources than "normal" requests, so an adversary can only multiply its own bandwidth by less than the average number of hops for requests multiplied by the replication level r. Similarly, poisoning attacks can sometimes be mitigated using content validation hooks (Section 5.3.3). If necessary, we expect values to be encrypted to protect against adversaries observing values; this can be compatible with content validation by using techniques such as Freenet's CHKs.

An active adversary could also join the network with peers that simply passively drop all requests that are received. This kind of attack is already quite detrimental to overall operation for deterministic algorithms: any request that traverses any of the malicious peers fails. R-Kademlia's redundancy (r-replication) is a typical mitigation strategy. Figure 5.15 shows the impact of a dropping adversary on the performance of R^5N and R-Kademlia in terms of success rates for GET operations (initiated at peers selected uniformly at random in each round) for various topologies, all generated to have 2025 nodes and 30k edges. The GET operations were performed after a number of rounds of PUT operations which are initiated at the same peer in each round.

Later GET rounds in R^5N have higher success rates because additional PUT rounds increase availability for R^5N as more replicas are created. The benefit of R^5N over R-Kademlia is clearly seen in the randomized underlay topologies; while the unrestricted topologies are less affected by this type of attack for both R-Kademlia and R^5N. Importantly, the performance of R^5N never falls *below* that of R-Kademlia.

We now consider an attacker trying to prevent access to a particular key using an Eclipse attack. The attacker again simply drops all GET and PUT requests; however, this time the μ malicious nodes are not placed into the network at random but at the μ peers that are closest to the key. This represents an attacker performing a Sybil attack with free choice of identifier and node placement in a restricted-route topology — the strongest type of Sybil attacker we can imagine. This attack has a serious impact on Kademlia-based DHTs [SEnB07a].

While additional traditional protections against Sybil attacks could be deployed to make this attack impossible, Figure 5.16 shows that such measures may often be unnecessary for R^5N. Again, as rounds of PUT requests increase the number of replicas in the network, R^5N's success rate increases. Again, R^5N outperforms R-Kademlia and is especially strong in the case of a Sybil attack on the small-world underlay topology, where even the first round of GET requests succeeds with a much higher rate than R-Kademlia.



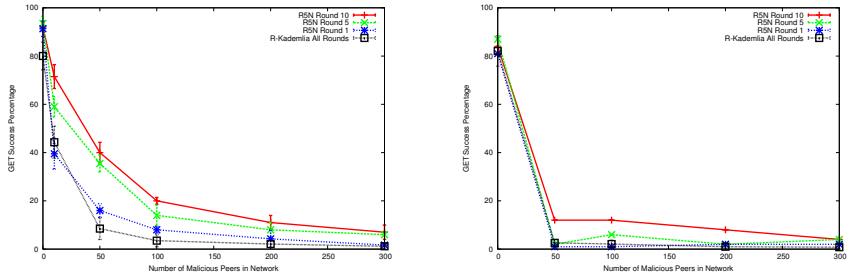
(a) Unrestricted

(b) 75% NATd

(c) small-world

(d) Random Graph

Figure 5.15: Number of malicious peers present at random locations in a network with 2025 peers vs. percentage of successful GET requests.



(a) Unrestricted

(b) 75% NATd

(c) small-world

(d) Random Graph

Figure 5.16: Number of malicious peers present at Sybil locations in a network with 2025 peers vs. percentage of successful GET requests.

5.3.5 Performance Analysis

To achieve high success rates, R^5N needs to create a sufficient number of replicas. A network with n nodes of degree c is expected to have $\frac{n}{c}$ nearest peers. Assuming that T is chosen large enough to achieve perfect mixing, $\sqrt{\frac{n}{c}}$ replicas would need to be created in order for a GET request to succeed with about 50% probability according to the birthday paradox. Once individual requests succeed with this probability, a small constant number of repetitions can be used to get high overall success rates. As we have shown experimentally, the relationship between the number of replicas in the network and the number of hops required for the respective PUT operations is almost linear (Figure 5.14). Since individual PUT and GET requests have complexity $O(\log n)$, routing in small-world networks using R^5N scales with $O(\sqrt{n} \cdot \log n)$. Note that this does not hold in sparse graphs with large diameter or graphs that are not expander graphs (such as a circle) because the routing tables could not be sufficiently populated.

5.4 CADET: Confidential Ad-hoc Decentralized End-to-End Transport

We finally now come to the design of CADET, our design to provide end-to-end connectivity on top of R^5N. CADET is expected to operate above a restricted communication layer with properties similar to those of Ethernet or WLAN, and in conjunction with the R^5N DHT that can operate in the resulting restricted-route network.

Given these two lower-layer components, the roles of both IP and TCP/UDP are then fulfilled by CADET. In particular, CADET takes care of the connectivity needs of peers that cannot establish direct connections. CADET uses the DHT to discover routes, performs authentication, encryption and traffic control between any two peers in the network — as long as they are in a connected graph.

CADET itself is organized in three layers. The bottom layer provides connectivity, like IP. The middle layer provides end-to-end authenticated encryption, similar to TLS. The top layer provides multiplexing, traffic control and other optional features, such as reliability, in a way similar to TCP, UDP or SCTP.

5.4.1 Connectivity

The bottom layer of CADET provides connectivity between two endpoints. By endpoints we understand peers that are running the applications that are going to communicate with each other. All the other peers may function as *relays*: peers that participate in a communication but are neither the origin nor the destination of the traffic. Relays simply forward messages from one neighbor to another.

The connectivity layer has two important concepts: *paths* and *connections*. A path is simply a sequence of peers, where each pair of peers from the sequence is directly connected (on the lower, Ethernet-like layer below CADET). A path is thus just information about the topology of the network stored at an individual peer; it does not carry any communication with other peers. It is roughly the equivalent of a phone number in the telephone system. A connection on the other hand is similar to a call in progress, similar to a phone circuit. It is a reserved path. When an origin peer wants to communicate with another peer in the network to which it cannot connect to directly, the origin peer discovers a path between the two and notifies all the peers in the path about its intention to communicate with the destination, thereby creating a connection.

Each connection has a unique 256 bit random ID. Each peer on the path needs to store the connection ID, together with information about the next and previous peer it should relay traffic to. Two peers can have more than one connection between each other; in fact, this is common to improve reliability and performance. Whenever multiple connections are available, they are used simultaneously by sending messages on the connection that has the least load at the time.

Path discovery

In order to discover paths to other peers, CADET uses two sources of information:

- Passive monitoring of connections being established by other peers. Here, CADET simply analyzes the traffic it relays for other peers and incorporates the topology information contained in connection requests into its own local view of the network.

- Explicit DHT requests, where the DHT is instructed to record the route taken by GET and PUT requests. More specifically, each peer periodically makes a *PUT* request to the DHT with its ID and information that may help to establish a direct connection to it (such as lower layer addressing information). When CADET tries to connect to a peer to which it does not know any path, CADET issues a *GET* request using the ID of the destination peer. By joining the *PUT* route and the *GET* route, CADET obtains a path towards the destination peer. CADET naturally also tries to make use of the information to try to establish a direct connection.

Connection establishment

When a peer decides to establish a new connection a destination peer, it first checks if there are any known paths towards the destination. If there are no paths, the service initiates a DHT query to find them.

When there is at least one known path, the origin peer creates a connection with a random 256 bit ID and sends a $CONNECTION\_CREATE$ message containing a list of all the peers in the path to the first peer on the path. Each

peer forwards the message to the next peer, while storing the connection details to be able to forward subsequent messages identified just by the connection ID. At the same time, each peer uses the path to passively learn potential paths to reach the origin, the destination, as well as all the intermediate peers in the connection.

The destination receives the incoming connection and responds to the origin peer with a $CONNECTION\_ACK$ message in order to confirm the correctness of the path. Each peer on the path uses the stored information from the creation message to send the message in the opposite direction. At the same time, all the potential paths to peers participating in the connection are considered confirmed.

The first peer sends another $CONNECTION\_ACK$ to confirm the receipt of the first $CONNECTION\_ACK$ and finish the connection creation. When a peer is expecting a $CONNECTION\_ACK$, it starts a timer to retransmit the message it sent (either $CONNECTION\_CREATE$ or $CONNECTION\_ACK$) in case the message or the acknowledgement was lost.

Keepalive

In order to avoid saturating the network with connections, idle connections are destroyed after a timeout. There are two idle counters, one for each direction, as any endpoint being down is a reason to tear down the connection. To avoid this timeout, peers periodically send keepalive traffic on idle connections.

Congestion Control

To avoid saturating intermediate peers with traffic, peers use an ACK mechanism for congestion control on each hop of a connection. Each peer has a dedicated buffer per connection and sends an ACK to the previous peer in the connection when the buffer has room for new messages. Since the lower layer does not guarantee reliable delivery, CADET uses a polling system to compensate for lost data or ACK messages.

5.4.2 Security

The second CADET layer, provides authentication and encryption to the communication, encapsulating all traffic in *tunnels*. A tunnel is a secured communication session between two peers. The tunnel uses connections to send data to remote peers, with a target of three connections at once. This redundancy serves for reliability and performance purposes. The first messages exchanged by two peers are a key exchange in which they authenticate each other and use ephemeral keys establish a session key. This session key is cycled periodically, and once it is changes it becomes impossible to decrypt captured traffic, even if the endpoints are seized and forensically analyzed. Additionally, each message uses a random initialization vector, to prevent the same transmitted

plaintext from rendering the same ciphertext and thus leaking information to an eavesdropper.

Tunnel establishment

When two peers are first connected, and periodically after that, they perform a handshake to establish the tunnel's encryption keys. Every peer in the network maintains an Ed25519 [BDL+12] ephemeral key signed with its permanent identity key. The tunnel key material is derived from the ephemeral keys of both endpoints. The ephemeral key is changed periodically every 12 hours, thus every tunnel's keys change on average every 6 hours, once for every peer's ephemeral key. Once the ephemeral key changes, all captured traffic that used encryption keys derived from the old ephemeral keys is no longer decryptable, therefore providing perfect forward secrecy.

To establish a tunnel, each peer sends the other endpoint an initial ephemeral key message containing its identity (the peer's public key), the ephemeral key, the ephemeral key's validity period and a signature to validate the data. Both peers send this in parallel, without waiting for each other.

On receipt of an ephemeral key message, each peer checks the signature and validity period. If everything is correct, CADET derives the key material from both ephemeral keys using Elliptic Curve Diffie Hellman [DH76]. This key material is fed to a key derivation function [Kal00], together with both peers identities and a salt value to obtain the symmetric keys used to encrypt payload traffic in the tunnel. The symmetric keys are 512 bits, composed by a 256 bit AES and a 256 bit Twofish key. CADET uses different sending and receiving symmetric keys, obtained by feeding the identities in different order in the Key Derivation Function.

As soon as the symmetric keys are obtained, each peer sends a challenge PING message to the other peer, encrypted with the new keys containing the remote peer's identity and a nonce.

To verify the correctness of the key exchange each peer checks its own identity in the PING message and if it is correct, sends the PING's nonce in a response PONG message.

Upon receipt of the PONG message, CADET checks the nonce and if it's correct, considers the tunnel as established.

5.4.3 Multiplexing

CADET uses fast Ed25519 public key cryptography, but it still important to minimize the use of this operation. To avoid triggering a handshake every time two applications on a pair of peers communicate, CADET multiplexes all communication channels for a given pair of endpoints inside one tunnel. A *channel* is a communication stream between two applications running on peers participating in the GNUnet network. They can be on the same or different peers. In case they are on different peers, the messages sent to each other are transmitted in a CADET tunnel to the other peers, which demultiplexes them

and delivers them to the corresponding application. In order to share a tunnel among multiple channels, each channel has a unique ID. CADET offers optional, per-channel options, such as out of order delivery and reliability.

Channel establishment

Channel establishment happens in a similar way to the connection establishment. First, the application requests a new channel, with the destination specified by a peer ID and a port number. It identifies the channel with a sequential 32 bit local ID with the most significant bit set to 1. The local ID is unique per client and many clients on a computer may use the same local ID.

CADET checks if there already is a tunnel towards the destination peer. If there is no tunnel, it creates a once as described previously.

Given an existing tunnel, the origin peer creates a channel with a sequential 32 bit ID and sends a $CHANNEL\_CREATE$ message on the tunnel. The ID has the most significant bit set to 0 and the second most significant bit depends of the relation between the public keys of the endpoints to ensure the IDs generated by the endpoints do not collide with each other.

The destination receives the incoming channel and checks the port number for local applications listening on that port. If an application is listening, it sends a $CHANNEL\_ACK$ message to the origin peer in order to confirm the channel and notifies the application about an incoming channel. Otherwise it waits for an application to open a port. CADET does not send any indication that a port may be closed to make it harder to probe for closed ports.

The first peer sends another $CHANNEL\_ACK$ to confirm the receipt of the first $CHANNEL\_ACK$ and finish the connection creation.

If the first peer receives a $CHANNEL\_NACK$ it destroys the channel and notifies the application about the failure.

When CADET is expecting a $CHANNEL\_ACK$ it starts a timer to retransmit the message it sent (either $CHANNEL\_CREATE$ or $CHANNEL\_ACK$) in case the message or the acknowledgement was lost.

Flow Control and Reliability

Similar to TCP streams, CADET offers reliable channels. CADET uses message ACKs for flow control on these channels, with a fixed window size of 64 messages. If a message is lost but subsequent messages are received, this can be indicated, since each ACK message has a 64 bit mask to signal "future" received messages. This allows the sender to retransmit only the messages that are really lost while sending subsequent messages in the window. ACK'ed packets are freed and the timing is used to adjust the retransmission delay for subsequent data.

Channels, in a similar fashion to UDP, have no flow control, although they benefit from the congestion control, since it is done on the connection level.

Communication session

Now that we have presented all CADET components, we describe how a typical CADET communication session is established. An example session with all elements can be seen in Figure 5.17

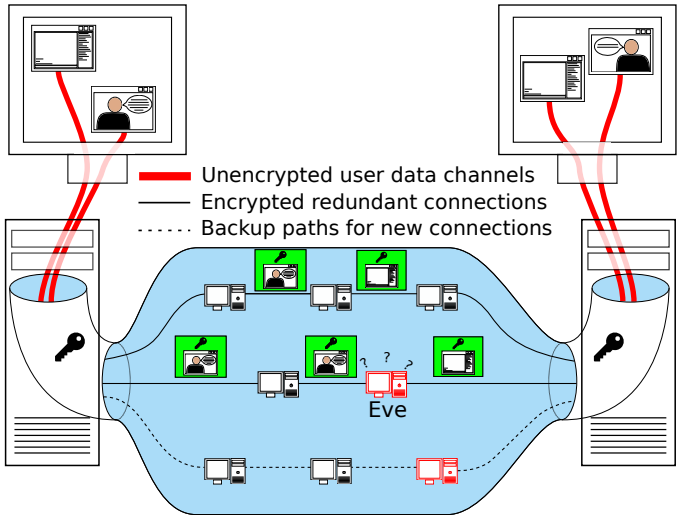


Figure 5.17: Channels inside a tunnel sending messages over two connections with one known backup path. As all traffic is encrypted, Eve cannot gain any insight, even if either peer's keys are later compromised.

First a client application requests a new channel to a certain destination peer and port number. The destination peer is defined by its public key and the port number is a 512 bit value. Applications should use port values that are non-public to ensure that only authorized clients can even connect. For example, a simple method would be to pick a value at random for services advertised via the GNU Name System, and to communicate the port number and the peer's public key as part of an application-specific GNS record.

CADET receives the request and checks whether there is an existing connection. If there is already an established connection towards the destination peer, this connection is used. Otherwise a new connection is established.

Then CADET checks for an existing tunnel towards the destination peer. If there is no tunnel, CADET creates it. Both endpoints use the existing connections to perform a handshake. In the handshake the peers authenticate each other and establish the session keys to communicate securely. If there is a tunnel when the request arrives (for instance, because another application running on the peer is already communicating with the remote peer, or did so recently), the existing tunnel is used.

The service creates a new channel inside the tunnel, allocating a unique number and specifying the destination port. If there is an application on the

destination peer listening to the given port, the channel is created. On the network layer, the protocol transmits $H(port|destination)$ where "destination" is the public key of the destination. Furthermore, the acknowledgement which confirms that the port is open includes port. This way, both parties prove to each other that they know the shared secret, in case the port is a secret. Including the "destination" in the hash ensures that the destination cannot just forward the request to another peer that has the port open to learn the secret. We do not include the sender in the hash as this would prevent using an efficient hash table lookup to handle incoming connections.

Afterwards, the two processes can communicate on the new channel by using the newly created channel. The tunnel is shared between all channels and each message sent uses one of the available connections.

If either of the endpoints does not need the channel anymore, it requests its destruction. The service notifies the other endpoint about the channel's destruction.

When the last channel inside a tunnel is destroyed the service starts a timer for the tunnel. After the timeout, the service destroys the tunnel, deleting the encryption keys and destroying all established connections.

5.5 Implementation

We have implemented CADET in the GNUnet peer-to-peer framework, using the R^5N DHT for routing GET and PUT requests in the resulting restricted environment.

5.6 Results

We evaluated CADET using the implementation in GNUnet and GNUnet's testbed infrastructure [Tot13a] which can be used to deploy, control and observe thousands of peers on a workstation PC. For the tests, a small network with 100 peers was used. Each of the experiments was repeated 10 times with a different random underlay topology being emulated each time. We introduce an artificial 20 ms round-trip delay to simulate a high-speed network.

All experiments were done running the normal GNUnet production code, including operating system scheduling, crypto operations and network traffic.

5.6.1 Churn Resistance

In the connectivity test we start a network with all peers online and an average of 21 random connections per peer. We select 10% of them to ping other randomly selected peers. Then we start churning the network stating at 80% and rapidly decreasing by 10% every 10 seconds until we reach 20%. We just churn non-participating (relay) peers, we never shut down peers doing or receiving pings. It is possible that at the end, when only peers sending and receiving pings are left, some of them might be disconnected from the network.

We contemplate three different scenarios. In the most favorable, peers are allowed to freely connect to other peers as they see fit. This would correspond to internet peers with public IP addresses and no blocking firewall. In a more restricted scenario we don't allow peers to establish new connections, but before starting the test we "warm up" the peers: we establish a random connection from each peer on the network to another random peer. This allows all peers to observe the $CONNECTION\_CREATE$ messages to which they are relays and better learn the topology of the network. In the most unfavorable scenario, we start the network and right away we demand that the peers start their benchmark connections, for what they must discover the routes via DHT and optimize them later, if possible. We choose to only perform pings with 10% of the peers, so peers do not learn too much of the topology from each other's connections and keep it differentiated from the "warm" case.

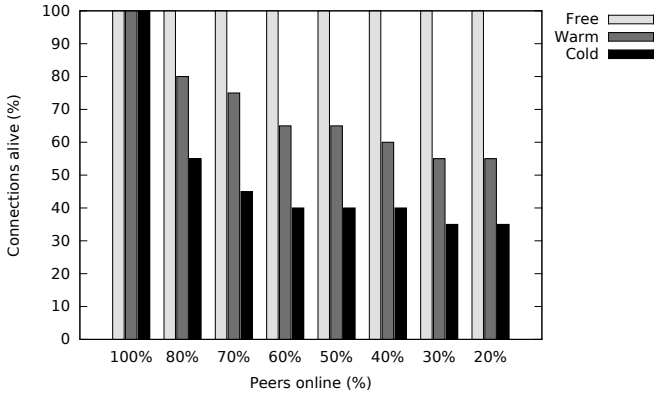


Figure 5.18: Percentage of peers that keep their connections as network is churned. We only counted connections between peers that remained up under the simulated churn. The test labelled as "Free" peers are allowed to establish new connections. In the "Warm" test the peers have time to learn the topology of the network prior to the churn. In the "Cold" test, peers have to find a route to their destination as the network is started.

In Figure 5.18 we see that in all three scenarios every peer manages to find a route to their destination. As we churn the network and peers start to disconnect, however, the results vary. In the case of "Free" peers, they never lose connectivity because they manage to quickly establish direct *Core* connections to their targets. As they no longer need relays, they remain unaffected by churn. In the restricted route scenarios, we observe that knowledge about the topology accumulated during the warm-up phase allow peers in the "warm" scenario to keep their connectivity much longer, due to knowledge of alternative paths and the redundant connections that it allows. Even with 80% of the network gone, more than 50% of peers manage to stay connected to their targets. In the worst case scenario, almost 40% of peers manage to stay connected after losing 80%

of the network in just a minute.

	Free	Warm	Cold
Avg	44.26 ms	49.18 ms	53.39 ms
std dev	13.97 ms	17.93 ms	18.64 ms
first round	55.99 ms	55.87 ms	62.38 ms
last round	38.52 ms	35.44 ms	44.98 ms

Table 5.2: Average and standard deviation of latency among peers running under different network conditions.

Additionally, in Table 5.2 we can see the average latency in the pings the peers performed. Keep in mind that the test was run with an emulated 20ms round-trip latency. We can see that, having gathered more information about the network topology, the peers in the warm scenario achieve to make initial connections with shorter paths, which leads to lower latency times. In the case of the free scenario, the lack of initial information is compensated by establishing new, direct connections. Looking at the latency in the last round, we can observe why the peers without the extra initial information lost connectivity more frequently: their average connection is sensibly longer, therefore more vulnerable to being disrupted by disconnected relay peers.

5.6.2 Latency

In order to measure the impact of the additional layers in latency we measured the round-trip times to machines running the normal unmodified GNUnet code on different types of networks, as pictured in Figure 5.19. First we tested three types of networks with computers with direct connectivity: in a Local Area Network with a direct cable connection (Peers A-C), in a Wireless Local Area Network on the same Access Point (Peers A-B) and between a university network and a typical home 16Mbit/1Mbit ADSL internet connection (Peers A-D). The fourth scenario involves two computers directly connected with a cable (A-C), of which only A has internet connectivity. The computer with no internet connectivity (C) pings the peer only reachable over the internet (D), using A as a relay. The control results are obtained with a simple ICMP ping. Each connection is tested over 150 round-trips. The results are presented in Table 5.3.

The LAN connection has the biggest proportional latency penalty, since the ICMP case is very close to hardware performance and nearly instantaneous. Traffic in GNUnet has to traverse multiple layers in userspace and has the overhead of encryption. This causes the latency to be 14ms compared to the 0.2ms of just ICMP, which is almost a hundred times more. The absolute penalty is even bigger in a wireless environment. The bigger packet size of the GNUnet

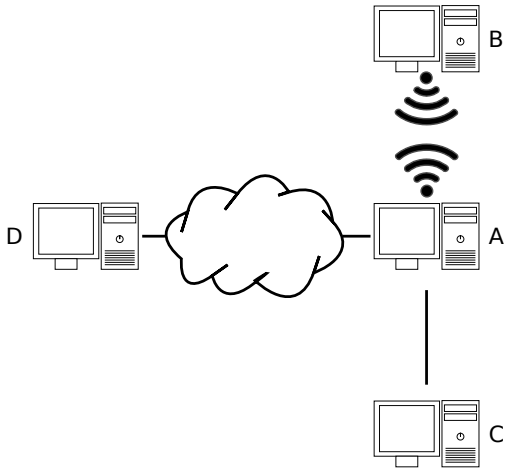


Figure 5.19: The test setup used for performance measurements. Peers B and C have no internet connectivity, only A can talk to D.

	ICMP		CADET	
	Avg	Std Dev	Avg	Std Dev
LAN	0.211 ms	0.042 ms	14.31 ms	4.236 ms
WLAN	5.017 ms	3.854 ms	26.23 ms	15.05 ms
WAN	37.94 ms	1.013 ms	44.66 ms	1.321 ms
Relay	N/A	N/A	56.38 ms	5.021 ms

Table 5.3: Average latency between systems running GNUnet on different connections. The latency penalty is less relevant over Internet connections.

ping probes causes collisions in the Access Point and the forced retransmissions increase the latency 21ms over the ICMP case, and the standard deviation spikes to 15ms. When the test is performed over a home internet connection the proportional impact is much smaller. In this case the pings over GNUnet are only 6.5 milliseconds longer with a very similar latency. This reduction in overhead can be explained by the computing power available: while the Local Area computers B and C are laptops the computer D is a workstation.

Finally, when we ping the workstation from a computer without internet connectivity, we obtain 56ms, which is a latency 3ms shorter than the LAN + Internet latencies combined. This is due to the fact that the relay node does not need to perform encryption on the traffic.

5.6.3 Bandwidth

In Table 5.4, we present the results of our bandwidth tests. In the same setup as the latency measurements we transferred several files between the computers. For the LAN test the files were 100MB. For Wireless LAN we used 100MB and 10MB files. For test over the internet, due to the very slow 1Mbps upload speed of the ADSL line, we used 1MB files. Each transfer was repeated 10 times. Between test repetition the GNUnet tunnels were not torn down, so only the first transfer is affected by the full handshake (connection and tunnel establishment). All subsequent transfers benefit from tunnel reuse.

To compare pure TCP speed we used the program `netcat`, as we wanted to avoid any overhead from other protocols, like HTTP or FTP. For CADET we used a `netcat`-equivalent command-line tool.

	TCP	CADET
LAN	105 MB/s	15.0 MB/s
WLAN	4.97 MB/s	5.01 MB/s
WAN	114 KB/s	103 KB/s
Relay	N/A	110 KB/s

Table 5.4: Average bandwidth in the different network types.

For the high speed LAN test the bandwidth for GNUnet was much smaller than with raw TCP. This is caused by the high CPU utilization for crypto and user space overhead. During these tests the CPU was operating at 100% while for TCP the load was not noticeably higher than idle. A faster CPU would help GNUnet offer better performance results for LAN speeds. On the Wireless LAN tests both protocols offer very similar results, as is the case in the internet results. When the CPU is free we see that GNUnet performs almost as well as TCP.

We observe that the test for relay traffic yields better bandwidth than the direct connection. We repeated this test, with similar results. The difference is small, so it is possible to explain it by fluctuations on the ADSL line.

5.7 Related Work

5.7.1 TCP/IP

TCP/IP [Pos81] shares most of the functionality with CADET. The three distinct functions (connectivity, state maintenance and multiplexing) are divided in just two layers, with IP providing connectivity, TCP or UDP multiplexing and the state maintenance shared between the two, with each connection defined by a 5-tuple of IP addresses, TCP/UDP ports and protocol. This causes

any change in the connectivity (switch to a different address, for instance) to break all established communications. In CADET, all application level communication happens in channels inside a tunnel, which is independent from the connection used to send the data, therefore connections can and do change constantly without affecting the application.

Regarding the individual components, IP solves the connectivity problem using hierarchical routing. This requires an authority to assign the addresses and a coordinated protocol to spread the routing information over the network, which GNUnet avoids as a design principle. Additionally, besides the security problems inherent in a trust-requiring environment, needed for the routing algorithms (BGP, OSPF) to work; as the address space is exhausted, fragmentation causes the routing tables to grow exponentially in size. While IPv6 targets some of this issues, it is by far not a complete solution.

TCP solves the reliability problem with byte-oriented ACK while CADET uses per-message ACKs, since GNUnet's Transport layer does not deliver fragmented messages to the Core layer. The biggest difference, however, is the flow and congestion control. TCP uses a *sliding window* for flow control and different *congestion windows* for congestion control. The basic mechanism is additive increase and multiplicative decrease in the window size. The window grows slowly until a duplicated ACK is received, which means a packet was lost, most probably due to a router dropping it due to a full buffer. The reaction to this depends on the implementation, but usually implies reducing the window size by a big step and resuming the slow increase. Since UDP does not use ACK, it lacks flow or congestion control. Since CADET uses ACKs for congestion control, it is guaranteed that it will never drop a message due to full buffers. This also allows the unreliable, UDP-like mode to benefit from congestion control and a rudimentary flow control. The extra traffic generated by the hop-by-hop congestion control ACK is minimized by using very big message sizes, which minimize the overhead used by the ACKs.

A related transport protocol, SCTP [Ste07], designed as part of the TCP/IP stack, implements some improvements like message-oriented transmissions, multihoming, multi-streaming and individual selection of features like in-order delivery and reliability. However, it still does not offer payload security and still suffers from the problems associated with BGP routing.

5.7.2 Tor

While Tor [DMS04a] also uses peer-to-peer relay nodes to forward data between endpoints, the goals of the systems are totally different. Tor uses forwarding to achieve anonymity and requires all nodes to be able to communicate with one another to create random paths among all the peers in the network. CADET allows environments with limited connectivity and uses forwarding to improve this connectivity, therefore using the shortest path it can find among a restricted set of peers. Tor uses onion encryption to hide the full paths from each relay peer, while CADET keeps them open in the hope that other peers can gain a better idea of the topology of the network and optimize their own connections.

5.7.3 net2o

net2o [Pay09] also uses source routing, but instead of establishing a connection by explicitly allocating resources at every relay peer on the path, net2o uses a bitmap in the message header to tell every hop which neighbor to send the message to. Each hop consults this bitmap and retransmits the message to the neighbor whose number is indicated in the field corresponding to it. This way, the relay does not need to store any information regarding the connections that traverse it. On the other hand, the source peer must know all the neighbors of all the relays on the path, as well as find the path itself, which is not trivial and the solution proposed by net2o has exponential complexity. While this method might be possible in an infrastructure setting where the neighbors of a core router very rarely change, the dynamic nature of Peer-to-Peer scenarios make this impracticable, as it would require constant updates to the peering information of each relay.

5.8 Conclusion

We have presented the first protocol for securely and efficiently estimating the size of a P2P network. Our protocol combines proximity to a deterministic sequence of (pseudo-)random values, staggered triggering of messages and a proof-of-work component. The scheme works for structured and unstructured networks, is inexpensive in terms of bandwidth, perfectly distributed imposing equal requirements in terms of computation and bandwidth on all nodes, and is quite accurate even for networks under churn. The protocol is secure against adversaries trying to make the network appear smaller and makes it computationally expensive (based on a parameter W) to make the network appear larger or to flood the network with unwarranted traffic.

Using the network size estimation algorithm, we have presented a robust routing algorithm for restricted-route networks. Our R^5N algorithm combines a random walk with a recursive variation of Kademlia and uses forwarding to multiple targets along the path for replication and redundancy. R^5N is robust against a range of some well-known attacks on DHTs, including poisoning attacks, Sybil attacks and Eclipse attacks. Its performance is generally comparable to that of a recursive implementation of Kademlia and outperforms other DHTs in restricted-route topologies.

Finally, using R^5N, we have created CADET, a secure decentralized transport for Ad-Hoc networks. CADET provides a secure communication channel by combining active DHT searches with passive monitoring to achieve good connectivity in most scenarios. We have shown that, while having notably higher latency and resource utilization than TCP in fast, high-bandwidth networks, in Internet-like scenarios the practical differences are minimal. We also have demonstrated that CADET adapts quickly to dramatic changes in the topology of the network, as it may be the case in peer-to-peer networks.

Chapter 6

The GNU Name System

This chapter is based on [WSG13, WSG14]. These papers were co-authored with Matthias Wachs and Martin Schanzenbach.

6.1 Introduction

The Domain Name System (DNS) is a unique distributed database and a vital service for most Internet applications. While DNS is distributed, it relies on centralized, trusted registrars to provide globally unique names. As the awareness of the central role DNS plays on the Internet rises, various institutions are using their power (including legal means) to engage in attacks on the DNS, thus threatening the global availability and integrity of information on the Web [Ess14]. This danger has also been recognized by the European Parliament, which has emphasized the importance of maintaining free access to information on the Web in a resolution [Eur11]. Tampering with the DNS can cause collateral damage, too: a recent study [Ano12] showed that Chinese censorship of the DNS has had worldwide effects on name resolution. At the same time, we observe that the Internet's importance for free communication has dramatically risen: the events of the Green Revolution in Iran and the Arab Spring have demonstrated this. Dissidents need communication channels that provide the easy linking to information that is at the Web's core. This calls for a censorship-resistant name system which ensures that names of Internet servers can always be resolved correctly.

DNS was not designed with security as a goal. This makes it very vulnerable, especially to attackers that have the technical capabilities of an entire nation-state at their disposal. The following are some of the most severe weaknesses that the DNS exhibits even in the presence of the DNS Security Extensions (DNSSEC). DNSSEC [AAL+05] was designed to provide data integrity and origin authentication to DNS. DNSSEC maintains the hierarchical structure of DNS and thus places extensive trust in the root zone and TLD operators. More importantly, DNSSEC fails to provide any level of query privacy [Ber08]: the

content of DNS queries and replies can be read by any adversary with access to the communication channel and can subsequently be correlated with users. On a technical level, current DNSSEC deployment suffers from the use of the RSA crypto system, which leads to large key sizes. This can result in message sizes that exceed size restrictions on DNS packets, leading to additional vulnerabilities [HS13]. Finally, DNSSEC is not designed to withstand legal attacks. Depending on their reach, governments, corporations and their lobbies can legally compel operators of DNS authorities to manipulate entries and certify the changes, and Soghoian and Stamm have warned that similar actions might happen for X.509 server certificates [SS11]. There can also be collateral damage: DNSSEC cannot prevent problems such as the recent brief disappearance of thousands of legitimate domains during the execution of established censorship procedures, accidentally requested the removal of 8,000 (legitimate) domain names from DNS and providers complied. The underlying attack vector in these cases is the same: names in the DNS have owners, and ownership can be taken away by different means.

To prevent such attacks, we need a censorship-resistant name system ensuring availability and resilience of names. For such a censorship-resistant name systems, this chapter advocates a solution in line with the ideas of the GNU project. Richard Stallman, founder of the GNU project, writes [Sta12]: "When a program has an owner, the users lose freedom to control part of their own lives." Similarly, ownership of a name implies the existence of some authority to exercise control over the property, and thus implies the possibility of coercion of that authority. Cryptographic identifiers can be created without the need for an authority; similarly, when users locally assign values to private labels, as done in *petname* systems, such personal labels also cannot be owned or confiscated.

This chapter presents the GNU Name System (GNS), a censorship-resistant, privacy-preserving and decentralized name system designed to provide a secure alternative to DNS, especially when censorship or manipulation is encountered. As GNS can bind names to any kind of cryptographically secured token, it can double in some respects as an alternative to some of today's Public Key Infrastructures, in particular X.509 for the Web.

The foundation of the GNS system is a petname system [Sti05], where each individual user may freely and securely map names to values. In a petname system, each user chooses a *nickname* as his preferred (but not necessarily globally unique) name. Upon introduction, users adopt the nickname by default as a *label* to refer to a new acquaintance; however, they are free to select and assign any *petname* of their choice in place of—or, in addition to—the nickname. Petnames thus reflect the personal choice of the individual using a name, while nicknames are the preferred name of the user that is being identified.

The second central idea is to provide users with the ability to securely delegate control over a subdomain to other users. This simple yet powerful mechanism is borrowed from the design of SDSI/SPKI. With the combination of petname system and delegation, GNS does not require nor depend on a centralized or trusted authority, making the system robust against censorship attempts. Decentralization and additional censorship-resistance is achieved by using a dis-

tributed hash table (DHT) to enable the distribution and resolution of key-value mappings. In theory, any DHT can be used. However, depending on the properties of the DHT in question, varying degrees of censorship-resistance will be the result. As such, the choice of the DHT is crucial to the system. Finally, GNS is privacy-preserving since both key-value mappings as well as queries and responses are encrypted such that an active and participating adversary can at best perform a confirmation attack, and can otherwise only learn the expiration time of a response.

While this combination yields a secure name system, it also violates a fundamental assumption prevailing on the Web, namely that names are globally unique. Thus, together with the working implementation of GNS, another key contribution of our work is the construction of system components to enable the use of GNS in the context of the Web. We provide ready-to-use components to enable existing Web applications to use GNS (and DNS in parallel, if desired) without any prior modifications and knowledge.

As an alternative public key infrastructure, GNS can also be combined with existing PKI approaches (such as X.509, DANE [Bar11], Tor's ".onion" or the Web-of-Trust) to either provide memorable names or alternative means for verification with increased trust agility. In combination with TLSA records, GNS can replace existing X.509 certification authorities as described in Section 6.8.3.

6.2 Requirements Analysis

To analyze the requirements a censorship-resistant name system has to fulfill, we start with a practical adversary model and the attacks a system has to withstand. Based on these, we then develop functional requirements for a censorship-resistant name system.

6.2.1 Adversary Model

The adversary used in this paper is modeled after nation state trying to limit access to information on the Internet. Our adversary can participate in any role in the system and can also assume multiple identities (Sybils) without an upper bound in relation to the total number of participants. The adversary can take over control of names using judicial or executive powers and is allowed to have more computational power then all benign users. This model excludes the use of a trusted third party.

On the other hand, the adversary cannot break cryptographic primitives and not prevent the usage of cryptography or encrypted communication. The adversary is also not able to take direct control of the systems of individual users, or at least if he does so, the system does not have to protect the users that are directly affected by such actions. As far as network communication is concerned, we assume that the adversary cannot generally prevent communication between benign participants.

Our adversary's goal is to prevent access to information on the Web by affecting the name resolution process, either by making it fail or by changing the value associated with an existing name. He can do so by influencing or controlling parties participating in the name system.

Some name systems were designed with a weaker adversary model in mind; in particular, the assumption that the adversary does not control the majority of the nodes or the majority of the computing power is a popular model in computer security in general. However, censorship-resistance is typically an issue for activists, and thus hardly a topic for the majority of Internet users. As a result, it is unlikely that any censorship-resistant name system is going to be used widely enough to compete with the computational power available to major governments. Thus, we advocate using the assumption that the adversary might have more computational power than all other participants combined.

6.2.2 Functional Requirements

The basic functionality of a name system for the Internet is to map memorable names to correct values. After all, name resolution provides names for systems such that human beings can easily remember them, instead of having to remember the more complicated (and possibly frequently changing) address values used by the network.

One of the most important Internet services is the Web, and a fundamental building block for Web services is the ability to link to information hosted on different systems; as humans often manually create these links, links are specified using names. Thus, a name system should be designed to support link resolution: a service provider must be able to link to a foreign resource, and the users of the service must then be able to resolve the name to an address for the intended destination.

6.3 Design Space for Name Systems

This section explores the theoretical design space for name systems; we will structure our discussion on how a name system can be realized using Zooko's triangle [WO06], an insightful conjecture on the design space for name systems (Figure 6.1).

Definition 1 (Memorable)**.** *A name is memorable if it is feasible for an attacker in our adversary model to obtain it by enumerating names (bit strings). In other words, the number of bits of entropy in a memorable name is insufficient against enumeration attacks.*

Definition 2 (Secure)**.** *A secure name system must enable benign participants to register and retrieve correct name-value mappings while experiencing active, malicious participants (which are assumed to follow the adversary model described in Section 6.2.1).*

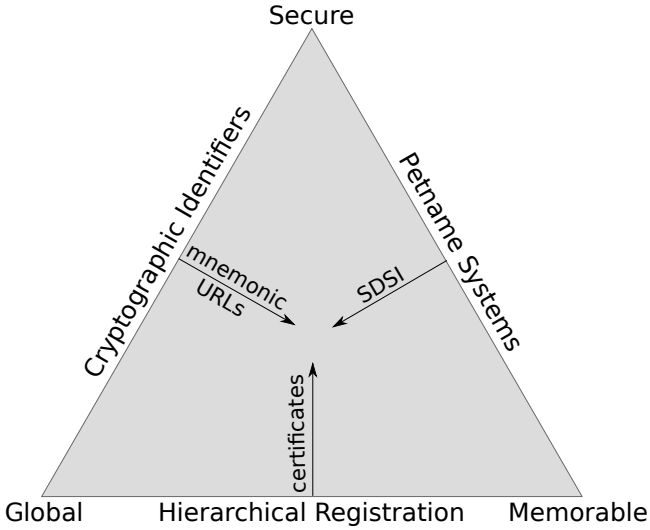


Figure 6.1: Illustration of Zooko's triangle and key approaches to name systems.

Definition 3 (Global)**.** *The system supports an unlimited number of participants without prior coordination or certification of participants. All benign participants receive the same (global) values for the same names.*

Theorem 2 (Zooko's triangle)**.** *It is impossible to have a name system that achieves* **memorable**, **secure** *and* **global** *names at the same time.*

We confirmed with Zooko Wilcox-O'Hearn that these definitions represent the intended interpretation of his formulation. We show now that Zooko's triangle is a valid conjecture *in our adversary model*:

Proof. All participants, including the adversary, are supposed to be able to register names under the "secure" property of the name system. As names are memorable, an adversary can enumerate all possible names. Thus, the adversary can perform a squatting attack by (if necessary) assuming the identities of name system components that restrict registration and performing the necessary computations (we assumed he is able to do those faster than the rest of the network combined). The adversary can use this attack to register *all* memorable names. As names are global, once the adversary has registered a name, that name can no longer be registered by anyone else.

Thus, the squatting attack can prevent the registration of memorable names by normal participants. Thus, in our security model, it is impossible to create a secure, global name system where memorable names are guaranteed to be available for registration by normal users. □

A trusted authority in control of name assignments would easily prevent such an attacker from being successful; however, the existence of such an authority

is outside of our security model. We would like to point out that the proof given above is controversial in the research community. We have had comments from reviewers ranging from assertions that the theorem is a trivial (or at least well-known) fact that does not require proof, to those questioning its veracity. We believe that this is the first formalization of Zooko's hypothesis and that the theorem holds in our security model — and that it is false under weaker assumptions. Thus, any name system *in our security model* must deemphasize one of the properties from Definitions 1–3. Figure 6.1 describes the three major design approaches in this context. The edges of the triangle represent the three simple designs, and the arrows towards the middle represent the three main designs which move toward satisfying all three properties.

6.3.1 Hierarchical Registration

In Zooko's triangle, a name system using hierarchical registration is a name system providing global and memorable names; however, in the hierarchical structure names are owned by organizations. These organizations receive the power to manage a subspace of the namespace by delegation from an organization ranked higher in the tree, which enables censorship. The well-known Domain Name System (DNS) is a distributed database realizing access to a name system with such an hierarchical structure.

The DNS is an essential part of the Internet as it provides mappings from host names to IP addresses, providing memorable names for users. DNS is hierarchical and stores name-value mappings in so-called *records* in a distributed database. A record consists of a name, type, value and expiration time. Names consist of *labels* delimited by dots. The root of the hierarchy is the empty label, and the right-most label in a name is known as the top-level domain (TLD). Names with a common suffix are said to be in the same *domain*. The *record type* specifies what kind of value is associated with a name, and a name can have many records with various types. The most common record types are "A" records that map names to IPv4 addresses.

The DNS database is partitioned into *zones*. A *zone* is a portion of the namespace where the administrative responsibility belongs to one particular authority. A zone has unrestricted autonomy to manage the records in one or more domains. Very importantly, an authority can delegate responsibility for particular *subdomains* to other authorities. This is achieved with an "NS" record, whose value is the name of a DNS server of the authority for the subdomain. The *root zone* is the zone corresponding to the empty label. It is managed by the Internet Assigned Numbers Authority (IANA), which is currently operated by the Internet Corporation for Assigned Names and Numbers (ICANN). The National Telecommunications and Information Administration (NTIA), an agency of the United States Department of Commerce, assumes the (legal) authority over the root zone. The root zone contains "NS" records which specify names for the authoritative DNS servers for all TLDs.

The Domain Name System Security Extensions (DNSSEC) [AAL+05] add integrity protection and data origin authentication for DNS records. DNSSEC

does not add confidentiality nor denial-of-service protection. It adds "DNSKEY" record types for public keys and for signatures on resource records ("RRSIG"). DNSSEC relies on a hierarchical public-key infrastructure in which all DNSSEC operators must participate. It establishes a trust chain from a zone's authoritative server to the trust anchor, which is associated with the root zone. This association is achieved by distributing the root zone's public key out-of-band with, for example, operating systems. The trust chains established by DNSSEC mirror the zone delegations of DNS. With TLD operators typically subjected to the same jurisdiction as the domain operators in their zone, these trust chains are at risk of attacks using legal means.

6.3.2 Cryptographic IDs and Mnemonics

A name system that can securely map globally unique identifiers to values can be achieved using cryptographic identifiers as names. In such a system security is achieved by certifying values using the private key associated with the cryptographic identifier. Names are with high probability globally unique. However, as cryptographic identifiers are long random bitstrings, they are not memorable. An example for a deployed name system with cryptographic identifiers is Tor's ".onion" namespace [DMS04b].

The proposed Tor mnemonic URL system [Sai12] aims to make the ".onion" names more memorable by encoding the hashes names into "human-meaningful" sentences. However, the resulting names will not be memorable by our Definition 1 as the high entropy of the original cryptographic identifiers remains. As (assuming sufficiently strong cryptographic primitives are used) an adversary would not be able to enumerate all cryptographic identifiers, Tor's mnemonic URL system would not result in memorable names as those names correspond to cryptographic identifiers and thus could also not be enumerated. Finally, it is important to note that Tor's mnemonic URLs are still work in progress; it is thus difficult to assess the usability of this approach.

6.3.3 Petnames and SDSI

A secure name system with memorable names can be created using so-called *petnames*. In a petname system, each user establishes names of his choice for other entities [Sti05]. Each user or service would be identified using a cryptographic identifier based on a public key; the service provider can then sign mapping information to certify integrity and authenticity of the data. Memorable names are achieved by mapping petnames to cryptographic identifiers. While such a system can provide security and memorability, the mappings are only local and petnames are meaningless (or have a different meaning) for other users. A simple example of a petname system is the `/etc/hosts` file that allows administrators to create a mapping from hostnames to addresses for the local system.

Extending petname systems with ideas from Rivest's Simple Distributed Security Infrastructure (SDSI) [RL96] adds the possibility of (secure) *delegation*,

allowing users to reference the petnames of other users. SDSI based delegation enables users to resolve other participant's names and thus enables linking to external resources. Delegation essentially adds another user's namespace in a subtree under a specific name. This creates an hierarchical namespace from the point of view of each user; globally, the resulting structure is simply a directed graph. While delegation broadens the accessibility of mappings, it does not achieve global names.

SDSI/SPKI is a merger of the Simple Distributed Security Infrastructure (SDSI) and the Simple Public Key Infrastructure (SPKI) [RL96]. It defines a public-key infrastructure that abandons the concept of memorable global names and does not require certification authorities. SDSI/SPKI has the central notion of *principals*, which are globally unique public keys. These serve as namespaces within which local names are defined. A name in SDSI/SPKI is a public key and a local identifier, e.g. $K-Alice$. This name defines the identifier $Alice$, which is only valid in the namespace of key K. Thus, $K_1-Alice$ and $K_2-Alice$ are different names. SDSI/SPKI allows namespaces to be linked, which results in compound names: $K_{Carol}-Bob-Alice$ is Carol's name for the entity which Bob refers to as $K_{Bob}-Alice$. Bob himself is identified by Carol as $K_{Carol}-Bob$. SDSI/SPKI allows assertions about names by issuing certificates[1]. A *name cert* is a tuple of *(issuer public key, identifier, subject, validity)*, together with a signature by the issuer's private key. The *subject* is usually the key to which a name maps. Compound names are expressed as certificate chains.

6.3.4 Timeline-based Name Systems

Timeline-based name systems, such as the Namecoin system [dot13], manage to combine global names, memorable names and security. In these systems, a global timeline with the domain registrations is secured by users performing proof-of-work computations, which in turn are used as "payment" for name registration.

Their existence does not contradict Zooko's triangle as their security depends on the adversary not having more computational power than the honest nodes; an adversary with sufficient computational power can create an alternative timeline with different domain registrations and a stronger proof-of-work, which would ultimately result in the system switching to the adversarial timeline. Thus, timeline-based systems do not fit the realistic adversary model we assumed for this paper (Section 6.2.1).

6.4 Practical Considerations

The previous section has outlined the design space for censorship-resistant name systems. However, implementations of these alternatives will have to address a range of technical and practical concerns which be will discussed here.

[1] Ultimately, SDSI/SPKI allows to create authorizations based on certificates and is a flexible infrastructure in general, but we will focus only on the names here.

6.4.1 Interoperability with DNS

To be accepted by users, a censorship-resistant name system should respect user's usage patterns and integrate with existing technologies. Users should not have to manually switch between alternative name systems and DNS. Syntax and semantics of the different name systems should also be similar to not confuse the user about the meaning of names.

Thus, a central requirement for any alternative name system will be interoperability with DNS. Users are used to DNS names and virtually all network applications today use DNS for name resolution. Thus, being interoperable with DNS will allow censorship-resistant alternatives to be used with a large body of legacy applications and facilitate adoption by end-users.

Interoperability with DNS largely implies that alternative name systems should follow DNS restrictions on names, such as limiting names to 253 ASCII characters, limiting labels to 63 characters and using Internationalizing Domain Names in Applications (IDNA) [FHC03] for internationalization. Furthermore, the name system should be prepared to return standard DNS records (such as "A" and "AAAA") to typical applications.

Interoperability with DNS should also include accessing the information of DNS from within the namespace of the censorship-resistant name system. For example, it is conceivable that a censor might block access to `www.example.com` by removing the nameserver information for `example.com` in the `.com` TLD, without blocking access to the nameserver of `example.com`. In this case, a censorship-resistant name system only needs to provide an alternative way to learn the nameserver for `example.com` — the lookup of `www` can then still be transmitted directly to the authoritative nameserver. In an alternative name system supporting delegation, this simply requires support for delegating subdomains back to DNS. This allows users to bypass censorship closer to the root of the DNS hierarchy even if the operators of the censored service do not explicitly support the censorship-resistant name system.

Finally, for good interoperability users must not be required to exclusively use an alternative domain name system — alternating between accessing DNS for domain names that are not censored and using the censorship-resistant name system should not require the user to reconfigure his system each time!

Interoperability and using multiple name systems with the same configuration can be easily achieved using pseudo-TLDs. A pseudo-TLD is a top level domain that is not actually participating in the official DNS. For example, using the pseudo-TLD ".key", a user might specify "ID.key" to access a name system based on cryptographic identifiers, or "NICK.pet" to access a pseudo-TLD ".pet" for petnames. Naturally, this only works as long as the names chosen for the pseudo-TLDs are not used by the global DNS.

Once pseudo-TLDs have been selected, the local DNS stub resolver can be configured (for example, using the Name Service Switch [Fre17]) to apply special resolution logic for names in the pseudo-TLDs. The special logic can then use alternative means to obtain and validate mappings, which will work as long as the final results returned can be again expressed as a DNS response.

6.4.2 End-to-End Security and Errors

Today, client systems typically only include a DNS stub resolver, delegating the name resolution process to a DNS resolver operated by their Internet Service Provider (ISP). As ISPs might be involved in censorship, they cannot be trusted to perform proper name resolution. Thus, secure name systems (including DNSSEC) must be deployed end-to-end to achieve the desired security.

This may not only require updating operating system resolvers. Existing applications sometimes implement their own DNS clients, and typical DNS APIs (such as POSIX's name resolution functions) do not include error reporting that incorporates security attributes. Browsers will thus be unable to benefit from TLSA records [HS12] until they either implement full DNSSEC resolver functions, or until operating system APIs are enhanced to allow returning additional information. A particularly critical example is the possibility to return unsigned records even within a DNSSEC deployment. As a result, DNSSEC protections can easily be disabled by replacing signed valid records with a set of invalid records without signature information.

6.4.3 Petnames and Legacy Applications

In addition to integration with existing systems an alternative name system also has to consider assumptions made by applications in higher layers, for example existing applications assuming globally unique names. Existing support for virtual hosting of websites in HTTP-based applications and TLS/SSL certificate validation both assume that the names given by the client match exactly the (DNS) name of the respective server. Links to external websites are typically specified using (globally unique) DNS names; as a result, relative names involving delegation from a SDSI-based name system would not be properly understood by today's browsers.

In lieu of directly modifying legacy applications, it might be possible to perform the necessary adaptations using proxies. Proxies might be used to translate hostnames from websites using delegation, and to perform SSL certificate validation (for example, by looking at TLSA [HS12] records from the secure name system instead of hostnames). Reverse proxies could be used to generate the virtual host names expected by the server, and to translate links with absolute links to those using the delegation chains provided by a SDSI-based name system. Additional records in the name system might be used to aid the conversion between relative names and legacy names by the proxies. In order to achieve end-to-end security, these proxies would naturally have to be operated within the trusted zone of the respective endpoints in the system.

6.4.4 Censorship-Resistant Lookup

Censorship-resistant distributed name systems need to consult name information from other participants and thus require a network protocol to perform censorship-resistant lookups. The most common method for implementing key-

based searches in decentralized overlay networks is the use of a *distributed hash table* (DHT).

Typical attacks on DHTs include poisoning and eclipse attacks. In a poisoning attack, the adversary attempts to make interesting mappings hard to find by placing many invalid mappings into the DHT. A censorship-resistant DHT for a name system that uses public keys to lookup values signed by the respective private key can easily defeat this type of attack by checking signatures. In an eclipse attack, the adversary tries to isolate particular key-value mappings from the rest of the network. Modern DHTs defend against this type of attack by replicating values at multiple locations [Pol10].

Some censorship-resistant DHTs such as X-Vine [MCB11] and R^5N [EG11b] additionally accept limited connectivity between the peers in the DHT, making it harder for the adversary to disrupt DHT operations in the IP layer. Furthermore, this also allows peers to restrict connections to known friends, making the DHTs more robust against Sybil attacks [Dou02] by building the overlay topology using existing social relationships.

One important property in this context will be query privacy. In existing centralized name systems, infrastructure providers can easily observe which names are used by which users. When the database is decentralized in a DHT, these central observation points are eliminated; however, now ordinary users can observe other user's queries, which maybe even more problematic for some applications. Thus, it is desirable to have encryption for queries and responses in the DHT. The encryption could be based on secrets only known to the user performing the resolution (such as the label and the zone); as a result, other users could only decrypt the resolution traffic with a confirmation attack where they would have to guess the label and zone of a query (or response). This would strengthen censorship-resistance as participants would typically not know which requests they are routing. Additional query privacy might be achieved by anonymizing the source of the request, for example by using onion routing [DMS04b]. Naturally, using such anonymization techniques increases latency.

6.4.5 Case study: Usability

Unlike DNS, the user's experience when using a name system based on SDSI depends on high-level user behavior: following a link corresponds to traversing the delegation graph and resolution is fully automatic. However, when users want to visit a fresh domain that is not discovered via a link, SDSI requires a trust anchor to be supplied via a registrar or out-of-band mechanisms, such as QR codes. This raises the question: how often are these inconvenient methods needed in practice?

To answer this question, we did a survey on surfing behavior. Specifically, we wanted to find out how often users would typically type in a new domain name for a site. A domain name is "new" if the user has never visited it before, and if the user is typing it in the name is also not easily available via some link. Typed in new domain names are thus the case where a SDSI-based name system (or PKI) would need to use some external mechanism to obtain the public key

of the zone.

Based on a limited and most likely biased survey where users volunteered the output of a simple shell script that inspected their browsers history database, we determined that given current Internet behavior, approximately 8% of domain names would require introduction via some out-of-band exchange. A key limitation of the survey's methodology was that we did not attempt to control who submitted results; we simply used the data of anyone who was willing and able to download and run the shell script that performed the analysis. This limited the sample to somewhat more technologically sophisticated users. The complete results from our survey and details on the methodology can be found in [Sch12]. Our conclusion is that a name system based on petnames and SDSI-style delegation stands a chance of being an acceptable choice if communication is hindered by censorship or strong security assurances (beyond those offered by the X.509 PKI or DNSSEC) are required.

6.5 Design of the GNU Name System

In the following, we describe the core concepts of GNS that are relevant to users. The cryptographic protocol used to ensure query privacy is explained in Section 6.6, and the protocol for key revocation in Section 6.8.5.

6.5.1 Names, Zones and Delegations

GNS employs the same notion of names as SDSI/SPKI: principals are public keys, and names are only valid in the local namespace defined by that key. Namespaces constitute the *zones* in GNS: a zone is a public-private key pair and a set of records. GNS records consist of a label, type, value and expiration time. Labels have the same syntax as in DNS; they are equivalent to local identifiers in SDSI/SPKI. Names in GNS consists of a sequence of labels, which identifies a *delegation path*. Cryptography in GNS is based on elliptic curve cryptography and uses the ECDSA signature scheme with Curve25519 [Ber06].

We realize a petname system by having each user manage his own zones, including, in particular, his own personal *master zone*.[2] Users can freely manage mappings for memorable names in their zones. Most importantly, they can delegate control over a subdomain to another user (which is locally known under the petname assigned to him). To this end, a special record type is used (see Section 6.5.5). This establishes the aforementioned delegation path. Each user uses his master zone as the starting point for lookups in lieu of the root zone from DNS. For interoperability with DNS, domain names in GNS use the pseudo-TLD ".gnu". ".gnu" refers to the GNS master zones (i. e. the starting point of the resolution). Note that names in the ".gnu" pseudo-TLD are always relative.

Publishing delegations in the DHT allows transitive resolution by simply following the delegation chains. Records can be private or public, and public

[2] Each user can create any number of zones, but must designate one as the master zone.

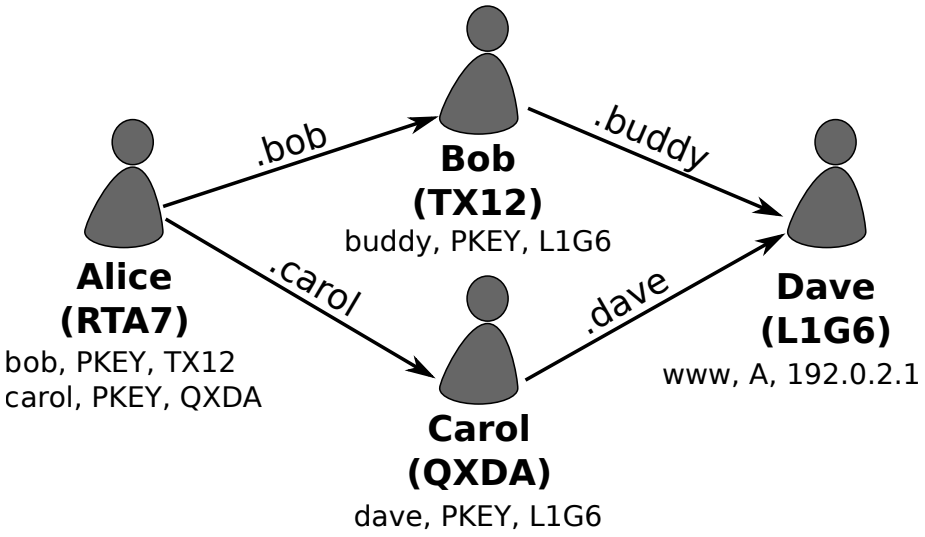


Figure 6.2: Name resolution graph in GNS. Each user is shown with a fingerprint of his master zone and the public records from this zone in the format *name, type, value*.

records are made available to other users via a DHT. Record validity is established using signatures and controlled using expiration values. The records of a zone are stored in a *namestore* database on a machine under the control of the zone owner.

We illustrate the abstract description above with the example shown in Figure 6.2. The figure shows the paths Alice's GNS resolver would follow to resolve the names "www.dave.carol.gnu" and "www.buddy.bob.gnu", both of which refer to Dave's server at IP "192.0.2.1". For Carol, Dave's server would simply be "www.dave.gnu". It is known to Alice only because both Bob and Carol have published public records indicating Dave, and Alice can resolve the respective delegation chain via her known contacts. Recall that zones are identified using public keys and records must be cryptographically signed to ensure authenticity and integrity.

6.5.2 Zone Management with Nicknames and Petnames

We now explain how the actual management of names is carried out in practice. Suppose Alice runs a web server and wants to make it available with GNS. In the beginning she sets up her master zone using GNS. After the public-private key pair is generated, Alice can create a revocation notice to be able to immediately revoke her GNS zone in case she gets compromised. Suppose Alice wants to propose that her preferred nickname is "carol" to other users. She therefore uses the new "NICK" record that GNS provides. In the value of this record,

she states that her nickname is "carol". For her web server, she creates an appropriate public "A" record under the name "www". This "A" record is the same as in DNS. To make it resolvable by other users, this record is marked as public and published in the DHT.

Now suppose we have a second user, Bob. He performs the same setup on his system, except that his preferred nickname is just "bob". Bob gets to know Alice in real life and obtains her public key. To be able to contact Alice and access her web server, he then adds Alice to his zone by adding a new delegation using the new "PKEY" record. Bob can choose any name for Alice's zone in *his* zone. Nevertheless, Bob's software will default to Alice's preferences and suggest "carol", as long as "carol" has not already been assigned by Bob. This is important as it gives Alice an *incentive* to pick a nickname that is (sufficiently) unique to be available among the users that would delegate to her zone. By adding Alice's public key under "carol", Bob delegates queries to the "*.carol.gnu" subdomain to Alice. Thus, from Bob's point of view, Alice's web server is "www.carol.gnu". Note that there is no need for Alice's nickname "carol" to be globally unique, they should only not already be in use within Alice's social group.

6.5.3 Relative Names for Transitivity of Delegations

Users can delegate control over a subdomain to another user's zone by indicating this in a new record, "PKEY". Suppose Dave is Bob's friend. Dave has added a delegation to Bob with a "PKEY" record under the name "buddy"—ignoring Bob's preference to be called "bob". Now suppose Bob wants to put on his webpage a link to Alice's webpage. For Bob, Alice's website is "www.carol.gnu". For Dave, Bob website is "buddy.gnu". Due to delegation, Dave can access Alice's website under "www.carol.buddy.gnu". However, Bob's website cannot contain that link: Bob may not even know that he is "buddy" for Dave.

We solve this issue by having Bob use "www.carol.+" when linking to Alice's website. Here, the "+" stands for the originating zone. When Dave's client encounters "+" at the end of a domain name, it should replace "+" with the name of the GNS authority of the site of origin. This mechanism is equivalent to relative URLs, except that it works with hostnames.

6.5.4 Absolute Names

In GNS, the ".gnu" pseudo-TLD is used to provide secure and memorable names which are only defined relative to some master zone. However, introducing new zones into the system ultimately requires the ability to reference a zone by an absolute identifier, which must correspond to the public key of the zone. To facilitate dealing with public keys directly, GNS uses the pseudo-TLD ".zkey", which indicates that the specified domain name contains the public key of a GNS zone. As a result, the ".zkey" pseudo-TLD allows users to use secure and globally unique identifiers. Applications can use the ".zkey" pseudo-TLD to

generate a domain name for a GNS zone for which the user does not (yet) have a memorable name.

A label in the ".zkey" pseudo-TLD must be the Crockford's Base32 encoded public key of a zone. We use the compressed point encoding of the 255-bit coordinates of Curve25519 [Ber06] to encode the name within the 63 character limitations for labels imposed by DNS. Names in the ".zkey" pseudo-TLD are resolved by querying the respective GNS zone. As each ".zkey" name uniquely identifies a public-private key pair, no authority is required to manage the ".zkey" pseudo-TLD.

6.5.5 Records in GNS

As GNS is intended to coexist with DNS, most DNS resource records from [Moc87, THKS03] (e. g., "A", "MX") are used with identical semantics and binary format in GNS. GNS defines various additional records to support GNS-specific operations. These records have record type numbers larger than 2^{16} to avoid conflicts with DNS record types that might be introduced in the future. We also introduce several new records, which we have described above and summarize here.

PKEY for delegation: "PKEY" records securely delegate control over a subdomain to another zone. Repeated delegation allows GNS to achieve transitivity of names. Secure delegation using "PKEY" records is central to GNS; it replaces the tree structure of DNS with a directed graph.

NICK for nicknames: This record type is used to specify the desired *nickname* for a zone. The value of the record consists of a label with the 63-character limit from DNS. If a nickname is desired for a zone, the same "NICK" record is added under *each* label of the respective zone; this ensures that the nickname is part of every response and thus no additional lookup is required to obtain the nickname.

GNS2DNS: "GNS2DNS" records delegate resolution for a subdomain from GNS to DNS.

Similar to "NS" records in DNS, the value in the "GNS2DNS" record is the name of the subdomain in DNS. In addition to the "GNS2DNS" record, the GNS zone must specify "A" or "AAAA" records under the same GNS label which specifies the IP address of the DNS resolver to contact for resolution (this is equivalent to the so-called glue records in DNS). For example:

	Name	RR Type	Value
Q:	www.example.gnu	A	
A:	example.gnu	GNS2DNS	example.com
A:	example.gnu	A	192.0.2.1
Q:	www.example.com *(DNS)*	A	
A:	www.example.com *(DNS)*	A	192.0.2.2

Given the first response, the GNS system will synthesize the DNS name "www.example.com" from the "GNS2DNS" record and the "www" remaining from the GNS name and send a DNS query to the DNS server at 192.0.2.1 based on the glue information from the "A" record. The resolution then continues using DNS. Note that this record type enables delegation to *DNS* from within GNS. Naturally, GNS cannot secure the DNS part of the resolution process.

LEHO: This record type specifies the legacy (DNS) hostname for a name in GNS. "LEHO" records are used to enable backwards-compatibility for virtual hosting and SSL certificate validation in combination with the client-side proxy. For example:

	Name	RR Type	Value
Q:	www.example.gnu	A	
A:	www.example.gnu	A	192.0.2.1
A:	www.example.gnu	LEHO	www.example.com

These are all the special record types that GNS needs. GNS maximizes compatibility with DNS by using the same length limits for labels and names, and the same encoding rules for internationalized names as DNS.

6.6 Query Privacy

To enable other users to look up records of a zone, all public records for a given label are stored in a cryptographically signed block in the DHT. To maximize user privacy when using the DHT to look up records, both queries and replies are encrypted. Let $x \in \mathbb{Z}_n$ be the ECDSA private key for a given zone and $P = xG$ the respective public key where G is the generator of the elliptic curve. Let $n := |G|$ and $l \in \mathbb{Z}_n$ be a numeric representation of the label of a set of records $R_{l,P}$. Using

$$h := x \cdot l \mod n \tag{6.1}$$

$$Q_{l,P} := H(hG) \tag{6.2}$$

$$B_{l,P} := S_h(E_{\mathtt{HKDF}(l,P)} R_{l,P}), hG \tag{6.3}$$

we can then publish $B_{l,P}$ under $Q_{l,P}$ in the DHT, where S_h represents signing with the private key h, HKDF is a hash-based key derivation function and E represents symmetric encryption based on the derived key. Any peer can validate the signature (using the public key hG) but not decrypt $B_{l,P}$ without knowledge of both l and P. Peers knowing l and P can calculate the query

$$Q_{l,P} = H(lP) = H(lxG) = H(hG) \tag{6.4}$$

to retrieve $B_{l,P}$ and then decrypt $R_{l,P}$.

Given this scheme, an adversary can only perform a confirmation attack; if the adversary knows both the public key of the zone and the specific label, he can perform the same calculations as a peer performing a lookup and, in this specific case, gain full knowledge about the query and the response. As the DHT records are public, this attack cannot be prevented. However, users can use passwords for labels to restrict access to zone information to authorized parties. The presented scheme ensures that an adversary that is unable to guess both the zone's public key and the label cannot determine the label, zone or record data.

6.7 Security of GNS

One interesting metric for assessing the security of a system is to look at the size of the trusted computing base (TCB). In GNS, users explicitly see the trust chain and thus know if the resolution of a name requires trusting a friend, or also a friend-of-a-friend, or even friends-of-friends-of-friends—and can thus decide how much to trust the result. Naturally, the TCB for all names can theoretically become arbitrarily large—however, given the name length restrictions, for an individual name it is always less than about 125 entities. The DHT does not have to be trusted; the worst an adversary can do here is reduce performance and availability, but not impact integrity or authenticity of the data.

For DNS, the size of the TCB is first of all less obvious. The user may think that only the operators of the resolvers visible in the name and their local DNS provider need to be trusted. However, this is far from correct. Names can be expanded and redirected to other domains using "CNAME" and "DNAME" records, and resolving the address of the authority from "NS" records may require resolving again other names. Such "out-of-bailiwick" "NS" records were identified as one main reason for the collateral damage of DNS censorship by China [Ano12]. For example, resolving "google.com" requires correct information from "x.gtld-servers.net" (the authority for ".com"), which requires trusting "X2.gtld-servers.net" (the authority for ".net"). While the results to these queries are typically cached, the respective servers must be included in the TCB, as incorrect answers for any of these queries can change the ultimate result. Thus, in extreme cases, even seemingly simple DNS lookups may depend on correct answers from over a hundred DNS zones [DSKM12]; thus, with respect to the TCB, the main difference is that DNS is very good at obscuring the TCB from its users.

In the following, we discuss possible attacks on GNS within our adversary model. The first thing to note is that as long as the attacker cannot gain direct control over a user's computer, the integrity of master zones is preserved. Attacks on GNS can thus be classified in two categories: attacks on the network, and attacks on the delegation mechanism.

Attacks on the network can be staged as Eclipse attacks. The success depends directly on the DHT. Our choice, R^5N, shows a particularly good resistance against such attacks [EG11b]. Poisoning is not possible, as the adversary

cannot provide valid signatures and the DHT rejects malformed key-value pairs.

Concerning the delegation mechanism, the attacker has the option of tricking a user into accepting rogue mappings from his own zones. This requires social engineering. We assume that users of an anti-censorship system will be motivated to carefully check whose mappings they trust. Nevertheless, if the attacker succeeds, some damage will be done: all users that use this mapping will be affected. The effect thus depends on the "centrality" of the tricked user in the GNS graph. It is difficult to give estimates here, as the system is not deployed yet. In order to maximize the effects of his attack, the attacker would have to carry out his social engineering many times, which is naturally harder. Comparing this to DNSSEC, we note that even when a compromise has been detected, DNS users cannot choose whose delegations to follow. In GNS, they can attempt to find paths in the GNS graph via other contacts. The system that is most similar and in deployment is the OpenPGP Web-of-Trust. Ulrich et al. found that the Web-of-Trust has developed a strong mesh structure with many alternative paths [UHHC11]. If GNS develops a similar structure, users would greatly benefit.

Finally, censorship does not stop with the name system, and for a complete solution we thus need to consider censorship at lower layers. For example, an adversary might block the IP address of the server hosting the critical information. GNS is not intended as an answer to this kind of censorship. Instead, we advocate using tools like Tor [DMS04b] to circumvent the blockade.

6.8 Special Features

This section describes some additional special features in GNS that are used to deal with corner cases that a practical system needs to deal with, but that might only be relevant for a subset of the users.

6.8.1 Automatic Shortening

Once Dave's client translates "www.carol.+" to "www.carol.buddy.gnu", Dave can resolve "carol.buddy.gnu" to Alice's public key and then lookup the IP address for Alice's server under the respective key in the DHT. At this point, Dave's GNS system will also learn that Alice has set her "NICK" record to "carol". It will then check if the name "carol" is already taken in Dave's zone, and—if "carol" is free—offer Dave the opportunity to introduce a PKEY record into Dave's zone that would *shorten* "carol.buddy.gnu" to "carol.gnu".

Alternatively, the record could be automatically added to a special *shorten zone* that is, in addition to the master zone, under Dave's control. In this case, Alice would become available to Dave under "carol.shorten.gnu", thus highlighting that the name was created by automatic shortening within the domain name.

In either case, shortening eliminates Bob from the trust path for Dave's future interactions with Alice. Shortening is a variation of trust on first use

(TOFU), as compromising Bob afterwards would no longer compromise Dave's path to Alice.

6.8.2 Relative Names in Record Values

GNS slightly modifies the rules for some existing record types in DNS. In particular, names in DNS values are always absolute; GNS allows the notation ".+" to indicate that a name is relative. For example, consider "CNAME" records in DNS, which map an alias (label) to a canonical name: as specified in RFC 1035 [Moc87], the query can (and in GNS will) be restarted using the specified "canonical name". The difference between DNS and GNS is that in GNS, the canonical name can be a relative name (ending in ".+"), an absolute GNS name (ending in ".zkey") or a DNS name.

As with DNS, if there is a "CNAME" record for a label, no other records are allowed to exist for the same label in that zone. Relative names using the ".+" notation are not only legal in "CNAME" records, but in all records that can include names. This specifically includes "MX" and "SOA" records.

6.8.3 Dealing with Legacy Assumptions: Virtual Hosting and TLS

In order to integrate smoothly with DNS, GNS needs to accommodate some assumptions that current protocols make. We can address most of these with the "LEHO" resource record. In the following, we show how to do this for Web hosting. There are two common practices to address here; one is virtual hosting (i. e. hosting multiple domains on the same IP address); the other is the practice of identifying TLS peers by their domain name when using X.509 certificates.

The problem we encounter is that GNS gives additional and varying names to an existing service. This breaks a fundamental assumption of these protocols, namely that they are only used with globally unique names. For example, a virtually hosted website may expect to see the HTTP header `Host: www.example.com`, and the HTTP server will fail to return the correct site if the browser sends `Host: www.example.gnu` instead. Similarly, the browser will expect the TLS certificate to contain the requested "www.example.gnu" domain name and reject a certificate for "www.example.com", as the domain name does not match the browser's expectations.

In GNS, each user is free to pick his own petname for the service. Hence, these problems cannot be solved by adding an additional alias to the HTTP server configuration or the TLS certificate. Our solution for this problem is to add the *le*gacy *ho*stname record type ("LEHO") for the name. This record type specifies that "www.example.gnu" is known in DNS as "www.example.com". A proxy between the browser and the web server (or a GNS-enabled browser) can then use the name from this record in the HTTP `Host:` header. Naturally, this is only a legacy issue, as a new HTTP header with a label and a zone key could also be introduced to address the virtual hosting problem. The LEHO records can also be used for TLS validation by relating GNS names to globally unique

DNS names that are supported by the traditional X.509 PKI. Furthermore, GNS also supports TLSA records, and thus using TLSA records instead of CAs would be a better alternative once browsers support it.

6.8.4 Handling TLSA and SRV records

TLSA records are of particular interest for GNS, as they allow TLS applications to use DNSSEC as an alternative to the X.509 CA PKI. With TLSA support in GNS, GNS provides an alternative to X.509 CAs and DNSSEC using this established standard. Furthermore, GNS does not suffer from the lack of end-to-end verification that currently plagues DNSSEC.

However, to support TLSA in GNS a peculiar hurdle needs to be resolved. In DNS, both TLSA and SRV records are special in that their domain names are used to encode the service and protocol to which the record applies. For example, a TLSA record for HTTPS (port 443) on `www.example.com` would be stored under the domain name `_443._tcp.www.example.com`.

In GNS, this would be a problem since dots in GNS domain names are supposed to always correspond to delegations to another zone. Furthermore, even if a special rule were applied for labels starting with underscores, this would mean that, say the A record, for `www.example.com` would be stored under a different key in the DHT than the corresponding TLSA record. As a result, an application would experience an unpredictable delay between receiving the A record and the TLSA record. As a TLSA record is not guaranteed to exist, this would make it difficult for the application to decide between delaying in hope of using a TLSA record (which may not exist) and using traditional X.509 CAs for authentication (which may not be desired and is likely less secure).

GNS solves this problem by introducing another record type, the BOX record. A BOX record contains a 16-bit port, a 16-bit protocol identifier, a 32-bit embedded record type (so far always SRV or TLSA [Bar11]) and the embedded record value. This way, BOX records can be stored directly under `www.example.com` and the corresponding SRV or TLSA values are thus never delayed — not to mention the number of DHT lookups is reduced. When GNS is asked to return SRV or TLSA records via DNS, GNS recognizes the special domain name structure, resolves the BOX record and automatically unboxes the BOX record during the resolution process. Thus, in combination with the user interface (Figure 6.3) GNS effectively hides the existence of BOX records from DNS users.

We note that DNS avoids the problem of indefinite latency by being able to return NXDOMAIN in case a SRV or TLSA record does not exist. However, in GNS NXDOMAIN is not possible, largely due to GNS's provisions for query privacy. Furthermore, DNS can solve the efficiency problem of a second lookup by using its "additional records" feature in the reply. Here, a DNS server can return additional records that it believes may be useful but that were not explicitly requested. However, returning such additional records might not always work, as DNS implementations can encounter problems with the serious size restrictions (often just 512 bytes) on DNS packets. As GNS replies can contain

up to 63 kB of payload data, we do not anticipate problems with the size limit in GNS even for a relatively large number of unusually big TLSA records.

6.8.5 Revocation

In case a zone's private key gets lost or compromised, it is important that the key can be revoked. Whenever a user decides to revoke a zone key, other users must be notified about the revocation. However, we cannot expect users to explicitly query to check if a key has been revoked, as this increases their latency (especially as reliably locating revocations may require a large timeout) and bandwidth consumption for every zone access — just to guard against the relatively rare event of a revoked key. Furthermore, issuing a query for zone revocations would create the privacy issue of revealing that a user is interested in a particular zone. Existing methods for revocation checks using certificate

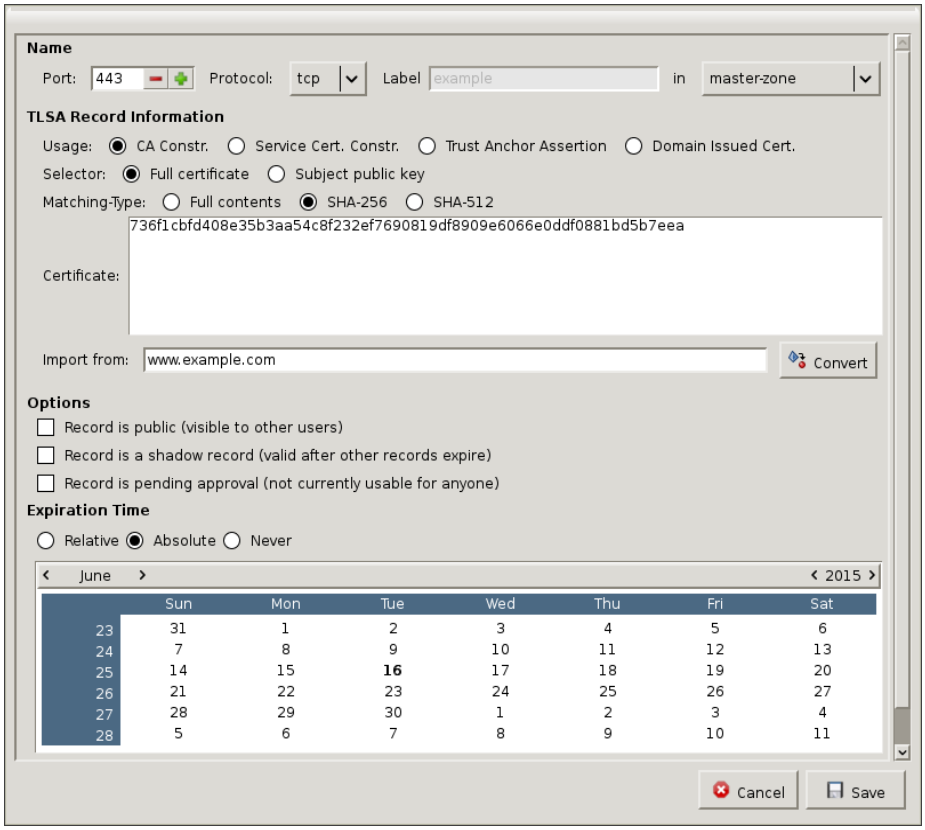


Figure 6.3: The user can remain unaware of the behind-the-scenes boxing when creating TLSA records in the GNS zone management interface.

revocation lists in X.509 have similar disadvantages in terms of bandwidth, latency increase and privacy.

Instead of these traditional methods, GNS takes advantage of the P2P overlay below the DHT to distribute revocation information by flooding the network. When a peer wants to publish a revocation notice, it simply forwards it to all neighbors; all peers do the same when they receive previously unknown valid revocation notices. However, this simple Byzantine fault-tolerant algorithm for flooding in the P2P overlay could be used for denial of service attacks. Thus, to ensure that peers cannot abuse this mechanism, GNS requires that revocations include a revocation-specific proof of work. As revocations are expected to be rare special events, it is acceptable to require an expensive computation by the initiator. After that, all peers in the network will remember the revocation forever (revocations are a few bytes, thus there should not be an issue with storage).

In the case of peers joining the network or a fragmented overlay reconnecting, revocations need to be exchanged between the previously separated parts of the network to ensure that all peers have the complete revocation list. This can be done using bandwidth proportional to the difference in the revocation sets known to the respective peers using Eppstein's efficient set reconciliation method. In effect, the bandwidth consumption for healing network partitions or joining peers will then be almost the same as if the peers had always been part of the network.

This revocation mechanism is rather hard to disrupt for an adversary. The adversary would have to be able to block the flood traffic on all paths between the victim and the origin of the revocation. Thus, our revocation mechanism is not only decentralized and privacy-preserving, but also much more robust compared to standard practice in the X.509 PKI today, where blocking of access to certificate revocation lists is an easy way for an adversary to render revocations ineffective. This has forced vendors to include lists of revoked certificates with software updates.

6.8.6 Shadow Records

GNS records can be marked as "shadow records"; the receiver only interprets shadow records if all other records of the respective type have expired. This is useful to ensure that upon the timeout of one set of records the next set of records is immediately available. This may be important, as propagation delays in the DHT are expected to be larger than those in the DNS hierarchy.

6.8.7 Availability and Caching

By default, each authority pushes all public records for a given label once every four hours; as the records are retrieved from the DHT, it is pointless to maintain several peers for a zone for load balancing. Nevertheless, it is of course possible to improve availability by operating more than one peer in the overlay and to replicate the functions of the zone authority over multiple peers. However, the

use of the DHT should make this unnecessary, which is important as we would not expect typical dissidents to necessarily have the capacity for redundancy.

To minimize the load on the network and to reduce latency, all validated records are cached until they expire. The local system also contains the primary database for all of the zones for which the peer is authoritative (Figure 6.4).

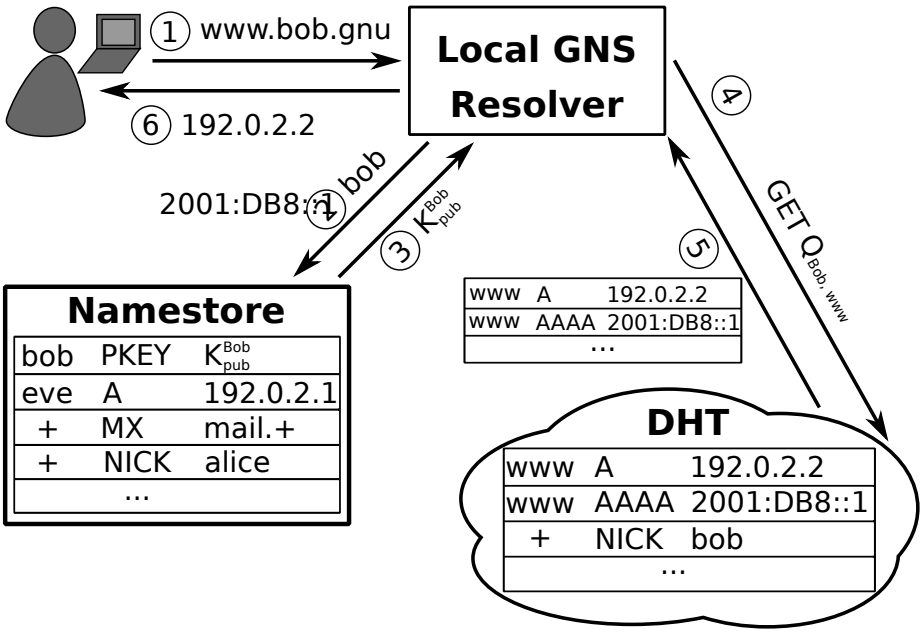


Figure 6.4: In the GNS architecture, most requests will be answered by the local namestore which is the authority for all names in the user's personal ".gnu" pseudo-TLD. The user can delegate subdomains to other users using "PKEY" records. If not cached, names from those subdomains are then resolved using a DHT. Note that the records in the DHT are encrypted and cryptographically signed (Section 6.6).

6.8.8 Intercepting DNS queries

A GNS implementation only needs to intercept all DNS queries for the ".gnu" (and ".zkey") pseudo-TLD and inject appropriate responses. All other TLDs are forwarded to the traditional DNS system. Our current implementation provides three alternative methods to do so:

- On GNU systems, a plugin for the name services switch (NSS) [Fre17] in GNU libc can be used to answer GNS queries before a DNS request is ever created. Mechanisms similar to NSS exist for other platforms; we also have an equivalent plugin working on Microsoft Windows.

- The resolver configuration (usually `/etc/resolv.conf`) can be changed to point to an IP address (i.e. 127.0.0.1) with a modified DNS resolver. We have implemented a DNS-to-GNS gateway which resolves names in the ".gnu" pseudo-TLD internally, and acts as a proxy for all other TLDs by passing those requests to an actual DNS server.

- The browser can be configured to use an HTTP SOCKS Proxy. This option offers additional advantages, and is discussed in more detail in Section 6.8.9.

The NSS-based approach has the key advantage that it allows GNS to learn the identity of the user that issued the query. As a result, we can fully personalize the GNS lookup on a per-user basis by maintaining a simple mapping between (local) user names and the respective zone keys. A potential disadvantage is that some applications may bypass the operating system and directly contact a DNS resolver. Here, the network-level approaches can provide an alternative. For browsers, the easiest method is to configure a SOCKS Proxy, which can provide additional capabilities beyond allowing GNS to intercept DNS queries (discussed in more detail in Section 6.8.9).

The DNS-to-GNS proxy is useful to allow legacy systems to access the GNS distributed database without installing GNS or changing their system configuration. To allow this, we have registered a domain name in DNS ("zkey.eu") where the DNS authority passes all requests on to GNS. Anyone controlling a name in DNS can use the DNS-to-GNS proxy to create such a gateway.

While this trick can help users access GNS information without installing GNS, it only offers security or censorship-resistance advantages if the proxy operator (and the network to the proxy operator) can be trusted.

6.8.9 A HTTP SOCKS Proxy for Legacy Browsers

Our current implementation uses a client-side proxy to do the expansion of relative names and SSL verification. A proxy implementation has the advantage that it works with virtually all browsers. However, compared to native support by browsers, using a proxy has the disadvantage that dynamic links, which might be generated by code executing within the browser, cannot be translated. Native support for GNS by browsers would improve security and usability.

The proxy from our current implementation speaks the SOCKS4a protocol, which allows the browser to also delegate resolution of domain names to the proxy. This is important as it allows the proxy to perform GNS resolution and obtain "LEHO"records. If the target server is accessed using a GNS name, the proxy replaces relative GNS names in the HTML; connections to systems using DNS names are simply proxied without processing the content.

Another issue the client proxy tackles is the *Same-Origin-Policy (SOP)* imposed by modern browsers. The SOP forbids scripts or cookies to access a different name in the domain namespace. For example, if you browse `www.example.gnu` then JavaScript code from `www.example.com` is forbidden to run. This can be an issue as the cookies and JavaScript code might use the legacy hostname (LEHO)

instead of the GNS name and would then be ignored in accordance with the SOP. To solve this issue, the proxy translates links pointing to the LEHO and modifies the domain names in cookies to satisfy the SOP. This is implemented using HTML rewriting and the use of Cross-Origin-Resource-Sharing [vK12].

6.9 Applications

While using GNS as a general alternative to DNS may seem appealing, there are some applications where GNS has particular appeal because of shortcomings in DNS. In particular, GNS is a natural fit when it comes to handling social relationships.

6.9.1 Key Exchange

A first simple application is to introduce "CERT" records to store OpenPGP public keys in GNS. Once users do this, GNS provides an alternative to the Web-of-Trust. However, unlike the Web-of-Trust, GNS provides query privacy. Furthermore, the identity model is different. In key signing parties for the Web-of-Trust, users are commonly expected to correctly perform a complicated multi-step process[3] to assure that government-issued identity cards match email addresses, thus linking OpenPGP keys to DNS information and real-world identities. This is problematic, as DNS and mail server operators can theoretically change the identity associated with an email address. Furthermore, the use of real-world identities makes the protocol fundamentally unsuitable for users that would like to be pseudonymous. Finally, the Web-of-Trust includes complex notions of what is required for a key to be verified by the trust graph, including different levels of trust that each user can specify for other users.

With GNS, the key exchange protocol is greatly simplified. After creating his key pair, Bob only needs to give his public key to Alice via an authenticated channel. Alice then assigns a label for the key (or confirms the suggested nickname) in her zone. Alice is not expected to check Bob's identity, as "bob.gnu" is *her* name for Bob. Furthermore, Bob's email is irrelevant, his identity is his public key and it is not tied to his email provider or DNS. Finally, Alice could email Bob at "*mailbox*@bob.gnu", assuming Bob configured an MX record for his zone and shared a mailbox name with her.

6.9.2 Telephony

A second simple application is to use GNS for P2P voice applications. Existing P2P voice applications, such as Skype, typically use a centralized service for user authentication. This is highly problematic as this is one place where attacks can be mounted against the system, from denying access to interception and impersonation. One alternative is the use of X.509 client certificates for users,

[3] See `http://www.keysigning.org/methods/sassaman-efficient` and `https://wiki.debian.org/Keysigning` for popular instructions.

which is, for example, supported by Mumble[4]. However, the use of certificate authorities (CAs) in X.509 allows a large number of CAs to act as trusted third parties, with the weakest CA determining the security of the system. Furthermore, the cost of certification or the desire to use pseudonyms drive users to use self-signed certificates, which provide no more than TOFU security.

We implemented a simple conversation service on top of GNUnet which uses GNS to establish a secure connection between the participants. GNS "PHONE" records contain the public key of a peer and an integer specifying the *line* under which a user application realizes a phone service for incoming calls. A *call* can then be made by specifying the GNS name that resolves to the "PHONE" record. The connection to the target peer is then secured using ECDHE and AES using the public key of the target peer. The caller signs the call request with his zone key. The callee performs a reverse lookup against the caller's public key to determine the *caller id*. If the caller's public key is not in the callee's zone, a ".zkey" name is generated from the public key instead.

6.9.3 Other Applications

Other applications that would be a good fit for GNS include naming for Tor hidden services (memorable names for ".onion"), and identity management in fully decentralized P2P social networking applications [Tot13b].

6.10 Censorship in Other Layers

Censorship does not stop with the name system. For example, censors can also attempt to block information by destination IP address. Blocking IP addresses is actually easier than censoring DNS; however, there is an increased chance of collateral damage as with virtual hosting, a single IP address can host many sites and services. Tools that help users circumvent IP-level censorship can also benefit from censorship-resistant name systems.

For example, the Tor network [DMS04b] is an anonymizing public virtual network for tunneling TCP connections over the P2P overlay network. While Tor is often associated with the goal of providing anonymity for HTTP clients, it can also be used to circumvent censorship by tunneling (the Tor overlay) traffic in other protocols, such as TLS. Tor also offers the possibility of hosting services within the Tor network, here with the primary goal of providing anonymity to the operators of the servers. Accessing these "hidden services" using cryptographic identifiers is not particularly user-friendly.

Given a decentralized censorship-resistant name system, it should be easy to provide names for services offered within such P2P overlays. The name system would map names to a new record type that identifies the respective service and peer (instead of using "A" or "AAAA" records to reference a host on the Internet). Such service endpoint addresses can then again be translated to IP addresses in the entry node's private address range to enable communication

[4] `http://mumble.sourceforge.net/`

of legacy applications with the P2P service. The result would be still close to hidden services in Tor, though it would not necessarily have to also provide support for anonymity.

6.11 Related Work

Timeline-based systems in the style of Bitcoin [Nak08] have been proposed to create a global, secure and memorable name system [Swa11]. Here, the idea is to create a single, globally accessible timeline of name registrations that is append-only. In the Namecoin system [dot13], a user needs to expend computational power on finding (partial) hash collisions in order to be able to append a new mapping. This is supposed to make it computationally infeasible to produce an alternative valid timeline. It also limits the rate of registrations. However, the Namecoin system is not strong enough in our adversary model, as the attacker has more computational power than all other participants, which allows him to create alternative valid timelines. Note that our adversary model is not a far-fetched assumption in this context: it is conceivable that a nation-state can muster more resources than the small number of other entities that participate in the system, especially for systems used as an alternative in places where censorship is encountered or during the bootstrapping of the network, when only a small number of users participate.

The first practical system that improves confidentiality with respect to DNS queries and responses was DNSCurve [Ber08]. In DNSCurve, session keys are exchanged using Curve25519 [Ber06] and then used to provide authentication and encryption between caches and servers. DNSCurve improves the existing Domain Name System with confidentiality and integrity, but the fundamental issues of DNS with respect to the adversary trying to modify DNS mapping is not within its focus.

GNS has much in common with the name system in the Unmanaged Internet Architecture (UIA) [For08], as both systems are inspired by SDSI. In UIA, users can define personal names bound to self-certifying cryptographic identities and can access namespaces of other users. UIA's focus is on universal connectivity between a user's many devices. With respect to naming, UIA takes a clean-slate approach and simply assumes that UIA applications use the UIA client library to contact the UIA name daemon and thus understand the implications of relative names. In contrast, GNS was designed to interoperate with DNS as much as possible, and we have specifically considered what is needed to make it work as much as possible with the existing Internet. In terms of censorship-resistance, both systems inherit basic security properties from SDSI with respect to correctness.

6.12 Summary and Conclusion

We have outlined the limitations of censorship-resistant name systems and shown that it is not possible to achieve memorable, secure and global names in a unified name system. However, it is possible to use pseudo-TLDs to allow users to cherry-pick between multiple name systems, offering combinations of two of the three desirable properties. Among the theoretical ideas, the SDSI-design using delegation is the only which has so far not been attempted in practice. Here, the lack of globally unique names creates additional issues for legacy applications that need to be mitigated. Focusing on Web applications, we have performed a survey which shows that a delegation-based name system would offer significant benefits over simpler petname systems, as most name resolutions in practice arise from users following links. As each design offers unique advantages, developers of censorship circumvention tools should consider the integration or interoperability of their systems with *multiple* secure name systems via pseudo-TLDs, including DNS/DNSSEC, cryptographic identifiers and petnames with delegation.

Based on this analysis, we have introduced, GNS, a censorship-resistant, privacy-enhancing name system which avoids the use of trusted third parties. GNS provides names that are memorable, secure and transitive. Placing names in the context of each individual user eliminates ownership and effectively eliminates the possibility of executive or judicial control over these names.

GNS can be operated alongside DNS and begins to offer its advantages as soon as two parties using the system interact, enabling users to choose GNS or DNS based on their personal trade-off between censorship-resistance and convenience. We have implemented proxies that can be used in lieu of native support by browsers.

Part III

Outlook

Chapter 7

Ongoing and future work

This chapter summarizes the current direction of the GNUnet project and includes work that has either not been published or not been implemented or both.

7.1 Secure multiparty computations

Efficient secure multiparty computations form important building blocks for future privacy-preserving Internet applications. The GNUnet framework will thus incorporate common primitives for secure multiparty computations, such that applications can perform cooperative computations on private data.

7.1.1 Private scalar product

This section is based on unpublished joint work with Christian Fuchs and Tanja Lange.

We have implemented a privacy-preserving scalar product protocol in GNUnet, which is based on an original idea of Ioannidis et al. [IGA02] and was refined by Amirbekyan et al. [AEC07]. In the original protocol, Alice learns the scalar product $\sum a_i b_i$ of her private vector $\vec{a}$ with Bob's private vector $\vec{b}$. The protocol is privacy-preserving in that Alice cannot discern details about $\vec{b}$ other than what she can learn from $\vec{a}$ and the scalar product $\sum a_i b_i$, and Bob does not learn anything.

In recommender systems, Alice's vector is often sparse. Thus, performance can be improved by allowing Alice to select a subset of the elements of the vectors to be multiplied. Naturally, depending on how Alice determines this mask, Bob may be able to deduce that certain elements in a_i are zero.

We present two variants of the protocol, one based on homomorphic encryption, and one based on solving DLOG over Elliptic curves. The first variant is suitable for scalar products where the final value is large, while the second is suitable for scalar products where the final value is small.

Choice of Homomorphic Cipher

Our protocol uses the Paillier cryptosystem [PP99]:

$$\mathrm{E}_K(m) := g^m \cdot r^n \mod n^2, \tag{7.1}$$

$$\mathrm{D}_K(c) := \frac{(c^\lambda \mod n^2) - 1}{n} \cdot \mu \mod n \tag{7.2}$$

where the public key $K = (n, g)$, m is the plaintext, c the ciphertext, n the product of $p, q \in \mathbb{P}$ of equal length, and $g \in \mathbb{Z}^*_{n^2}$. In Paillier, the private key is (λ, μ), which is computed from p and q as follows:

$$\lambda := \mathrm{lcm}(p-1, q-1), \tag{7.3}$$

$$\mu := \left(\frac{(g^\lambda \mod n^2) - 1}{n} \right)^{-1} \mod n. \tag{7.4}$$

Paillier was chosen as it offers additive homomorphic public-key encryption, that is

$$\mathrm{E}_K(a) \otimes \mathrm{E}_K(b) \equiv \mathrm{E}_K(a + b) \tag{7.5}$$

for some public key K.

As the Paillier homomorphism cannot efficiently handle negative numbers, an appropriate offset s needs to be added to all inputs to ensure that the calculations will only involve positive numbers. s has to be chosen to be larger than the smallest possible input, and small enough such that the sum of all concurrently used offsets and operands is always smaller than n. We assume that a sufficiently large value for s is chosen and known to all participants.

Protocol 1

Let $(\mathrm{E}_K, \mathrm{D}_K)$ represent encryption and decryption operators with a additive homomorphic public key algorithm using key K.

The following protocol is used to compute the scalar product over a subset of two vectors of length n:

1. Alice initiates the protocol by transmitting her public key A, a mask $M \subseteq \{1, \ldots, n\}$ specifying which subset of the vectors will be multiplied, and $\mathrm{E}_A(s + a_i)$ for $i \in M$ to Bob. As discussed before, s denotes a shared static offset to enable scalar product operations with negative inputs.

2. Bob creates two random permutations π and π' over the elements in M, and a random vector r_i for $i \in M$ and sends

$$R := \mathrm{E}_A(s + a_{\pi(i)}) \otimes \mathrm{E}_A(s - r_{\pi(i)} - b_{\pi(i)}) \tag{7.6}$$
$$= \mathrm{E}_A(2 \cdot s + a_{\pi(i)} - r_{\pi(i)} - b_{\pi(i)}), \tag{7.7}$$
$$R' := \mathrm{E}_A(s + a_{\pi'(i)}) \otimes \mathrm{E}_A(s - r_{\pi'(i)}) \tag{7.8}$$
$$= \mathrm{E}_A(2 \cdot s + a_{\pi'(i)} - r_{\pi'(i)}), \tag{7.9}$$
$$S := \mathrm{E}_A(\sum (r_i + b_i)^2), \tag{7.10}$$
$$S' := \mathrm{E}_A(\sum r_i^2) \tag{7.11}$$

3. Alice decrypts R and R' and computes for $i \in M$:

$$a_{\pi(i)} - b_{\pi(i)} - r_{\pi(i)} = \mathrm{D}_A(R) - 2 \cdot s, \tag{7.12}$$
$$a_{\pi'(i)} - r_{\pi'(i)} = \mathrm{D}_A(R') - 2 \cdot s, \tag{7.13}$$

which is used to calculate

$$T := \sum_{i \in M} a_i^2 \tag{7.14}$$
$$U := -\sum_{i \in M} (a_{\pi(i)} - b_{\pi(i)} - r_{\pi(i)})^2 \tag{7.15}$$
$$U' := -\sum_{i \in M} (a_{\pi'(i)} - r_{\pi'(i)})^2 \tag{7.16}$$

and then computes

$$P := \mathrm{D}_A(S) + T + U \tag{7.17}$$
$$= \mathrm{D}_A\left(\mathrm{E}_A\left(\sum_{i \in M} (b_i + r_i)^2\right)\right) + \sum_{i \in M} a_i^2 + \left(-\sum_{i \in M} (a_i - b_i - r_i)^2\right) \tag{7.18}$$
$$= \sum_{i \in M} \left((b_i + r_i)^2 + a_i^2 - (a_i - b_i - r_i)^2\right) \tag{7.19}$$
$$= 2 \cdot \sum_{i \in M} a_i(b_i + r_i). \tag{7.20}$$
$$P' := \mathrm{D}_A(S') + T + U' \tag{7.21}$$
$$= \mathrm{D}_A\left(\mathrm{E}_A\left(\sum_{i \in M} r_i^2\right)\right) + \sum_{i \in M} a_i^2 + \left(-\sum_{i \in M} (a_i - r_i)^2\right) \tag{7.22}$$
$$= \sum_{i \in M} \left(r_i^2 + a_i^2 - (a_i - r_i)^2\right) \tag{7.23}$$
$$= 2 \cdot \sum_{i \in M} a_i r_i. \tag{7.24}$$

Finally, Alice computes the scalar product using:

$$\frac{P - P'}{2} = \sum_{i \in M} a_i(b_i + r_i) - \sum_{i \in M} a_i r_i = \sum_{i \in M} a_i b_i. \tag{7.25}$$

Protocol 2

The protocol uses a group $\langle g \rangle$ of some large but finite order o. Our implementation uses Curve25519 [Ber06] (but we will stick to multiplicative notation here).

Alice's public key is $A = g^a$, her private key is a. Alices gives to Bob $(g_i, h_i) = (g^{r_i}, g^{r_i a + a_i})$ using random values r_i for $i \in M$. These are ElGamal encryptions of g^{a_i}. Since the r_i are big they hide the a_i and a.

Bob can compute $(g_i^{b_i}, h_i^{b_i})$ for $i \in M$. Note that

$$(g_i^{b_i}, h_i^{b_i}) = (g^{r_i b_i}, g^{a r_i b_i + a_i b_i}) \tag{7.26}$$

is an ElGamal encryption of $g^{a_i b_i}$ with randomness $r_i b_i$ for Alice's key. Since Alice is not supposed to learn the individual values $a_i b_i$ but only their sum, Bob computes the products

$$\left(\prod_{i \in M} g_i^{b_i}, \prod_{i \in M} h_i^{b_i} \right)$$

(since he does not need the individual encryptions this is faster done by using a multi-exponentiation like Bos-Coster's [BC90]). This equals

$$\left(\prod_{i \in M} g_i^{b_i}, (\prod_{i \in M} g_i^{b_i})^a g^{\sum_{i \in M} a_i b_i} \right)$$

i.e. the ElGamal encryption of $g^{\sum_{i \in M} a_i b_i}$ under randomness $\sum_{i \in M} r_i b_i$ and for Alice's public key.

When Bob sends this result to Alice, she can then compute

$$\left(\prod_{i \in M} g_i^{b_i} \right)^{-a} \cdot \left(\prod_{i \in M} g_i^{b_i} \right)^{a} \cdot g^{\sum_{i \in M} a_i b_i} = g^{\sum_{i \in M} a_i b_i}.$$

Assuming $\sum_{i \in M} a_i b_i$ is sufficiently small, Alice can then obtain this value by solving the DLP.

The issue here is that the a_i and b_i must be small enough that computing DLPs makes sense — a DLP in an interval of size l costs roughly $\sqrt{l}$ group operations [Pol00].

7.1.2 Private set intersection cardinality with signatures

In [GRBG16] we proposed a protocol for private set intersection cardinality with signatures. Here, the set members are public keys, and by the signatures the

public keys on one side are used to confirm that the respective public key belongs into the set. This protocol then allows two participants to securely compute the set intersection cardinality of two sets, where one of the participants has his set signed by the members in the set. This secure multiparty computation does not follow the usual "honest but curious" adversary model; instead, if say Alice learns the result, a malicious computational partner Bob may attempt to make the set intersection look larger than it actually is. The signatures are used to prevent Bob from stuffing his set with false entries.

We envision using the protocol for privacy-preserving detection of abusive behavior in OSNs, where a large overlap between Alice's subscriptions and Bob's subscribers is an indicator of the benign nature of Bob.

We also envision using the protocol to assess the validity of public keys for an entity in a privacy-enhanced variant of the Web-of-trust. Here the users still sign each other's keys, but do not expose the resulting social graph to the world. We call this variant the Fog-of-trust protocol to indicate that the relationships between users remain obscured (albeit not perfectly, as the size of the intersection which is revealed naturally reveals some information about the sets).

We will now briefly sketch the new protocol (which is yet to be implemented).

The Boneh-Lynn-Shacham (BLS) signature scheme

We first outline the BLS signature scheme [BLS01], which begins with a gap co-Diffie-Hellman group pair (G_1, G_2) of order p with an efficiently-computable bilinear map $e\colon G_1 \times G_2 \to G_T$, a generator g_2 of G_2, and a cryptographic hash function $H_1 : \{0,1\}^* \to G_1$.

In the BLS scheme, a private key consists of a scalar $c \in \mathbb{Z}/p\mathbb{Z}$, while the corresponding public key is $C := g_2^c$, and a signature on a message m by C is $\sigma := H_1(m)^c$.

A signature σ is verified by checking that $e(H(m), C) = e(\sigma, g_2)$. If $\sigma = H(m)^c$ then this holds by bilinearity of e.

Our protocol

Given a cryptographic hash function h (which may or may not be identical to H), we define $Z' := \{h(x) | x \in Z\}$ whenever Z is some set under discussion, and assume a fixed system security parameter $\kappa \geq 1$ has been agreed upon. Each participant is identified by a public key $C = g_2^c$ for the BLS signature scheme. Each participant A has a subscriber list L_A consisting of tuples $(C, \sigma_{A,C})$ where $\sigma_{A,C} := H_1(A||\texttt{date})^c$ is a BLS signature affirming that $C = g_2^c$ was subscribed to A until some expiration `date`, the specifics of which depend on the application. We envision these signatures being provided in advance so that Bob's subscribers need not be online when he runs the protocol.

Suppose Alice wishes to know $n := |L_{\text{Alice}} \cap L_{\text{Bob}}|$. First, she generates an ephemeral private scalar $x_A \in \mathbb{Z}/p\mathbb{Z}$ and sends Bob

$$\mathcal{X}_{\text{Alice}} := \texttt{sort}\,[\, C^{x_A} \mid (C, \sigma_{A,C}) \in L_{\text{Alice}}\,]\,. \tag{7.27}$$

Second, Bob picks ephemeral private scalars $t_{\text{Bob},j} \in \mathbb{Z}/p\mathbb{Z}$ for $j \in 1, \ldots, \kappa$ and computes

$$\mathcal{X}_{\text{Bob},j} := \texttt{sort}\left[C^{t_{\text{Bob},j}} \;\middle|\; (C, \sigma_{B,C}) \in L_{\text{Bob}} \right] \tag{7.28}$$

$$\mathcal{Y}_{\text{Bob},j} := \texttt{sort}\left[\overline{C}^{t_{\text{Bob},j}} \;\middle|\; \overline{C} \in \mathcal{X}_{\text{Alice}} \right]. \tag{7.29}$$

For $j \in 1, \ldots, \kappa$ and $(C, \cdot) \in L_{\text{Bob}}$, Bob picks more ephemeral private scalars $s_{j,C} \in \mathbb{Z}/p\mathbb{Z}$ and computes $S_{j,C} := g_2^{s_{j,C}}$ and $\sigma_{B,S_{j,C}} := H_1(B||\texttt{date})^{s_{j,C}}$. Let π denote the permutation applied by the `sort` for $\mathcal{X}_{\text{Bob},j}$. Bob also computes

$$\mathcal{U}_{\text{Bob},j} := \pi\left[C^{t_{\text{Bob},j}} S_{j,C} \;\middle|\; (C, \sigma_{B,C}) \in L_{\text{Bob}} \right] \tag{7.30}$$

$$\mathcal{V}_{\text{Bob},j} := \pi\left[\sigma_{B,C}^{t_{\text{Bob},j}} \sigma_{B,S_C} \;\middle|\; (C, \sigma_{B,C}) \in L_{\text{Bob}} \right]. \tag{7.31}$$

He then sends to Alice the commitments $\mathcal{Y}'_{\text{Bob},i}$, and $\mathcal{V}'_{\text{Bob},i}$ for $i \in 1, \ldots, \kappa$, along with $\mathcal{U}_{\text{Bob},i}$ itself.

Third, Alice picks a non-empty random $J \subseteq \{1, \ldots, \kappa\}$ and sends J to Bob.

Fourth, Bob sends Alice all the scalars $t_{\text{Bob},j}$ and $\mathcal{V}_{\text{Bob},j}$ for $j \notin J$, as well as $\mathcal{X}_{\text{Bob},j}$ and all his scalars $\pi[s_{j,C} | (C, \sigma_{B,C}) \in L_{\text{Bob}}]$ for $j \in J$.

Fifth, Alice verifies Bob's commitments : For $j \notin J$, Alice checks that $t_{\text{Bob},j}$ matches the commitment $\mathcal{Y}'_{\text{Bob},j}$ by computing $\mathcal{Y}_{\text{Bob},j}$ herself. Also for $j \notin J$, Alice checks that $\mathcal{V}_{\text{Bob},j}$ matches the $\mathcal{V}'_{\text{Bob},j}$ and verifies the signatures in $\mathcal{V}_{\text{Bob},j}$ using $\mathcal{U}_{\text{Bob},j}$ as public keys. These signatures validate because we employ the BLS pairing-based signature scheme where:

$$\begin{aligned}
e(H_1(B||\texttt{date}), C^{t_{\text{Bob},j}} S_{j,C}) &= e(H_1(B||\texttt{date}), g_2^{t_{\text{Bob},j}+s_{j,C}}) \\
&= e(H_1(B||\texttt{date}), g_2)^{t_{\text{Bob},j}c+s_{j,C}} \\
&= e(H_1(B||\texttt{date})^{t_{\text{Bob},j}c+s_{j,C}}, g_2) \\
&= e(H_1(B||\texttt{date})^{t_{\text{Bob},j}c} H_1(B||\texttt{date})^{s_{j,C}}, g_2) \\
&= e(\sigma_{B,C}^{t_{\text{Bob},j}} \sigma_{B,S_C}, g_2)
\end{aligned}$$

For $j \in J$, Alice verifies $\mathcal{U}_{\text{Bob},j}$ by computing it herself from $\mathcal{X}_{\text{Bob},j}$ and the $s_{j,C}$.

For $j \in J$, Alice computes

$$\mathcal{Y}_{\text{Alice},j} := \left\{ \hat{C}^{x_A} \;\middle|\; \hat{C} \in \mathcal{X}_{\text{Bob},j} \right\}. \tag{7.32}$$

Finally, she obtains the result from $|\mathcal{Y}'_{\text{Alice},j} \cap \mathcal{Y}'_{\text{Bob},j}| = n$ for $j \in J$, checking that these values agree for all $j \in J$.

An attack on this blinded signature scheme translates into an attack on the underlying BLS signature scheme. If Bob tries to manipulate to increase the overlap, the cut-and-choose part detects this with probability $1 : 2^\kappa$.

7.2 Social networking

The NSA's PRISM program [NSA13] demonstrates that encrypting communication is insufficient and that society needs to transition from hosted messaging

applications like Google Mail or Facebook Messenger towards decentralized applications where data is end-to-end encrypted, and not in plaintext accessible on third-party systems financed by data mining for advertising. To move social networking applications away from service providers offering hosting, several key issues need to be addressed:

First, by removing the "trusted" service provider, we eliminate the simple authentication model where the service provider authenticates users via their user name and associated authentication data, such as passwords. Existing global public key infrastructures that are not provider-specific, like DNS and X.509, are also ill-suited for decentralized social networking applications, as they require payment and would reintroduce a service provider. However, the GNU Name System (GNS) is uniquely suited as a PKI for a decentralized social networking application, as one can easily model users as zones and social relationships as delegations in GNS. How visible these links between users are (private, shared via DHT, or even fully public) is a choice each user can make for each link.

Second, a modern Web-based service provider can more easily manage system-wide upgrades. If a new version of the service is to be rolled out, the provider updates the server-side software and database schemata, and then ships new software (typically as JavaScript) to the clients as they access the service via their browsers. Thus, the provider can be quite certain that virtually *all* clients run the latest version of the software. While clients loose all autonomy on their computing — features can be added or removed without their consent at any time — this dramatically simplifies upgrades for the provider. In contrast, in a decentralized P2P social networking application, one has to plan for clients running diverse implementations by diverse authors, and possibly versions that may be several years old. To enable improvements to the software under these conditions, the network protocol needs to be *extensible*. XML, JSON and various other formats offer *syntactic* extensibility. For the GNUnet social network, we will use a variant of PSYC[1] (so far called PSYC2) which offers both syntactic and *semantic* extensibility. Semantic extensibility is achieved by building object-oriented dispatching into the handling of messages, thereby allowing the introduction of more specialized message types while also permitting older clients to handle the messages using more general/generic baseline functions.

Third, PSYC2 is designed to operate over a stateful multicast communication infrastructure. Thus, we need to create a scalable and secure multicast mechanism to distribute the information. Here, a key design goal is to ensure that only eligible participants can read messages, while also permitting non-participants to provide bandwidth. In particular, as participants with asymmetric Internet connections will rarely be able to serve as effective bandwidth multipliers, we want to support peers that are not participating in the group to fan out encrypted messages. At the same time, global rekeying must be avoided when peers leave, as especially the origin is unlikely to be able to participate in frequent rekeying operations for larger group sizes. In terms of message dis-

[1] `http://about.psyc.eu/`

tribution, we plan to avoid both rigid designs like SplitStream [CDK+03] and high-latency designs based purely on gossip, and instead use a flexible forest construction with gossip operating in the background to achieve resilience.

Finally, to facilitate developers using this system, the social abstractions are packaged into an API [Tot13b] that offers a generic vocabulary for developing social networking applications. Key concepts in this API include:

nym pseudonym of another user in the network (≡ GNS zone)

place where social interactions happen (≡ multicast channel)

host owner of a place (≡ origin of multicast channel)

guest visitor of a place (≡ multicast subscriber)

Functions in the API then offer operations that allow a host or guests to `enter` or `leave` a place, for guests to `talk`, for the host to `announce` something. The host can also `admit`, `reject` or `eject` guests. To obtain context, guests can `replay` the (message) history of a place and `look` at the state of a place.

7.3 Payment and incentives

While Bitcoin [Nak08] offers a fully decentralized P2P payment system, we reject this solution for GNUnet on both ethical and technical grounds. Bitcoin is unethical as it prevents the state from effectively extracting taxes from economic activity, or from enforcing basic rules on allowed economic activity. If we want to live in a society where there is the possibility of the state providing basic safety (education, health care, justice) and can enforce the rule of law on economic activity, it would be unethical to deploy a system that fundamentally supports an anarcho-capitalistic and unregulated digital economy. Bitcoin is technically inadequate as it fails to offer good privacy for citizens, and is also too inefficient (high latency, high power consumption) to be viable for many applications.

The need for a design where the state can efficiently collect taxes precludes the use of a pure P2P architecture, as effective taxation requires that the state can obtain the information required for taxation from a few easily identified systems. With a P2P payment system, the state may have to audit any general-purpose computer!

Thus, despite our general love for decentralisation, Chaum's untraceable electronic cash [CFN90] offers the right design choice for a payment system: customers spending money can remain anonymous, while merchants receiving money are identifiable and thus taxable. Mint operators (who issue the digital coins) have the records which allow the government to track income.

A key issue with Chaum's design is that its use of blind signatures [Cha83] provides atomic coins: the coins are indivisible. While Brands [Bra93] introduces k-show signatures to achieve divisible coins, the resulting transactions are linkable, which reduces anonymity. With GNU Taler[2], we are introducing an

[2] `https://taler.net/`

improved Chaumian digital payment system where we can give change (virtually arbitrary divisibility), are computationally more efficient than Brands (but require on-line spending) and can even give refunds to anonymous customers — while at the same time retaining unlinkability and taxability of income. The key operation that enables this is Taler's *refresh* protocol, which enables a customer to exchange the residual value of a partially spent coin for new coins at the mint, providing unlinkable change. In Taler, coins are not merely signed digital tokens, but signed public (ECC) keys. This is necessary to identify the owner of a coin, allows coins to be used to sign digital contracts, and is crucial for the refresh protocol. Finally, Taler also explicitly introduces an auditor into the design, where the role of the auditor is to verify the correct operation of the mint.

With respect to GNUnet (and other P2P systems), Taler provides a trivial mechanism for providing incentives to contributing resources. Naturally, payments may not always be the ideal solution. Where applicable, alternative reward systems like those rewarding contributors with better throughput in file sharing applications [Gro03, Coh03] might still be considered. Still, an efficient and privacy-preserving payment mechanism will be a useful building block for many other applications we envision, in particular news distribution and secure privacy-preserving auctions [Tei17].

7.4 News distribution

Accessing news online is a key Internet activity for most users. For our discussion, news distribution includes browsing news websites (possibly using aggregators based on syndication [NS05]) as well as reading timelines in online social networks. News distribution is critical for a democratic society, as democracy is only meaningful if the voters are able to make informed choices. Naturally, using propaganda and deception has been part of the standard playbook for corporations and governments for a long time [Bel82], including historically the introduction of copyright for the purpose of censorship [Fal11].

P2P news distribution can principally provide both privacy and censorship-resistance, but we need to find ways to address other key aspects of the journalistic process:

- Sourcing: somebody still has to find interesting information.
- Contextualization and editing: somebody has to present the information, make it accessible and put it correctly into its context.
- Translation: text may need to be translated into the natural language and possibly cultural context for different groups of readers.
- Ranking: readers cannot possibly handle all of the information available today, so ranking algorithms are needed to provide an editorial filter to focus attention on the important news.

- Presentation: readers expect news to be presented in an accessible and appealing fashion.

The key challenge today for sourcing, editing and translation is that journalists working online sometimes find it difficult to be adequately paid for their work. With the Taler system, we hope to lower the barrier for individuals to setup for-pay online services, especially for news distribution as readers ought to be particularly sensitive about their privacy in this context.

For ranking, we plan to use collaborative ranking using cosine similarity computed using secure multiparty computation. We have extended the SMC dot product protocol of Ioannidis [IGA02] to handle signed values and include a set intersection operation to better handle extremely sparse vectors. We investigated a method combining a secure multiparty protocol for computing signed set intersection cardinality with supervised learning to classify messages as likely abusive based on private meta data [AGRBG16], which ought to be combined with the collaborative ranking algorithm in the future.

More work is needed to provide the actual distribution of news items and to facilitate collaborative editing and achieve adequate visual presentation.

7.5 Process architecture improvements

We envision various improvements to the existing testing, monitoring and deployment processes for GNUnet as the system moves closer to production.

7.5.1 Testing

Right now, developers typically write testcases to test the APIs that they introduce, and they sometimes use the code coverage analysis to guide their testing. Nevertheless, testing at this level makes it difficult to test paths where peers violate the protocol, as using the canonical APIs does not allow one to trigger malicious behavior. In some cases, this is addressed by adding configuration options or API calls that enable malicious versions of the core logic for testing. However, these malicious versions need to be hand-crafted and extending APIs or configurations to support them is laborious and thus not always justified.

Fuzzing provides an alternative method to test whether components handle malicious inputs. Especially guided fuzzers like American Fuzzy Lop[3] that try to automatically maximize code coverage can be useful to systematically synthesize interesting testcases, and in particular to ensure good coverage of branches performing error handling. Thus, one plan for the future is to fuzz each component by mutating the message streams it receives via IPC/RPC.

7.5.2 Monitoring

In terms of monitoring the deployed system, the existing statistics subsystem is limited in its view to a single peer. Using the SMC scalar product, it should

[3] `http://lcamtuf.coredump.cx/afl/`

be possible to have peers correlate statistics (while obscuring private data) to obtain a less insular view of network events, and possibly flag anomalies that are not sufficiently clear from the individual peer's data.

7.5.3 Deployment

Finally, we are working with the GNU Guix team to facilitate the installation of GNUnet using the Guix functional (and reproducible) build and package management system. Ideally, this would result in an easy to use, multi-distribution installation routine using Guix, as well as the ability of Guix users to upgrade Guix by downloading packages from other Guix peers, instead of FTP/HTTP mirrors.

7.6 Network architecture evolution

Currently, the GNUnet network architecture is totally egalitarian: all peers are treated as equal in all respects. However, this ideal rarely holds in practice. In reality, some peers will have high-speed network connectivity and be virtually always available and reachable. Other peers may use a slow DSL line and be off during the night. Finally, peers running on mobile phones may have to worry about battery drain and the cost of bandwidth may be excessively high. In the past, P2P networks like Gnutella and Tor have evolved to explicitly accommodate peers with different capabilities.

For GNUnet, we envision that peers — while in principle equal — may announce certain strong preferences to their neighbors, which those might than typically try to respect. In particular, we envision the network to include at least three classes of peers (Figure 7.1). (1) Infrastructure peers are peers with good network connectivity and high uptime. They should be preferred for DHT queries, and as relays. (2) Desktop peers are peers that are happy to contribute, but have sub-par bandwidth, latency or availability compared to infrastructure peers. The should be used if the infrastructure peers fail for any reason, and generally be included in DHT queries and relay operations as a backup option. (3) Mobile peers are unwilling to really spend resources for the needs of other users, as they are paying a very high price for their contributions. They may still be involved in operations as a last resort, but peers falling back to this option have to expect extremely low quality service.

The goal of this structure is purely performance optimization, but with the meta goal of never giving up control over user data or service availability to infrastructure providers. This key property is maintained as long as protocols only consider the service class of a peer as indicative for the performance to be expected, and not start to rely on any particular service class or peer for availability, confidentiality or integrity.

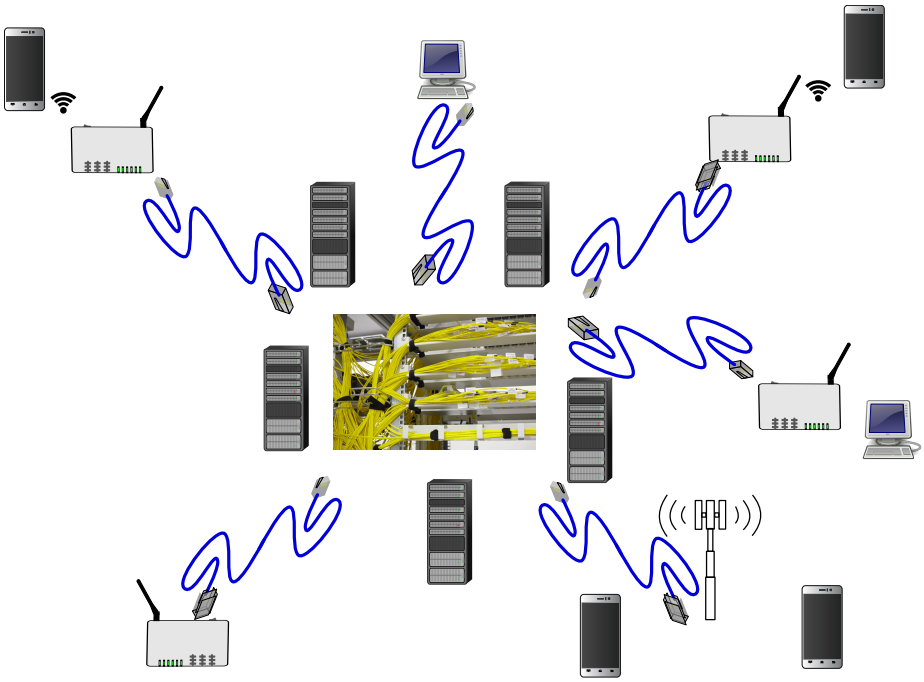


Figure 7.1: Network architecture with service classes. In the future, peers may voluntarily declare which level of service they are willing to provide, with expected levels including infrastructure (i.e. data-center-hosted), backup (i.e. desktops at home or office) or incapable (i.e. mobiles).

7.7 A new underlay

As an overlay network, GNUnet typically operates on top of the TCP/IP stack. However, doing so means that GNUnet is susceptible to the myriad of attacks against TCP/IP, from BGP route hijacking to TCP RST injection. While the impact of such attacks may largely be a loss of performance, we should nevertheless consider what the ideal underlay network for GNUnet would look like. As GNUnet treats TCP/IP more like a layer-1 technology (in the OSI model), this question basically is about the design of a new LAN protocol, a modern version of Ethernet or the 802.11 (WLAN) protocol family.

Figure 7.2 shows what an ideal packet format for a new physical underlay for GNUnet might look like. The packet begins with a public key, which identifies the `destination` of the packet. This is the only useful non-encrypted information. The public key is followed by a second ephemeral public key chosen by the sender. We note that this key does **not** identify the sender, the key is ephemeral in the sense that the sender picks a fresh ephemeral key for each packet. This header is followed by fixed size payload which is symmetrically encrypted us-

32 byte destination $D = dG$ (ECC Point)
32 byte ephemeral key $S = sG$ (ECC Point)
$2^{16} - 128$ byte encrypted payload ($K = ECDHE(d, S)$)
64 byte HMAC

Figure 7.2: Ideal packet. Once packets look like this, routers have no choice but to be neutral.

ing ECDHE based on the key of the recipient and the ephemeral key of the sender. Finally, the packet concludes with an HMAC (encrypt-than-MAC), but any modern authenticated encryption scheme providing confidentiality and integrity will basically do at this point. The use of a fixed size payload not only simplifies handling in hardware, but also ensures that packet size does not leak information. Thus, with this format on the link-layer, spy programs like TEMPORA [MBH+13] would become useless.

The receiver would learn the actual identity of the sender after decrypting the payload. Here, the sender's public key would be used to sign the ephemeral key. To make such public key operations viable at reasonable speeds, encoding and decoding the payload should be done directly in hardware. Hosts could discover their neighborhood by sending service discovery packets to a broadcast address with a well-known private key, for example $0G$ or $1G$ (with private keys $d = 0$ or $d = 1$ respectively). This hardware design would essentially be a drop-in replacement for GNUnet's CORE layer, allowing the system to drop legacy TCP/IP support entirely, dramatically improve performance by eliminating many layers in the stack, and leaving only the more interesting network functions to software.

The proposed header has some similarities to MinimaLT [PZS+13], where there sender's public key is also not transmitted in the clear and where the receiver's public key is obtained via the name system prior to establishing a connection. However, MinimalLT runs over IP and UDP, and thus ultimately does still leak information about the source address. Furthermore, the cryptography in MinimaLT is connection-oriented, and thus an adversary can *at least* learn the number of bytes and the duration of a connection. With the proposed scheme, sessions are no longer visible on the network as each packet is encrypted with a new ephemeral key. However, this key advantage requires much more public key operations and thus hardware support.

Chapter 8

Related projects

There are several projects related to GNUnet with at least somewhat similar ambitions. We note that we only consider an activity a "related project" if it involves a larger team developing and releasing software over a sustained period of time. This chapter gives an overview of some of the more important projects in this area.

Most related projects fall into one of two main categories. There are *patch projects* that try to address one very specific concern with the current Internet architecture while keeping the rest intact, and then there are *sledgehammer projects* that rebuild a significant part of the Internet around their specific sledgehammer technology. An example for an unrelated patch project would be IPv6 [DH95], which focuses on IPv4 address space exhaustion, and tries to make modest changes to the rest of TCP/IP. An example for a sledgehammer are the various academic efforts on content-centric or information-centric networking, which try to rearchitect the Internet around the concept of querying for information or content, instead of connecting to a particular server or service.

GNUnet is different from patch and sledgehammer projects in that it neither tries to solve just one particular issue nor applies just one key technique over and over again, and instead offers a toolchest to solve many issues. Nevertheless, all of these related projects are inspirational as a source of tools, or as a means to deploy solutions faster.

8.1 Patch projects

A number of important patch projects have been developed by the IETF. DNSSEC [KG06] attempts to fix authenticity of DNS records, and DANE [Bar11] uses DNSSEC to provide an alternative to the X.509 [HFPS99] public key infrastructure. DNS privacy considerations [Bor15] seeks to reduce meta data leakage from the DNS protocol. TCP MD5 signatures [Lee03] improve authenticity for TCP, in particular protecting against RST injection attacks.

The Tor project [DMS04b] uses onion routing to avoid leaking IP addresses

when establishing TCP connections, in particular for surfing the Web.

SCION [BRSP15] provides an alternative to the border gateway routing protocol (BGP) [LR91] using cryptography to prevent a host of known attacks on BGP [Mur06] such as BGP route hijacking or sinkholes, and ensures fast adaptation to link failures.

8.2 Sledgehammer projects

The Freenet project [CSBH01] has built a surprisingly diverse number of services on top of an anonymous file sharing layer. The I2P project [jP03] has done the same using an uni-directional [Her11, HG11] onion routing layer as the key abstraction. Finally, RetroShare[1] uses end-to-end encrypted friend-to-friend communication (combining TLS and PGP) as its key primitive. All of the above networks use these functions to implement services like chat, asynchronous messaging, forums and file sharing.

8.3 Semi-related projects

There is a number of semi-related projects, such as Tribler (exploiting ideas from Bittorrent), Maidsafe (exploiting ideas from Bitcoin) and Ethereum (also exploiting ideas from Bitcoin), which use their sledgehammer to build decentralized networks, but without the necessary focus on privacy-enhancing technologies that would make them related to GNUnet.

8.4 net2o

Finally, Bernd Paysan's `net2o`[2] is an attempt to "reinvent the Internet" that like GNUnet is concerned with privacy and security and also provides a toolchest of solutions. `net2o` uses packet switching information carried in each packet for simple fast routing, which also limits information leakage from headers. It has a new scheme for flow control, that does not suffer from buffer bloat [GN11] and provides low latency. It integrates a Forth-like command interpreter with a stack machine into the network stack, offering a large number of commands.[3] While `net2o` offers a surprisingly large number of new and interesting ideas, it is a one-man project that seems to have so far not received any academic attention.

[1] `http://retroshare.sourceforge.net/`
[2] `https://net2o.de/`
[3] `https://fossil.net2o.de/net2o/doc/trunk/wiki/commands.md`

Chapter 9

Discussion & Conclusion

GNUnet provides a modular foundation for building secure, decentralized network applications. But like the Internet, the GNUnet will never be "finished": there is plenty of room for improvement for existing modules, and even more for additional features.

Some of the features on the roadmap will be critical to address key challenges faced by modern societies. However, making the complexity of the system more manageable for users and lowering the barrier to entry — in particular for the installation — will probably be more crucial for GNUnet to become widely used. Here, the GNU Guix distribution offers a glimmer of hope through the thicket created by the interdependency forest.

The existing development process is also somewhat immature: deployed projects tend to evolve a more disciplined change management and review process, for example by forcing multiple developers to sign-off each change, by adding the requirement that regression tests have to pass *before* a commit, instead of having them run afterwards.

Today, GNUnet provides a platform for P2P research and development with a broader set of capabilities and features than any other such system in existence. The architecture also makes it relatively easy to develop extensions in various languages; today, (minor) extensions to the C codebase exist that use Java, Python, Ruby, Rust or JavaScript. Given the diversity in languages known to students today, this should also facilitate the use of GNUnet as a foundation for projects in graduate-level P2P courses. The development of a P2P textbook around GNUnet would thus be the next logical long text to be written about the network at large. After all, where would the Internet be today without Comer's TCP/IP books[1] educating the first generation of Internet engineers and researchers?

[1] `https://www.cs.purdue.edu/homes/comer/netbooks.html`

Bibliography

[3rd99] D. Eastlake 3rd. Domain Name System Security Extensions. RFC 2535 (Proposed Standard), March 1999. Obsoleted by RFCs 4033, 4034, 4035, updated by RFCs 2931, 3007, 3008, 3090, 3226, 3445, 3597, 3655, 3658, 3755, 3757, 3845.

[AAL+05] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. *IETF RFC 4033*, Mar. 2005.

[ABK+92] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, Nathan Linial, and Steven Phillips. Biased random walks. In *STOC*, pages 1–9. ACM, 1992.

[ABMW05] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions. *ACM Trans. Internet Technol.*, 5:299–327, May 2005.

[AE07] Chen Avin and Gunes Ercal. On the cover time and mixing time of random geometric graphs. *Theor. Comput. Sci*, 2007.

[AEC07] Artak Amirbekyan and Vladimir Estivill-Castro. A new efficient privacy-preserving scalar product protocol. In *Proceedings of the sixth Australasian conference on Data mining and analytics - Volume 70*, pages 209–214. Australian Computer Society, Inc., 2007.

[AGRBG16] Álvaro García-Recuero, Jeffrey Burdges, and Christian Grothoff. Privacy-preserving abuse detection in future decentralised online social networks. In *Data Privacy Management (DPM)*, pages 78–93, 2016.

[AKC+00] I. Androutsopoulos, J. Koutsias, K. Chandrinos, G. Paliouras, and C. Spyropoulos. An evaluation of naive bayesian anti-spam filtering, 2000.

[Alt06] Bob Altemeyer. *The Authoritarians*. University of Manitoba, 2006.

[AM14] Jacob Appelbaum and Nick Mathewson. Pluggable Transport Specification. `https://gitweb.torproject.org/torspec.git/blob/HEAD:/pt-spec.txt`, January 2014. accessed: 2014-04-25.

[Ano12] Anonymous. The collateral damage of internet censorship by dns injection. *ACM SIGCOMM Comp. Comm. Review*, 42(3):22–27, July 2012.

[Bar11] R. Barnes. Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE). RFC 6394 (Informational), October 2011.

[BBK+10] Marin Bertier, François Bonnet, Anne-Marie Kermarrec, Vincent Leroy, Sathya Peri, and Michel Raynal. D2ht: The best of both worlds, integrating rps and dht. In *Proceedings of the 2010 European Dependable Computing Conference*, pages 135–144, Washington, DC, USA, 2010. IEEE Computer Society.

[BC90] Jurjen Bos and Matthijs Coster. *Addition Chain Heuristics*, pages 400–407. Springer New York, New York, NY, 1990.

[BDG14] Nikita Borisov, George Danezis, and Ian Goldberg. Dp5: A private presence service. Technical report, Centre for Applied Cryptographic Research (CACR), University of Waterloo, May 2014.

[BDL+12] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[Bel82] J. Bowyer Bell. *Cheating*. St Martin's Press, 1982.

[Ber06] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[Ber08] Daniel J. Bernstein. Dnscurve: Usable security for dns. `http://dnscurve.org/`, August 2008.

[BER11] BEREC. A view of traffic management and other practices resulting in restrictions to the open Internet in Europe. `http://ec.europa.eu/digital-agenda/sites/digital-agenda/files/Traffic%20Management%20Investigation%20BEREC_2.pdf`, January 2011. accessed: 2014-04-25.

[BG03] Krista Bennett and Christian Grothoff. gap - Practical Anonymous Networking. In *Designing Privacy Enhancing Technologies*, pages 141–160. Springer-Verlag, 2003.

[BGHP02] K. Bennett, C. Grothoff, T. Horozov, and I. Patrascu. Efficient Sharing of Encrypted Data. In *Proceedings of ASCIP 2002*, 2002.

[BGK+09] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks Journal (COMNET), Special Issue on Gossiping in Distributed Systems*, April 2009.

[BHMW08] R. Bless, C. Hübsch, S. Mies, and O. Waldhorst. The Underlay Abstraction in the Spontaneous Virtual Networks (SpoVNet) Architecture. In *Proc. 4th EuroNGI Conf. on Next Generation Internet Networks (NGI 2008)*, pages 115–122, April 2008.

[BJBS+08] Javier Bustos-Jimenez, Nicolas Bersano, Satu Elisa Schaeffer, Jose Miguel Piquer, Alexandru Iosup, and Augusto Ciuffoletti. Estimating the size of peer-to-peer networks using lambert's w function. In Sergei Gorlatch, Paraskevi Fragopoulou, and Thierry Priol, editors, *Grid Computing*, pages 61–72. Springer US, 2008.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '01, pages 514–532, London, UK, UK, 2001. Springer-Verlag.

[Bor15] S. Bortzmeyer. DNS Privacy Considerations. RFC 7626 (Informational), August 2015.

[Boy02] Gerald E. Boyd. Accessing the internet by e-mail. `http://www.faqs.org/faqs/internet-services/access-via-email/`, 2002.

[Bra93] Stefan A Brands. An efficient off-line electronic cash system based on the representation problem, 1993.

[BRSP15] David Barrera, Raphael M. Reischuk, Pawel Szalachowski, and Adrian Perrig. SCION Five Years Later: Revisiting Scalability, Control, and Isolation on Next-Generation Networks. *arXiv e-prints*, August 2015.

[BYGM03] B. Beverly Yang and H. Garcia-Molina. Designing a super-peer network. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 49–60, March 2003.

[Cav09] Ann Cavoukian. Privacy by design. Technical report, Information & Privacy Commissioner, Ontario, Canada, 2009.

[CCF10] Thibault Cholez, Isabelle Chrisment, and Olivier Festor. Efficient dht attack mitigation through peers' id distribution. In *HOTP2P'10 - International Workshop on Hot Topics in Peer-to-Peer Systems*, Atlanta, Georgia, USA, 04/2010 2010.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: high-bandwidth multicast in cooperative environments. *SIGOPS'03 Operating Systems Review*, 37:298–313, 10/2003 2003.

[CF06] Martin Casado and Michael J. Freedman. Illuminating the shadows: Opportunistic network and web measurment, December 2006.

[CFN90] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Proceedings on Advances in cryptology*, pages 319–327. Springer-Verlag New York, Inc., 1990.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, November 1998.

[Cla88] D. Clark. The design philosophy of the darpa internet protocols. *SIGCOMM Comput. Commun. Rev.*, 18(4):106–114, August 1988.

[Coh03] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[CSBH01] Ian Clarke, Oskar Sandberg, Wiley Brandon, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 46–66, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[DaR09] Matteo Dell amico and Yves Roudier. A measurement of mixing time in social networks. In *5th International Workshop on Security and Trust Management*, Saint Malo, France, September 2009.

[DG16] Florian Dold and Christian Grothoff. Byzantine set-union consensus using efficient set reconciliation. In *International Conference on Availability, Reliability and Security (ARES)*, 2016.

[DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[DH95] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 1883 (Proposed Standard), December 1995. Obsoleted by RFC 2460.

[DLlKA05] George Danezis, Chris Lesniewski-laas, M. Frans Kaashoek, and Ross Anderson. Sybil-resistant dht routing. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS)*, pages 305–318. Springer, 2005.

[DMS04a] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router, 2004.

[DMS04b] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[dot13] The Dot-BIT project, A decentralized, open DNS system based on the bitcoin technology. `http://dot-bit.org/`, April 2013.

[Dou02] John R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[DSKM12] Casey Deccio, Jeff Sedayao, Krishna Kant, and Prasant Mohapatra. Quantifying dns namespace influence. *Comput. Netw.*, 56(2):780–794, February 2012.

[Duk14] V. Dukhovni. Opportunistic Security: Some Protection Most of the Time. RFC 7435 (Informational), December 2014.

[EDG09] Nathan S. Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on tor using long paths. In *18th USENIX Security Symposium*, pages 33–50. USENIX, 2009.

[EF94] K. Egevang and P. Francis. Rfc 1631: The ip network address translator (nat), 1994.

[EG11a] Nathan Evans and Christian Grothoff. Beyond simulation: Large-scale distributed emulation of p2p protocols. In *4th Workshop on Cyber Security Experimentation and Test (CSET 2011)*. USENIX Association, 2011.

[EG11b] Nathan Evans and Christian Grothoff. R5n: Randomized recursive routing for restricted-route networks. In *5th International Conference on Network and System Security*, pages 316–321, Milan, Italy, 2011. IEEE.

[EG15] Yves Eudes and Christian Grothoff. Skynet, le programme ultra-secret de la nsa créé pour tuer. *Le Monde*, (20.10.2015):14, January 2015.

[EGG07] Nathan S. Evans, Chris GauthierDickey, and Christian Grothoff. Routing in the dark: Pitch black. In *23rd Annual Computer Security Applications Conference*, pages 305–314. IEEE Computer Society, 2007.

[EGH+03] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21:341–374, November 2003.

[EGUV11] David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. What's the difference?: efficient set reconciliation without prior context. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, page 218–229, New York, NY, USA, 2011. ACM, ACM.

[EPG12] Nathan S. Evans, Bart Polot, and Christian Grothoff. Efficient and secure decentralized network size estimation. In *11th International IFIP TC 6 Networking Conference*, volume 7289 of *LNCS*, pages 304–317. IFIP, Springer Verlag, 2012.

[ER59] P. Erdős and A. Rényi. On random graphs. I. *Publ. Math. Debrecen*, 6:290–297, 1959.

[Ess14] Loek Essers. German court finds domain registrar liable for torrent site's copyright infringement. `http://www.itworld.com/print/403869`, February 2014.

[Eur11] European Parliament. Resolution on the EU-US Summit of 28 November 2011, November 2011. P7-RC-2011-0577.

[Fal11] Rick Falkvinge. History of copyright. `https://falkvinge.net/2011/02/01/history-of-copyright-part-1-black-death/`, 2011.

[FB96] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045, IETF, November 1996.

[FBH+02] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing Web Censorship and Surveillance. In *Proceedings of the* 11^{th} *USENIX Security Symposium*, San Francisco, CA, 2002.

[FGR03] Ronaldo A. Ferreira, Christian Grothoff, and Paul Ruth. A Transport Layer Abstraction for Peer-to-Peer Networks. In *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (GRID 2003)*, pages 398–403. IEEE Computer Society, 2003.

[FHC03] P. Faltstrom, P. Hoffman, and A. Costello. Internationalizing Domain Names in Applications (IDNA). RFC 3490 (Proposed Standard), March 2003. Obsoleted by RFCs 5890, 5891.

[For08] Bryan Alexander Ford. *UIA: A Global Connectivity Architecture for Mobile Personal Devices*. PhD thesis, Massechusetts Institute of Technology, 2008.

[FRA+05] Ronaldo A. Ferreira, Murali Krishna Ramanathan, Asad Awan, Ananth Grama, and Suresh Jagannathan. Search with probabilistic guarantees in unstructured peer-to-peer networks. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 165–172, Washington, DC, USA, 2005. IEEE Computer Society.

[Fre17] Free Software Foundation. The GNU C Library - System Databases and Name Service Switch. `https://www.gnu.org/software/libc/manual/html_node/Name-Service-Switch.html`, 2017.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[GG08] Chris GauthierDickey and Christian Grothoff. Bootstrapping of peer-to-peer networks. In *Proceedings of DAS-P2P*, pages 205–208, Turku, Finland, August 2008. IEEE.

[GN11] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 9(11):40:40–40:54, November 2011.

[Gnu02] Gnutella. http://gnutella.wego.com. `http://gnutella.wego.com`, 2002.

[Gon01] Li Gong. Projext jxta: A technology overview. Technical Report 650 690-1330, Sun Microsystems, April 2001.

[GRBG16] Alvaro Garcia-Recuero, Jeffrey Burdges, and Christian Grothoff. Privacy-preserving abuse detection in future decentralised online social networks. In *Data Privacy Management (DPM)*, pages 78–93, 2016.

[Gro03] Christian Grothoff. An Excess-Based Economic Model for Resource Allocation in Peer-to-Peer Networks. *Wirtschaftsinformatik*, 3-2003, June 2003.

[Gro05] Christian Grothoff. Reading File Metadata with extract and libextractor. *LinuxJournal*, 6-2005:86–88, June 2005.

[GWE+15] Christian Grothoff, Matthias Wachs, Monika Ermert, Jacob Appelbaum, David Larousserie, Yves Eudes, and Laura Poitras. Morecowbells: Nouvelles révélations sur les pratiques de la nsa. *Le Monde*, (24.1.2015), January 2015.

[HC07] Jani Hautakorpi and Gonzalo Camarillo. Evaluation of dhts from the viewpoint of interpersonal communications. In *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*, pages 74–83. ACM, 2007.

[Her11] Michael Hermann. Privacy-implications of performance-based peer selection by onion-routers: A real-world case study using i2p. Master's thesis, TU-Munich, March 2011.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 2459 (Proposed Standard), January 1999. Obsoleted by RFC 3280.

[HG11] Michael Herrmann and Christian Grothoff. Privacy implications of performance-based peer selection by onion routers: A real-world case study using i2p. In *Proceedings of the 11th Privacy Enhancing Technologies Symposium (PETS 2011)*, July 2011.

[Hol14] Ralph Holz. *Empirical analysis of Public Key Infrastructures and investigation of improvements.* PhD thesis, TU Munich, 2014.

[HS12] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. *IETF RFC 6698*, Aug. 2012.

[HS13] Amir Herzberg and Haya Shulman. Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org. In *CNS 2013. The Conference on Communications and Network Security. IEEE.* IEEE, 2013.

[Hus10] Geoff Huston. Ipv4 address report. `http://www.potaroo.net/tools/ipv4/`, March 2010.

[HWF09] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 31–42, New York, NY, USA, 2009. ACM.

[HZH+15] Z. Hu, L. Zhu, J. Heideman, A. Mankin, D. Wessels, and P. Hoffman. Dns over tls: Initiation and performance considerations. `http://tools.ietf.org/html/draft-ietf-dprive-dns-over-tls-00`, Sept 2015.

[IGA02] Ioannis Ioannidis, Ananth Grama, and Mikhail Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *Proceedings of the 2002 International Conference on Parallel Processing*, pages 379–. IEEE Computer Society, 2002.

[Inc15] Ubuntu Inc. Apparmor. `http://apparmor.net/`, October 2015.

[IW01] Prakash Iyer and Ulhas Warrier. *InterngetGatewayDevice:1 Device Template Version 1.01*. UPnP Forum, `http://www.upnp.org/standardizeddcps/igd.asp`, November 2001.

[J. 08] J. Rosenberg and R. Mahy et. al. Session Traversal Utilities for NAT (STUN). RFC 5389, IETF, October 2008.

[JMB05] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23:219–252, August 2005.

[jP03] jrandom (Pseudonym). Invisible internet project (i2p) project overview. `https://geti2p.net/_static/pdf/i2p_philosophy.pdf`, August 2003.

[Kal00] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), September 2000.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[KG06] O. Kolkman and R. Gieben. DNSSEC Operational Practices. RFC 4641 (Informational), September 2006. Obsoleted by RFC 6781.

[KL70] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 49(1):291–307, 1970.

[Kle00] Jon M. Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, 2000.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research (JAIR)*, 4:237–285, 1996.

[Kor83] J. T. Korb. A Standard for the Transmission of IP Datagrams Over Public Data Networks. RFC 877, IETF, September 1983.

[KPG+05] Dionysios Kostoulas, Dimitrios Psaltoulis, Indranil Gupta, Ken Birman, and Al Demers. Decentralized schemes for size estimation in large and dynamic groups. In *Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*, pages 41–48, Washington, DC, USA, 2005. IEEE Computer Society.

[KSS08] Jon Kleinberg, Mark Sandler, and Aleksandrs Slivkins. Network failure detection and graph connectivity. *SIAM J. Comput.*, 38:1330–1346, August 2008.

[LCC+02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.

[Lee03] M. Leech. Key Management Considerations for the TCP MD5 Signature Option. RFC 3562 (Informational), July 2003.

[Lin02] Götz Lindenmaier. libfirm – a library for compiler optimization research implementing firm. Technical Report 2002-5, September 2002.

[LLK10] Chris Lesniewski-Laas and M. Frans Kaashoek. Whanau: A sybil-proof distributed hash table. In *NSDI*, pages 111–126. USENIX Association, 2010.

[LMSW10] Thomas Locher, David Mysicka, Stefan Schmid, and Roger Wattenhofer. Poisoning the Kad Network. In *11th International Conference on Distributed Computing and Networking (ICDCN), Kolkata, India*, volume 5935, pages 195–206. Springer, January 2010.

[LNOM08] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro*, 28(2):39–55, 2008.

[LPW06] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006.

[LR91] K. Lougheed and Y. Rekhter. Border Gateway Protocol 3 (BGP-3). RFC 1267 (Historic), October 1991.

[LSG+04] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, and M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proc. of the 3rd IPTPS*, 2004.

[Mal93] G. Malkin. RIP Version 2 Protocol Analysis. RFC 1387 (Informational), January 1993. Obsoleted by RFC 1721.

[MAM+99] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999. Obsoleted by RFC 6960, updated by RFC 6277.

[Mar06] Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351:394–406, February 2006.

[MBH+13] Even MacAskill, Julian Borger, Nick Hopkins, Nick Davies, and James Ball. Gchq taps fibre-optic cables for secret access to world's communications. `http://www.theguardian.com/uk/2013/jun/21/`

`gchq-cables-secret-world-communications-nsa`, June 2013.

[MBR03] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: distributed hashing in a small world. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association.

[MCB11] Prateek Mittal, Matthew Caesar, and Nikita Borisov. X-vine: Secure and pseudonymous routing using social networks. *Computer Research Repository*, abs/1109.0971, 9/2011 2011.

[MD05] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 183–195, Washington, DC, USA, May 2005. IEEE Computer Society.

[MEGK10] Andreas Müller, Nathan Evans, Christian Grothoff, and Samy Kamkar. Autonomous nat traversal. In *10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P 2010)*, pages 61–64. IEEE, 2010.

[MHJT14] Brad Miller, Ling Huang, A.D. Joseph, and J.D. Tygar. I know why you went to the clinic: Risks and realization of https traffic analysis. In Emiliano De Cristofaro and StevenJ. Murdoch, editors, *Privacy Enhancing Technologies*, volume 8555 of *Lecture Notes in Computer Science*, pages 143–163. Springer International Publishing, 2014.

[Mic88] Sun Microsystems. RPC: Remote Procedure Call Protocol specification: Version 2. RFC 1057 (Informational), June 1988.

[MKC08a] A. Müller, A. Klenk, and G. Carle. Behavior and Classification of NAT devices and implications for NAT-Traversal. *IEEE Special issue on Middleboxes*, pages 14–19, September 2008.

[MKC08b] A. Müller, A. Klenk, and G. Carle. On the Applicability of knowledge-based NAT-Traversal for future Home Networks. In *IFIP Networking 2008, Springer, Singapore*, May 2008.

[MKM06] Erwan Le Merrer, Anne-Marie Kermarrec, and Laurent Massoulié. Peer to peer size estimation in large and dynamic networks: A comparative study. In *15th IEEE International Symposium on High Performance Distributed Computing 2006*, pages 7–17, 2006.

[MLMKG06] Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *Proceedings of the twenty-fifth annual*

ACM symposium on Principles of distributed computing, PODC '06, pages 123–132, New York, NY, USA, 2006. ACM.

[MM02] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *1st International Workshop on Peer-to Peer Systems*, pages 53–65, Cambridge, March 2002.

[MNR02] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, page 183–192, New York, NY, USA, 2002. ACM, ACM.

[Moc87] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (INTERNET STANDARD), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604.

[Mur06] S. Murphy. BGP Security Vulnerabilities Analysis. RFC 4272 (Informational), January 2006.

[MV96] K. Moore and G. Vaudreuil. An Extensible Message Format for Delivery Status Notifications. RFC 1894, IETF, January 1996.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[NAL07] Axel Neumann, Corinna Aichele, and Marek Lindner. B.a.t.m.a.n status report. Technical report, The Better Approach To Mobile Ad-Hoc Networking Project, June 2007. http://open-mesh.net/batman.

[NS05] M. Nottingham and R. Sayre. The Atom Syndication Format. RFC 4287 (Proposed Standard), December 2005. Updated by RFC 5988.

[NSA13] NSA. Nsa prism program slides. `http://www.theguardian.com/world/interactive/2013/nov/01/prism-slides-nsa-document`, Nov 2013.

[OHKY10] Zhonghong Ou, Erkki Harjula, Otso Kassinen, and Mika Yliantila. Performance evaluation of a kademlia-based communication-oriented p2p system under churn. *Comput. Netw.*, 54:689–705, April 2010.

[OY02] L. Ong and J. Yoakum. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational), May 2002.

[Pay09] Bernd Paysan. `http://net2o.de/internet-2.0.html`, 2009.

[PG11] Bart Polot and Christian Grothoff. Performance regression monitoring with gauger. *LinuxJournal*, (209):68–75, September 2011.

[PG14] Bart Polot and Christian Grothoff. Cadet: Confidential ad-hoc decentralized end-to-end transport. In *IEEE/IFIP Annual Mediterranean Ad Hoc Networking Workshop (MedHocNet)*, 2014.

[PL98] Patrick Pantel and Dekang Lin. Spamcop: A spam classification & organization program. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[PM14] Trevor Perrin and Moxie Marlinspike. Axolotl ratchet. `https://github.com/trevp/axolotl/wiki`, 2014.

[Pol00] J. M. Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of Cryptology*, 13(4):437–447, Sep 2000.

[Pol10] Bart Polot. Adapting blackhat approaches to enhance the resillience of whitehat application scenarios. Master's thesis, Technische Universität München, 2010.

[Pos81] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[PP99] Pascal Paillier and David Pointcheval. Efficient public-key cryptosystems provably secure against active adversaries. In *Advances in Cryptology-ASIACRYPT99*, pages 165–179. Springer, 1999.

[Pur04] Gregor N. Purdy. *Linux iptables Pocket Reference*. O'Reilly Media, Inc., 2004.

[PZS+13] W. Michael Petullo, Xu Zhang, Jon A. Solworth, Daniel J. Bernstein, and Tanja Lange. Minimalt: Minimal-latency networking through better security. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 425–438, New York, NY, USA, 2013. ACM.

[RD01] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, 31(4):161–172, August 2001.

[RGRK04] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[RL96] Ronald L. Rivest and Butler Lampson. SDSI – a simple distributed security infrastructure. `http://groups.csail.mit.edu/cis/sdsi.html`, 1996.

[RMM10] J. Rosenberg, R. Mahy, and P. Matthews. Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun). RFC 5766 (Review Copy), IETF, February 2010.

[Rum07] Rudolph Rummel. *The Blue Book of Freedom: Ending Famine, Poverty, Democracy and War*. Cumberland House Publishing, 2007.

[RWHM03] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), March 2003. Obsoleted by RFC 5389.

[Sai12] Alex Fink Sai. Mnemonic .onion urls. `http://goo.gl/aOpKo`, February 2012.

[San06] Oskar Sandberg. Distributed routing in small-world networks. In *Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments*, pages 144–155, 2006.

[Sch12] Martin Schanzenbach. A Censorship Resistant and Fully Decentralized Replacement for DNS. Master's thesis, Technische Universität München, 2012.

[SDHH98] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[SE01] P. Srisuresh and K. Egevang. Traditional IP Network address Translator (Traditional NAT). RFC 3022, IETF, January 2001.

[SEnB07a] Moritz Steiner, Taoufik En-najjary, and Ernst W. Biersack. Exploiting kad: possible uses and misuses. *Computer Communication Review*, 37(5):65–70, October 2007.

[SENB07b] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. A global view of kad. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 117–122, New York, NY, USA, 2007. ACM.

[SFK08] P. Srisuresh, B. Ford, and D. Kegel. Rfc 5128: State of peer-to-peer (p2p) communication across network address translators (nats), 2008.

[SGH08] Tallat M. Shafaat, Ali Ghodsi, and Seif Haridi. A practical approach to network size estimation for structured overlays. In *Proceedings of the 3rd International Workshop on Self-Organizing Systems*, IWSOS '08, pages 71–83, Berlin, Heidelberg, 2008. Springer-Verlag.

[Sin14] Supriti Singh. Experimental comparison of byzantine fault tolerant distributed hash tables. Master's thesis, Saarland University, 2014.

[SL03] Andrei Serjantov and Stephen Lewis. Puzzles in p2p systems. In *8th CaberNet Radicals Workshop, Corsica*, 2003.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[SR06] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 189–202, New York, NY, USA, 2006. ACM Press.

[SS11] C. Soghoian and S. Stamm. Certified lies: Detecting and defeating government interception attacks against SSL. In *Proc. 15th. Int. Conf. Financial Cryptography and Data Security*, Mar 2011.

[Sta02] Richard M. Stallman. *Free Software, Free Society: Selected Essays of Richard M. Stallman.* GNU Press, Boston, Massachusetts, 2002.

[Sta12] Richard Stallman. Why software should not have owners. `http://www.gnu.org/philosophy/why-free.html`, 2012.

[STAH99] P. Srisuresh, G. Tsirtsis, P. Akkiraju, and A. Heffernan. DNS extensions to Network Address Translators (DNS_ALG). RFC 2694 (Informational), September 1999.

[Ste07] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335, 7053.

[Sti05] Marc Stiegler. An introduction to petname systems. `http://www.skyhunter.com/marcs/petnames/IntroPetNames.html`, February 2005.

[Swa11] Aaron Swartz. Squaring the triangle: Secure, decentralized, human-readable names. `http://www.aaronsw.com/weblog/squarezooko`, January 2011.

[Sze12] Maximilian Szengel. Decentralized evaluation of regular expressions for capability discovery in peer-to-peer networks. Masters, Technische Universitaet Muenchen, Garching bei Muenchen, 11/2012 2012.

[Tei17] Markus Teich. Implementing privacy preserving auction protocols. Master's thesis, TUM, Munich, 02/2017 2017.

[The13] The I2P Project. Transport overview - transport selection. `http://geti2p.net/en/docs/transport`, December 2013. accessed: 2014-04-25.

[THKS03] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. DNS Extensions to Support IP Version 6. RFC 3596 (Draft Standard), October 2003.

[Tot13a] Sree Harsha Totakura. Large scale distributed evaluation of peer-to-peer protocols. Masters, Technische Universität München, Garching bei München, 06/2013 2013.

[Tot13b] Gabor X Toth. Design of a social messaging system using stateful multicast. Master's, University of Amsterdam, Amsterdam, 2013.

[TP11] S. Turner and T. Polk. Prohibiting Secure Sockets Layer (SSL) Version 2.0. RFC 6176 (Proposed Standard), March 2011.

[UHHC11] Alexander Ulrich, Ralph Holz, Peter Hauck, and Georg Carle. Investigating the OpenPGP Web of Trust. In *16th European Symposium on Research in Computer Security (ESORICS)*, September 2011.

[vdBKM12] Ruud van de Bovenkamp, Fernando Kuipers, and Piet Van Mieghem. Gossip-based counting in dynamic networks. In *IFIP International Conferences on Networking (Networking 2012)*, pages 404–419, Prague, CZ, 05/2012 2012. Springer Verlag, Springer Verlag.

[vK12] Anne van Kersteren. Cross-origin resource sharing. Technical report, W3c Working Draft 3, `http://www.w3.org/TR/cors/`, April 2012.

[Wac15] Matthias Wachs. *A Secure and Resilient Communication Infrastructure for Decentralized Networking Applications*. Phd, Technische Universität München, München, 02/2015 2015.

[Wel00] B. Wellington. Domain Name System Security (DNSSEC) Signing Authority. RFC 3008 (Proposed Standard), November 2000. Obsoleted by RFCs 4035, 4033, 4034, updated by RFC 3658.

[WO06] Zooko Wilcox-O'Hearn. Names: Decentralized, secure, human-meaningful: Choose two. `http://zooko.com/distnames.html`, Jan 2006.

[WOG14] Matthias Wachs, Fabian Oelmann, and Christian Grothoff. Automatic selection and resource allocation for resilient communication in decentralized networks. In *IEEE International Conference on Peer-to-Peer Computing (P2P 2014)*, 2014.

[WS98] Duncan Watts and Steve Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[WSG13] Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. On the feasibility of a censorship resistant decentralized name system. In *6th International Symposium on Foundations & Practice of Security (FPS 2013)*, volume 8352 of *LNCS*, page 14. Springer Verlag, 2013.

[WSG14] Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. A censorship-resistant, privacy-enhancing and fully decentralized name system. In *13th International Conference on Cryptology and Network Security (CANS 2014)*, pages 127–142, 2014.

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. Sybilguard: Defending against sybil attacks via social networks. In *The ACM SIGCOMM'06 Conference*, pages 267–278. ACM Press, 2006.

[ZCSY08] Hai Zhuge, Xue Chen, Xiaoping Sun, and Erlin Yao. Hring: A structured p2p overlay based on harmonic series. *IEEE Trans. Parallel Distrib. Syst.*, 19:145–158, February 2008.