

Network Architectures  
and Services  
NET 2015-02-1

# Dissertation

## **A Secure and Resilient Communication Infrastructure for Decentralized Networking Applications**

Matthias Wachs



Network Architectures and Services  
Department of Computer Science  
Technische Universität München







TECHNISCHE UNIVERSITÄT MÜNCHEN  
Institut für Informatik  
Lehrstuhl für Netzarchitekturen und Netzdienste

## **A Secure and Resilient Communication Infrastructure for Decentralized Networking Applications**

Dipl.-Inform. Matthias Wachs

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Thomas Huckle
Prüfer der Dissertation:	1. Emmy Noether-Nachwuchsgruppenleiter Christian Grothoff, Ph.D.
	2. Univ.-Prof. Dr. Thomas Neumann
	3. Univ.-Prof. Dr. Uwe Baumgarten

Die Dissertation wurde am 16. Oktober 2014 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25. Januar 2015 angenommen.

Cataloging-in-Publication Data

Matthias Wachs

*A Secure and Resilient Communication Infrastructure for Decentralized Networking Applications*

Dissertation, February 2015

Network Architectures and Services, Department of Computer Science

Technische Universität München

ISBN 3-937201-45-9

ISSN 1868-2634 (print)

ISSN 1868-2642 (electronic)

Network Architectures and Services NET-2015-02-1

Series Editor: Georg Carle, Technische Universität München, Germany

© 2015, Technische Universität München, Germany

## ACKNOWLEDGMENTS

*Change will not come if we wait for some other person, or if we wait for some other time. We are the ones we've been waiting for. We are the change that we seek.*

---

Barack Obama

The work presented in Chapter 3 was published at CRiSIS 2010 [WGT12]. Chapter 5 is an extended version of [WOG14] presented at P2P 2014 and the adoption of machine learning approaches are described in Fabian Oelmann's master thesis [Oeh14]. Chapter 6 is based on the papers [WSG13] presented at FPS 2013 and [WSG14] presented at CANS 2014. The fundamental idea of GNS presented in Chapter 6 was first published in Martin Schanzenbach's master thesis [Sch12].

This research was supported in part by the OpenLab project, "OpenLab: extending FIRE testbeds and tools", funded by the EC under FP Grant agreement No 287581 and by the Deutsche Forschungsgemeinschaft (DFG) under ENP GR 3688/1-1.

I would like to thank my advisor, Christian Grothoff, for his guidance and patience throughout the course of my graduate career and that he made it possible for me to pursue my PhD in his research group. I want to thank Martin Schanzenbach for his valuable work on GNS during his master thesis, Fabian Oehlmann for his work on machine learning techniques for ATS, Omar Tarabai for his contributions for the OpenLab project and the efforts in his master thesis, and Robert Schirmer for his work on operations research methods and ATS. Special thanks go to everyone involved with the design of GNS for their insightful comments and discussions on the design. And finally I would like to thank all co-authors, students, all GNUet developers and all other contributors to my research efforts and everyone I forgot.



## ABSTRACT

The contribution of this thesis is the study, design and implementation of a resilient and secure communication infrastructure for decentralized peer-to-peer networks. On today's Internet, free and unrestricted communication between users is often restricted due to limited connectivity between participants, attempts to degrade service for certain traffic classes and other filtering and manipulation attempts. Peer-to-peer networks are in particular impacted by these effects since peer-to-peer networks are a target for censorship attempts due to the lack of a centralized node to control, their dependency on end-to-end connectivity and on a neutral network treating all traffic classes equally. In this thesis, we design and implement a communication infrastructure for decentralized peer-to-peer networks employing existing Internet infrastructure and technologies with the goal of re-establishing unhindered communication between users. The proposed communication infrastructure tries to provide and improve connectivity between participants and to improve quality of service for applications by detecting and counteracting traffic management and degradation attempts. The communication infrastructure has the goal to make communication resilient against censorship attempts and provides with GNS a public key infrastructure that provides a secure, resilient, and privacy-preserving way to map human-memorable names to addresses and other information.

This thesis starts with a motivation for the objective of this work introducing and explaining limitations with respect to unrestricted communication on today's Internet. We describe the different parties interested in controlling and influencing traffic on the Internet and their motivations and give an overview over technical restrictions to unrestricted communication a communication infrastructure tailored for the requirements of decentralized peer-to-peer networks and trying to provide resilient communication has to cope with and counteract.

As a second step, we analyze if the current Internet can provide a suitable foundation for resilient communication between peers in a peer-to-peer overlay network. For this evaluation, we analyze how resilient the Internet and its routing infrastructure is against Byzantine failures of providers or links between networks and what impact these failures have on routing in a peer-to-peer network. We use a graph representation of the Internet topology generated from BGP routing data and path measurement data and apply a new graph separation heuristic to find the smallest set of providers and networks to partition the Internet. The size of the resulting separators gives evidence how hard it is to partition the Internet in a way that prevents communication between partitions and therefore could have an impact on routing between peers in the peer-to-peer overlay.

We then present the design and implementation of a resilient and secure communication infrastructure for the GUNet peer-to-peer framework. This communication infrastructure has the goal to increase connectivity between participants, improve connectivity for peers in restricted environments, counteract service degradation and traffic manipulation attempts and provide secure communication between participants of the peer-to-peer network. One of the key approaches used to detect and counteract service degradation is to support multiple transport mechanisms and to provide the possibility to

switch from a degraded communication channel to an alternative channel providing better communication properties.

In a peer-to-peer network, a peer has to communicate with multiple communication partners at the same time. With a peer-to-peer communication infrastructure supporting multiple transport mechanisms and multiple client applications, the challenge arises which mechanism provides the best properties and satisfies application requirements best and how to distribute available resources among communication partners. With ATS, we present an approach for automatic transport selection and resources allocation tailored for the needs of decentralized peer-to-peer networks supporting multiple transport mechanisms and multiple applications with possibly contradictory requirements. We present three approaches to solve the problem of address selection and resource allocation based on a heuristic, solving the problem as an optimization problem and using machine learning techniques. We evaluate the performance of the solution approaches and compare the quality of solutions provided by the approaches.

Finally to allow users to communicate securely over the Internet and to address entities and services in the network, we present the design and implementation of the GNU Name System (GNS), a fully decentralized, privacy-preserving, and censorship-resistant name system designed as an alternative to the existing DNS. GNS is based on the idea of a *petname system* relying on local namespaces and linking namespaces using delegation to make names transitive. Due to its design it can also double as an alternative to existing security infrastructures like the X.509 public key infrastructure. GNS provides a privacy-preserving publication and lookup mechanism storing data encrypted in a DHT so only a benign user can perform publication and lookup operations successfully and attackers cannot monitor name publication and resolution in the network.

## ZUSAMMENFASSUNG

Der Fokus dieser Arbeit liegt auf dem Entwurf und der Implementierung einer ausfallsicheren und zensurresistenten Kommunikationsinfrastruktur für dezentralisierte Peer-to-Peer Netzwerke. Im heutigen Internet ist die Kommunikation zwischen Teilnehmern auf Grund fehlender Konnektivität und dem Versuch, bestimmte Dienste einzuschränken oder vollständig zu filtern, oftmals eingeschränkt. Peer-to-Peer Netzwerke sind von diesen Ansätzen besonders stark betroffen, da sie zum einen keine zentralisierten Komponenten besitzen, die man dediziert kontrollieren könnte, und zum anderen auf Grund ihrer starken Abhängigkeit vom Ende-zu-Ende Prinzip, das fordert, dass alle Verkehrsklassen gleich behandelt werden sollten. In dieser Arbeit entwerfen und implementieren wir eine Kommunikationsinfrastruktur für dezentralisierte Peer-to-Peer Netzwerke, die auf der existierenden Infrastruktur des Internets und dessen Technologien aufbaut und das Ziel hat, die uneingeschränkte Kommunikation zwischen den Nutzern wiederherzustellen. Zusätzlich stellt diese Arbeit mit dem GNU Name System (GNS) eine sichere und resiliente Public-Key-Infrastruktur bereit, die die Privatsphäre der Anwender schützt.

Die Grundlagen dieser Arbeit sind eine Übersicht über die Einschränkungen der Ende-zu-Ende Konnektivität und Versuchen, Dienste im heutigen Internet einzuschränken. Weiterhin analysiert die Arbeit, ob das Internet eine geeignete Grundlage für resiliente Kommunikation in einem Peer-to-Peer Overlay bieten kann. Hierbei untersuchen wir, wie widerstandsfähig basierend auf dem byzantinischen Fehlermodell das Internet und seine Routing-Infrastruktur gegen Ausfälle von Netzwerken oder Verbindungen zwischen Netzwerken sind und welche Auswirkungen ein solcher Ausfall auf das Routing in einem Peer-to-Peer Overlay hat. Auf diesen Arbeiten aufbauend, präsentieren wir den Entwurf und die Implementierung einer resilienten und sicheren Kommunikationsinfrastruktur für das GUNet Peer-to-Peer Framework. Einer der hierbei verwendeten Schlüsselansätze, um die Beeinflussung von Diensten zu erkennen und dieser entgegenzuwirken, ist die Verwendung mehrerer Kommunikationsprotokolle und die Möglichkeit zwischen diesen wechseln zu können. Die Arbeit stellt ein Verfahren für die automatische Auswahl des zu verwendenden Kommunikationsprotokolls und für die Ressourcenzuweisung an die einzelnen Teilnehmer vor, das mit Hinblick auf die Anforderungen eines Frameworks für Peer-to-Peer Anwendungen entworfen wurde. Wir stellen drei Ansätze zur Lösung dieses Problems aufbauend auf einer Heuristik, linearer Optimierung und maschinellem Lernen vor und bewerten die Leistung der einzelnen Ansätze und die Qualität erzeugten Lösungen. Abschließend stellen wir das GNU Name System (GNS) vor. Das GNS ist eine vollständig dezentralisierte, zensurresistente und die Privatsphäre der Anwender schützende Alternative zu DNS. GNS baut auf der Idee eines *petname systems* auf und verwendet lokale Namensräume und verknüpft diese mittels des Prinzips der *Delegation* um Namen transitiv zu verwenden. GNS legt Daten verschlüsselt in einer Distributed Hash Table ab und verhindert so, dass Angreifer aus mitgeschnittenem Verkehr des Systems Schlüsse auf Daten und Anwender ziehen können. Durch seinen Entwurf kann GNS zusätzlich als Alternative zu existierenden Public-Key-Infrastrukturen wie der X.509 Infrastruktur verwendet werden.



# Contents

<b>1. Introduction</b>	<b>1</b>
1.1 Thesis Objectives and Research Questions . . . . .	2
1.2 Positioning and Goals . . . . .	3
1.3 Contributions and Document Structure . . . . .	4
<b>2. Background</b>	<b>7</b>
2.1 The Internet Protocol Architecture . . . . .	7
2.2 Internet Layer Protocols . . . . .	8
2.2.1 Internet Protocol . . . . .	8
2.2.2 IPv4 . . . . .	8
2.2.3 IPv6 . . . . .	9
2.3 Transport Layer Protocols . . . . .	9
2.3.1 TCP . . . . .	9
2.3.2 UDP . . . . .	10
2.4 Domain Name System . . . . .	10
2.5 The X.509 Public Key Infrastructure . . . . .	12
2.6 Middleboxes . . . . .	13
2.6.1 DiffServ . . . . .	14
2.6.2 Deep Packet Inspection . . . . .	14
2.6.3 Packet Filter . . . . .	15
2.6.4 Network Address Translation . . . . .	16
2.6.5 DS-Lite . . . . .	19
2.6.6 Proxy Servers . . . . .	20
2.7 Centralized Client/Server Architectures . . . . .	21
2.8 Decentralized Peer-to-Peer Networking Architectures . . . . .	22
2.8.1 Structured and Unstructured Peer-to-Peer Architectures . . . . .	22
2.8.2 Distributed Hash Tables . . . . .	23
2.8.3 The GUNet Peer-to-Peer Framework . . . . .	24
2.9 Conclusion and Findings . . . . .	25
<b>3. Resilience of Communication on the Internet</b>	<b>27</b>
3.1 Introduction . . . . .	27
3.2 Background and Related Work . . . . .	28
3.3 Calculating Separators . . . . .	29
3.3.1 Finding Edge Separators . . . . .	29
3.3.2 Edge Separators for Weighted Graphs . . . . .	30
3.3.3 Finding Node Separators . . . . .	31
3.4 Graph Generation . . . . .	32
3.4.1 Construction of AS Graphs from BGP Routing Information . . . . .	33
3.4.2 Construction of AS Graphs from Traceroutes . . . . .	34

3.4.3	Merge of BGP and Traceroute Graphs . . . . .	34
3.4.4	Weight Generation . . . . .	34
3.5	Experimental Results . . . . .	36
3.5.1	Unweighted AS Graphs . . . . .	36
3.5.2	Weighted AS Graphs . . . . .	37
3.6	Discussion . . . . .	39
3.7	Conclusion and Findings . . . . .	39
<b>4.</b>	<b>Resilient and Secure Communication for Decentralized Networks</b>	<b>41</b>
4.1	Objectives . . . . .	41
4.2	Scope and Limitations . . . . .	42
4.3	Design and Implementation . . . . .	43
4.3.1	Peers and Peer Identities . . . . .	44
4.3.2	Plugin Specific Address Formats . . . . .	45
4.3.3	Generic Address Format . . . . .	46
4.3.4	Transport Sessions . . . . .	47
4.3.5	HELLO messages . . . . .	47
4.3.6	Bootstrapping and Neighbor Discovery . . . . .	48
4.3.7	Persistent Storage of Peer Information . . . . .	50
4.3.8	Address Management and NAT Support . . . . .	50
4.3.9	Overlay Topology Management . . . . .	52
4.3.10	Managing Active Addresses and Session . . . . .	52
4.3.11	The Transport Service . . . . .	53
4.3.12	The UNIX Domain Socket Transport . . . . .	70
4.3.13	The TCP Transport . . . . .	70
4.3.14	The UDP Transport . . . . .	71
4.3.15	The HTTP(S) Transport . . . . .	72
4.3.16	The WLAN Transport . . . . .	76
4.3.17	The Bluetooth Transport . . . . .	77
4.3.18	The Distance Vector Routing Transport . . . . .	78
4.3.19	Secure Communication Between Peers with CORE . . . . .	79
4.4	Evaluation . . . . .	80
4.4.1	Methodology and Setup . . . . .	80
4.4.2	Experimental Setup . . . . .	80
4.4.3	Methodology . . . . .	82
4.4.4	Results on Local Performance . . . . .	83
4.4.5	Results on Network Performance . . . . .	89
4.5	Related Work and Comparison . . . . .	92
4.5.1	Tor's Pluggable Transport Architecture . . . . .	92
4.5.2	SPOVnet's ARIBA Resilient Transport Underlay . . . . .	93
4.5.3	I2P's Transport Architecture . . . . .	93
4.5.4	BitTorrent Protocol and Obfuscation . . . . .	94
4.6	Conclusion and Findings . . . . .	94
<b>5.</b>	<b>Address Selection and Resource Allocation in Decentralized Peer-To-Peer Networks</b>	<b>97</b>
5.1	Background and Analysis of the Problem Setting . . . . .	98
5.1.1	Peers . . . . .	98
5.1.2	Transport Mechanisms . . . . .	98
5.1.3	Transport Mechanisms with Multiple Addresses . . . . .	99

5.1.4	Network Scopes . . . . .	99
5.1.5	Bandwidth Restrictions for Network Scopes . . . . .	100
5.1.6	Resource Allocation and Address Selection . . . . .	101
5.1.7	Transport Properties . . . . .	101
5.1.8	Application Requirements . . . . .	104
5.1.9	Summary . . . . .	105
5.2	Design and Architecture . . . . .	106
5.2.1	Input for the Transport Selection . . . . .	107
5.2.2	Output from Transport Selection . . . . .	107
5.2.3	Objectives for Transport Selection and Resource Allocation . . . . .	107
5.2.4	Scope and Limitations . . . . .	108
5.3	Input Normalization and Correlation . . . . .	109
5.3.1	Preference Normalization and Correlation . . . . .	109
5.3.2	Performance Property Normalization . . . . .	110
5.4	The Greedy Heuristic Solver . . . . .	111
5.4.1	Design of the Solver . . . . .	111
5.4.2	Discussion . . . . .	114
5.5	The Mixed Integer Linear Programming Solver . . . . .	114
5.5.1	Linear Programming . . . . .	114
5.5.2	Design of the Solver . . . . .	116
5.5.3	Discussion . . . . .	119
5.6	The Reinforcement Learning Solver . . . . .	119
5.6.1	Machine Learning . . . . .	120
5.6.2	Design of the Solver . . . . .	121
5.6.3	Discussion . . . . .	123
5.7	Related Work and Comparison . . . . .	123
5.7.1	Quality of Service in IP Networks . . . . .	123
5.7.2	The SpoVNet Project . . . . .	124
5.7.3	The Tor Project . . . . .	125
5.7.4	The Invisible Internet Project . . . . .	125
5.8	Implementation . . . . .	126
5.8.1	The ATS Service . . . . .	126
5.8.2	ATS Information . . . . .	127
5.8.3	Interacting with ATS . . . . .	128
5.8.4	Peer and Address Management . . . . .	129
5.8.5	Management of Transport Performance Properties . . . . .	130
5.8.6	Management of Application Preferences . . . . .	130
5.8.7	The Solver API . . . . .	131
5.8.8	ATS Solvers . . . . .	133
5.8.9	The Greedy Heuristic Solver . . . . .	133
5.8.10	The Mixed Integer Linear Programming Solver . . . . .	135
5.8.11	The Reinforcement Learning Solver . . . . .	141
5.9	Evaluation . . . . .	145
5.10	Discussion . . . . .	146
5.11	Conclusion and Findings . . . . .	148
<b>6.</b>	<b>GNS - A Decentralized, Privacy-Preserving and Censorship-Resistant Name System</b>	<b>151</b>
6.1	Introduction and Motivation . . . . .	151

6.2	Background . . . . .	156
6.2.1	The Domain Name System . . . . .	156
6.2.2	The Domain Name System Security Extensions . . . . .	156
6.2.3	SDSI/SPKI . . . . .	157
6.2.4	Distributed Storage in Peer-to-Peer Overlay Networks . . . . .	158
6.3	Functional Requirements . . . . .	158
6.3.1	Adversary Model . . . . .	158
6.3.2	Functional Requirements for an Alternative Name System . . . . .	159
6.4	Design Space for Name Systems . . . . .	159
6.4.1	Hierarchical Registration . . . . .	160
6.4.2	Adding Security to Hierarchical Registration . . . . .	161
6.4.3	Cryptographic Identifiers . . . . .	162
6.4.4	Making Cryptographic Identifiers Memorable . . . . .	162
6.4.5	Petname Systems . . . . .	163
6.4.6	Linking Local Namespaces . . . . .	163
6.5	Practical Considerations . . . . .	164
6.5.1	Interoperability with DNS . . . . .	164
6.5.2	End-to-End Security and Error Handling . . . . .	165
6.5.3	Legacy Applications . . . . .	166
6.5.4	Censorship-Resistant Lookup . . . . .	166
6.5.5	Privacy-Preserving Name Resolution . . . . .	166
6.6	Design of the GNU Name System . . . . .	167
6.6.1	Names, Zones and Delegations . . . . .	168
6.6.2	Zone Management with Nicknames and Petnames . . . . .	170
6.6.3	Relative Names for Transitivity of Delegations . . . . .	170
6.6.4	Censorship-Resistant and Privacy-Preserving Publication and Name Resolution . . . . .	170
6.6.5	Automatic Shortening . . . . .	172
6.6.6	Absolute Names in GNS . . . . .	173
6.6.7	Delegation to Legacy Name Systems . . . . .	173
6.6.8	Handling TLSA and SRV Records in GNU Name System (GNS) . . . . .	174
6.6.9	Records in GNS . . . . .	174
6.6.10	Shadow Records . . . . .	178
6.6.11	Revocation in GNS . . . . .	178
6.6.12	Dealing with Legacy Assumptions: Virtual Hosting and TLS . . . . .	179
6.7	Security Analysis . . . . .	180
6.8	Implementation of GNS . . . . .	181
6.8.1	Architecture . . . . .	181
6.8.2	Cryptography Used in GNS . . . . .	182
6.8.3	Identity Management for GNS . . . . .	182
6.8.4	Records in GNS . . . . .	183
6.8.5	Managing GNS Zones and Persistent Storage . . . . .	184
6.8.6	Caching GNS Information . . . . .	187
6.8.7	Zone Revocation with GNS . . . . .	188
6.8.8	Censorship-Resistant and Privacy-Preserving Publication and Name Resolution . . . . .	188
6.8.9	GNS Shortening . . . . .	193
6.8.10	GNS on Multi-User Systems . . . . .	194
6.8.11	Integration with the Name Resolution Process . . . . .	194

---

6.8.12	Accessing GNS from DNS . . . . .	198
6.9	Related Work and Comparison . . . . .	198
6.9.1	OpenDNS . . . . .	198
6.9.2	Namecoin . . . . .	199
6.9.3	TrickleDNS . . . . .	200
6.9.4	CoDNS . . . . .	200
6.9.5	Unmanaged Internet Architecture . . . . .	201
6.10	Use Cases for GNS . . . . .	201
6.10.1	Telephony . . . . .	201
6.10.2	Decentralized Online Social Networking . . . . .	202
6.10.3	Messaging . . . . .	203
6.10.4	DNSSEC Done Right: Securing the Web . . . . .	203
6.10.5	Other Applications . . . . .	204
6.10.6	Synergy . . . . .	204
6.10.7	Out-of-Band Exchange of Zone Information . . . . .	204
6.11	Conclusion and Findings . . . . .	205
<b>7.</b>	<b>Conclusion and Findings</b>	<b>207</b>
7.1	Future Work . . . . .	209
	<b>Bibliography</b>	<b>211</b>
	<b>List of Figures</b>	<b>226</b>
	<b>List of Tables</b>	<b>228</b>
	<b>Appendix</b>	<b>229</b>
<b>A.</b>	<b>Headers</b>	<b>231</b>
A.1	The NAMESTORE Plugin API . . . . .	231
A.2	The NAMECACHE Plugin API . . . . .	232



## 1. INTRODUCTION

Decentralized peer-to-peer overlay networks can enable censorship-resistant communication and help to re-establish free and unrestricted communication between users. The Internet and its predecessors were envisioned to provide the fabric interconnecting users and systems. With the increasing success of the Internet in the 1990s, many technic enthusiasts hoped that the Internet can provide the foundation for free exchange and communication between users worldwide [Fin01]. On today's Internet free and unrestricted communication is restricted and censored for various reasons. With the help of peer-to-peer overlays, we can overcome these limitations and re-establish free communication between users.

The architects of what has become the Internet today envisioned the Internet to be a *dumb* network interconnecting systems and users and forwarding data on a *best-effort* basis. But on today's Internet, the possibility to establish connections between users is limited for various reasons, preventing users to communicate directly with each other and degrading them to consumers of services provided by large service providers. The Internet was intended to treat all traffic equal on a best-effort basis, independent from source, destination or application. But on today's Internet, various parties try to abolish the idea of a *neutral* network and introduce traffic classes to prioritize or discriminate certain network traffic introducing a class society on the Internet. With the growing importance of the Internet in our lives, more and more parties are interested in monitoring communication between users and controlling and restricting access to services and information on the Internet, restricting freedom of expression and unrestricted communication between users. Common examples for censorship include blocking IP addresses, resetting TCP connections, and the censorship of names in the Domain Name System (DNS).

Decentralized peer-to-peer systems are in particular affected by these limitations. Due to their communication paradigm, every participant is required to directly communicate with other participants. Therefore, peer-to-peer systems are in particular affected by limitations to end-to-end connectivity. Peer-to-peer traffic is often discriminated against, since decentralized applications do not comply with the business models of providers and Internet companies. Furthermore do these applications not provide a centralized point to control communication and therefore are decentralized applications and their users often linked with illegal and criminal activities. Centralized architectures provide such a single point of control which can be used to control and restrict communication. Therefore, decentralized applications cannot rely on such architectures.

This thesis presents the study, design, and implementation of a secure, resilient, and censorship-resistant communication infrastructure tailored for the needs of fully-decentralized peer-to-peer networks. The proposed communication infrastructure aims to re-establish free and unrestricted communication between users in a peer-to-peer overlay and to provide the foundation for the realization of future, decentralized Internet applications. It has the objective to overcome limitations on today's Internet with respect to limited end-to-end connectivity between users and best-effort communication and to be resistant against censorship attempts.

## 1.1 Thesis Objectives and Research Questions

The main objective of this thesis is to analyze the requirements and present the design and implementation of a secure, resilient, and censorship-resistant communication infrastructure, tailored for the requirements of fully-decentralized peer-to-peer systems as a foundation for future decentralized networking applications. With this communication infrastructure, we focus on providing secure, resilient, and censorship-resistant communication between participants and provide connectivity in case of missing, degraded, or failing communication infrastructure. We improve connectivity for participants located in restricted network environments affected by limitations to end-to-end connectivity, counteract service degradation attempts, and provide a privacy-preserving and censorship-resistant way to refer to services and other information in the network.

With the presented communication infrastructure, we have to answer the question how a communication infrastructure designed with respect to the requirements of an application framework like GUNet, supporting multiple client applications at the same time and supporting multiple means of communication, can determine the “best” communication mechanism for each communication partner and allocate resources to optimally fulfill the applications’ requirements. To answer this question, we have to analyze the problem and the problem domain and develop the design and implementation of an automatic transport selection and resource allocation mechanism for decentralized peer-to-peer networks.

To provide users with the possibility to refer to services and entities in the network in a resilient and privacy-preserving way, we present the design and implementation of an alternative, censorship-resistant and privacy-preserving name system which allows users to address services and information in a resilient and secure way respecting the users’ privacy. With the GNU Name System (GNS), we have the objective to provide a generic approach to map names to values and provide an alternative for centralized, hierarchical public key security infrastructures like X.509.

Communication between participants and providing a secure way for users to address services and entities in the network are fundamental functionalities required for a fully decentralized peer-to-peer system. But realizing such a system on top of today’s Internet is challenging due to restrictions to end-to-end and best-effort communication on today’s Internet. Thus, this thesis answers the following questions:

- What are restrictions to end-to-end connectivity and best-effort communication on today’s Internet? How do adversaries try to prevent access to information and services on the Internet?
- Does the Internet provide a suitable foundation to re-establish end-to-end communication between users with a resilient peer-to-peer overlay?
- What are the requirements and a possible design for a secure and censorship-resistant communication infrastructure?
- How can a communication infrastructure support multiple communication protocols and optimally satisfy applications with a priori unknown requirements in a dynamic peer-to-peer environment?
- How can services and information be accessed in a resilient, decentralized and privacy-preserving way without relying on centralized authorities?

## 1.2 Positioning and Goals

For the design of new networking architectures and services, it is commonly distinguished between *clean slate* approaches, dismissing existing solutions to a given problem and proposing a new solution to the problem, and approaches building on top of existing solutions, using or trying to improve these approaches. For the proposed communication infrastructure, we rely on today's Internet as a communication underlay and try to re-establish unrestricted end-to-end connectivity and best-effort communication between participants using a peer-to-peer overlay to allow participants to communicate with each other. To realize this infrastructure, we rely on existing Internet infrastructure and established protocols. The communication infrastructure has the goal to provide connectivity for peers when existing communication infrastructure is failing, to attenuate limitations to end-to-end connectivity effecting communication between participants and to detect and counteract service degradation attempts. With the proposed infrastructure, we do not propose a clean slate approach replacing the Internet, but build on top of existing, established infrastructures and protocols using them as a foundation to re-establish end-to-end connectivity between users.

One approach to make communication resilient against censorship or service degradation is to make the traffic less susceptible for filtering and traffic shaping appliances using *traffic hiding* or *traffic morphing* techniques. With *traffic hiding*, network traffic is obfuscated or hidden within other network traffic to prevent an attacker from detecting and classifying traffic as illegitimate. With *traffic morphing*, applications try to make their network traffic look like traffic from a legitimate networking application to prevent an attacker to classify it as illegitimate. With our communication infrastructure, we do not rely on these approaches as fundamental design concepts to make communication resistant against censorship. We consider the use of such techniques as an arms race between developers and censors with the advantage on the side of the censor. With our communication infrastructure, we rely on an extensible design which can be extended with new communication protocols and detecting degradation attempts and switching to a different, not affected communication channel.

To make communication resistant against censorship and degradation, several state-of-the-art peer-to-peer applications support multiple transport mechanisms with the idea to switch between these mechanisms. With our approach for automatic transport selection and resource allocation, we focus on how to optimally satisfy application requirements with in advance unknown, possibly contradictory requirements in an unknown and dynamic environment, an approach particularly useful for a peer-to-peer framework. With our approach, we have the objective to find a solution making communication resilient against degradation and optimally fulfilling application requirements at the same time by finding the optimal communication mechanism for each communication partner in the peer-to-peer overlay and providing a resource allocation reflecting the importance of each communication partner. With security being a focus of our approach and our assumptions about the environment containing malicious participants, peers do not collaborate and do not try find a globally optimal solution. With our approach, every peer tries to find a solution optimal for the respective peer based on its local view.

With GNS, we propose an alternative name system as a replacement for the existing Domain Name System (DNS) and security infrastructures. Several other approaches try to improve DNS to provide authenticated or confidential name resolution or propose alternative systems to replace DNS. But none of these approaches can achieve the design goals we require for a fully decentralized, censorship-resistant and privacy-preserving name

system with the adversary model used in this work assuming a powerful adversary. We therefore propose with GNS an alternative to DNS but focus in the design on integration of both systems and integration with existing applications. With GNS, we have the goal to unify the functionality of name systems and security infrastructures in an integrated approach.

### 1.3 Contributions and Document Structure

The main contribution of this thesis is the study, design, and implementation of a secure, resilient and censorship-resistant communication infrastructure for decentralized peer-to-peer applications with a focus on improving connectivity between peers and re-establishing best-effort, end-to-end connectivity in the peer-to-peer overlay. With GNS, the design and implementation of a fully decentralized, censorship-resistant and privacy-preserving name system and security infrastructure is presented.

The thesis starts with an overview over the architecture of the Internet and important protocols used on today's Internet in Chapter 2. With this overview, the thesis presents restrictions and limitations to unrestricted end-to-end connectivity and best-effort communication, particularly important for the design of a communication infrastructure focusing on the requirements of peer-to-peer systems. This chapter also introduces DNS manipulation as an approach often employed by adversaries to censor services and remove information on the Internet.

To elaborate if the Internet can provide a reasonable foundation for a peer-to-peer system trying to re-establish end-to-end connectivity between participants using a peer-to-peer overlay, this thesis analyzes in Chapter 3 how resilient the Internet's routing infrastructure is against a Byzantine failure of networks or communication links. With this analysis, we show that the backbone of the Internet is surprisingly well connected and hard to partition and therefore provides a reasonable foundation for a peer-to-peer overlay. Special focus has therefore to be put on improving connectivity for participants in restricted perimeter networks.

In Chapter 4, we present, based on the findings of the previous chapters, the design and implementation of a resilient and secure communication infrastructure tailored for the needs of decentralized peer-to-peer systems. A key contribution of this infrastructure is the support of multiple communication protocols and the functionality to monitor and detect degradation attempts against communication mechanisms and switch mechanisms in case of a degradation attempt.

In Chapter 5, we answer the question how a communication infrastructure should select addresses and allocate resources to communication partners to optimally satisfy application requirements with respect to communication properties. A main contribution of this chapter is an in-depth analysis of the given problem domain and the design of an automatic transport selection and resource allocation mechanism and its implementation in the GUNet peer-to-peer framework. We present three different solution approaches to find a solution to the given problem, show the suitability of the proposed approaches and compare the quality of the solutions provided in different scenarios.

With GNS presented in Chapter 6, we present the design and implementation of a fully-decentralized, censorship-resistant and privacy-preserving name system as an alternative to DNS. We start with giving an overview over current name resolution on the Internet and define the adversary model used with GNS. Based on this work, we explore the possible design space for name systems and present the design and implementation of GNS. This chapter also elaborates how GNS can replace existing security infrastructures

like X.509 and how GNS can be integrated with common usage patterns on the Internet and applications and we highlight additional use cases for GNS.

In Chapter 7, we summarize and discuss the work done in this thesis and highlight the important findings elaborated in this work. Finally, we give an outlook on future work to be done with the respective topics of this work.



## 2. BACKGROUND

### 2.1 The Internet Protocol Architecture

Communication on the Internet is based on a whole set of different protocols, each focusing on a different aspect of communication. These protocols rely on each other and therefore create a *protocol stack*. To describe the interaction of protocols in this protocol stack and the functionality provided by protocols, the protocols are often fitted into *network models* to represent this stack. These models intend to represent and standardize the required aspects and functionalities to communicate over the network as a layered architecture and fit the existing protocols and network architectures into the model.

Network models are often seen controversial since they represent a theoretic design and do not represent the reality on the network. Some network models create layers in the architecture not existing in reality or force protocols to fit into this layered architecture. Nonetheless if not seen too strict, these models are useful to describe the architecture and interaction of network protocols.

Internet protocol architecture is often represented using two models: the Open Systems Interconnection Model (OSI) and the *Internet model*. The OSI model, maintained by International Organization for Standardization (ISO) and defined in [ISO94], defines seven layers, numbered from 1 to 7, to describe the network model: physical layer (layer 1, physical connections), data link layer (2, links between nodes connected directly), network layer (3, (unreliable) communication between nodes on the same network), transport layer (4, reliable communication between nodes on the same or different networks), session layer (5, sessions between applications), presentation layer (6, data conversion) and application layer (7, connections between applications). On different hosts, entities located on the same layer communicate with each other.

The TCP/IP model of the Internet introduces four layers communication with between network entities: communication between hosts directly connected *link layer*, communication between networks (*Internet layer*), end-to-end communication (*transport layer*) and communication between software applications (*application layer*).

In both models, at the lowest layer of the stack mechanisms are required to access the network physically and to communicate with other hosts connected directly and on the same network (layer 1-2 in OSI model and link layer in TCP/IP model). In addition, we need to communicate with hosts not located on the same network (layer 3 in OSI model and Internet layer in TCP/IP model) and establish connections to exchange data with these hosts (Layer 4 in OSI model and transport layer in TCP/IP model) to give these to the respective network application (Layer 7 in OSI model and application layer in TCP/IP model). Both models are based on the idea that from the bottom of the stack to the top, protocols rely on the underlying protocols to realize their functionality and that protocols on the same layering level communicate with each other. The models rely on encapsulation, so upper layers encapsulating their traffic in the lower layer protocols.

This work does not focus on the aspects of accessing the network itself except for minor aspects when describing the Wireless Local Area Network (WLAN) and Bluetooth

TRANSPORT plugins in Section 4.3.16 and Section 4.3.17. In the remainder of this section we will therefore focus on giving a short introduction to protocols related to communication between hosts on network and Internet level and do not focus on the physical and (data)link layers.

## 2.2 Internet Layer Protocols

### 2.2.1 Internet Protocol

The Internet Protocol (IP), defined in [Pos81a], is the standard protocol used for packet forwarding and routing on the Internet. IP is a protocol located on the network layer of the (conceptual) ISO/OSI model [ISO94], providing connection-less packet forwarding without reliability on a best-effort basis. It defines the IP address space and with its routing functionality allows the linking of networks, the fundamental concept of the Internet.

Following the idea of the end-to-end principle to realize functionality in the end systems of the network, packets are forwarded statelessly in the network and without error recovery in the intermediate systems. Error correction is supposed to be realized in the end systems. When an intermediate system forwarding a packet detects a corrupted packet, it dismisses the packet. Due to this design, various error conditions can occur: data corruption, packet loss, packet duplication and out-of-order delivery of packets.

The IP protocol was proposed by Cerf and Kahn in 1974 and specified 1980 in [Pos81a]. At the moment to different version of the IP protocol are used: version 4 called Internet Protocol version 4 (IPv4) and version 6 called Internet Protocol version 6 (IPv6), both described in the next sections.

### 2.2.2 IPv4

IP Version 4 or IPv4, defined in [Pos81a], uses a 32-bit address space and provides about  $2^{32}$  possible addresses. IPv4 addresses are commonly represented using a dotted decimal representation: each byte as a decimal value separated by a dot. Originally an IP address was split in two parts: the *network* part and the address itself. Five different network classes, named from class A to E, were defined by a start address and the length of the network address for each network. This *classfull* approach was in 1981 replaced by Classless Inter-Domain Routing (CIDR) to allow more flexible address allocation in smaller blocks. With CIDR, variable length network addresses are possible by representing the IPv4 address together with its routing prefix mask defining the size of the IP network. Address blocks are assigned to users by the Internet Assigned Numbers Authority (IANA) and the regional sub-organizations called Regional Internet Registries (RIRs). IPv4 reserves several *special use address blocks* and addresses ([CV10]) used for testing and documentation purposes and for addresses with a limited scope requiring special treatment like loopback, link-local and multicast addresses ([CAG05, AAMS01, Mog84]) and private IP networks ([RMK+96]). IPv4 provides connectionless, unreliable, best-effort communication. Due to varying possible message sizes on underlying network layers, IPv4 supports packet fragmentation and reassembly. For this functionality, it employs a Maximum Transmission Unit (MTU) value indicating the maximum size of a datagram. To realize its functionality, IPv4 uses several helper protocols like Address Resolution Protocol (ARP) to resolve IPv4 to link layer addresses or Dynamic Host Configuration Protocol (DHCP) to automate address assignment.

### 2.2.3 IPv6

With the increasing success of the Internet, its commercialization and the increasing number of users, it became evident in the beginning 1990s that a successor for IPv4 is required to counteract exhaustion of IP addresses and networks and the increasing size of routing tables [GA92]. As a successor for IPv4, IPv6 is defined [DH98] published in December 1998. IPv6 was designed to replace IPv4 but to be inter-operable providing the possibility of a transition process [BM95]. IPv6 provides a 128-bit address space. Each address space is commonly represented as eight groups of four hexadecimal digits separated by colons. IPv6 provides several improvements over its predecessor. To provide one-to-many communication, IPv6 provides *multicasting* with designated multicast addresses for different *multicast scopes* [HD06]. It provides Stateless Address Autoconfiguration (SLAAC) to allow hosts to autoconfigure network interfaces with IPv6 without manual configuration and only minimal configuration of routers and without additional servers [TNJ07]. Like IPv4, IPv6 depends on several other protocols like Neighbor Discovery Protocol (NDP) and Internet Control Message Protocol Version 6 (ICMPv6) to realize its service. To allow a seamless transition from IPv4 to IPv6, several transition mechanisms were developed to allow the coexistence of both protocols. With *dual-stack* defined in [NG05], hosts and routers provide full support for both IPv4 and IPv6 and both protocols can coexist on the same network. To support networks only capable of one of both protocols, transition mechanisms like *6to4* [CM01], *Teredo* [Hui06], *6in4* [NG05] and *ISATAP* [TGT08], encapsulating one protocol in another were developed. The general transition process from IPv4 to IPv6 can be described as slow: according to Google's IPv6 statistics<sup>1</sup>, in September 2014 only 3.5-4% of all users used native IPv6 to access Google's services. One reason for the slow transition is the complexity to ensure functioning of IPv6 in already deployed infrastructure but also a missing psychological strain for companies to deploy IPv6.

## 2.3 Transport Layer Protocols

Transport layer protocols provide end-to-end connectivity with additional features required by higher layer network applications using an underlying network layer protocol like IP, providing packet delivery between network systems. Since IP is connectionless, does not protect against message loss nor prevent packet duplication or out-of order delivery, transport protocols provide additional features depending on their focus and design. On today's Internet multiple transport layer protocols are defined, but Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are the most prominent examples. Other examples (not in the focus of this work) are Stream Control Transmission Protocol (SCTP), Datagram Congestion Control Protocol (DCCP) and Resource Reservation Protocol (RSVP).

### 2.3.1 TCP

TCP, defined in [Pos81b], provides connection-oriented and reliable transfer between end systems. It provides reliable communication with error checking, ensures in-order message delivery and protects against packet loss using acknowledgments and retransmissions. In addition, it provides congestion control to not overload the network and flow control to not overload the receiver. TCP uses *port numbers* with a value between 0 and 65535 to distinguish between applications. So a TCP connection is identified by the five-tuple IP

<sup>1</sup> <https://www.google.com/intl/en/ipv6/statistics.html>

protocol version, source address, source port, destination address and destination port. Ports numbers are divided in three classes: system or well-known ports (0..1023), user or registered ports (1024..49151) and dynamic or private ports (49152..65535) [CET<sup>+</sup>11]. Port numbers are assigned by IANA: to reserve a port for an application, a port number in the user range can be registered with IANA. To register a port in the well-known range requirements are stricter and a requester must document why a port from the user range is not suitable for an application. Dynamic ports can be freely used by any application and are never registered with IANA. Private ports are used by applications when establishing an outbound connection: applications use a private port number as the source port for the outgoing connection. It is not possible to use this private port number to establish a connection with the application. TCP is used with many modern applications on the Internet and is particularly well-suited for connection-oriented applications not tolerating loss.

### 2.3.2 UDP

UDP, defined in [Pos80], provides lightweight a datagram service on top of IP. Like IP, it provides connection-less and unreliable communication without ensuring packet delivery, packet order and without preventing duplicates; however, UDP does ensure packet integrity with checksums. So UDP exposes major design principles of underlying layers to applications. Like TCP, UDP uses port numbers, with the same regulations as for TCP ports, to distinguish between different applications on a host. UDP is widely used with applications requiring only simple data transfer and able to tolerate loss and out-of-order delivery in exchange for avoiding the overhead of acknowledgments and connections setup. UDP is often used with time sensitive applications like DNS, media streaming services, video or speech telephony (for example in combination with Real-Time Transport Protocol (RTP) [GSC<sup>+</sup>96]) or stateless applications not requiring explicit acknowledgments for transmitted datagrams.

## 2.4 Domain Name System

IP addresses, as used with the protocols described in the previous sections, are hard to memorize for humans. Therefore, mechanisms to translate between addresses used by machines to communicate with each other and names memorable to humans are required. To provide easy to remember names for humans to address services on the Internet, the Domain Name System (DNS) provides a hierarchical names space to address services on the Internet. DNS is most commonly used to translate domain names to numerical IP addresses or obtain additional service information for a domain like mail server or the responsible name server.

DNS, defined in [Moc87a] and [Moc87b], is organized as a distributed database. The root of the hierarchical DNS namespace is called the *root zone*. The hierarchical namespace is managed by delegating control over portions of the namespace to subordinate organizations. These delegated portions are called *zones* in DNS and organizations are called *authoritative* for such a portion of the namespace. These organizations themselves can again delegate control over subdomains in their zone to other organizations, thus forming a hierarchy in the namespace. At the time of writing, the root zone was managed by the Internet Corporation for Assigned Names and Numbers (ICANN) in person of the IANA but ultimately controlled by the National Telecommunications and Information Administration (NTIA), an agency of the United State Department of Commerce [TA12].

NTIA announced to cede this authority to a yet to announce organization, but details of this transition are still unclear [Far14b]. DNS provides two namespaces: the *domain name* namespace, used to map names to addresses, and the IP namespace, used to map addresses reversely to addresses. DNS names represent the hierarchical structure of the DNS name space: a *domain name* is a sequence of *labels* separated by dots, with the right-most label being the empty label representing the root zone and each label representing an subordinate zone. The root zone is the highest level of the hierarchical DNS name space. Domains located a level below the root zone are the Top Level Domains (TLDs). For TLDs, DNS distinguishes between country-code Top Level Domains (ccTLDs) like “de”, “uk”, “us” and generic Top Level Domains (gTLDs) like “biz”, “info” or “org”. Domains one level below of TLDs are called *second-level domains*. To resolve a DNS name, clients use a DNS *resolver*. A DNS resolver recursively resolves the labels in a DNS name by querying the responsible DNS servers starting with right-most label and querying the root servers responsible for the servers authoritative for the TLD. A resolver can be configured to query only a single DNS server which performs the name resolution for such a *stub resolver*. Today many operating systems (including Microsoft Windows [Mic09] or GNU/Linux distributions like Ubuntu using dnsmasq [Kel14]) only implement stub resolvers and rely on DNS servers to perform the name resolution. These DNS servers are often provided by companies or Internet Service Providers (ISPs) and provide additional functionalities like DNS caching to reduce load for root servers caused by recursive lookups. Within the network models presented before, DNS is an application layer protocol. To resolve names, DNS relies on UDP. Using DNS names to address services provides many benefits: with DNS load balancing and failover can be easily implemented providing more than one address for a name. Therefore, most applications in higher layers use DNS names to addresses services on the Internet instead of using IP addresses.

DNS, as a key service for the functioning of Internet, is a target for attackers and censors to make information on the Internet inaccessible. DNS is a popular target for attackers, especially for censors with a political motivation and legal or executive powers as described in Section 1. DNS is based on a hierarchical design with control delegated from the root to subordinated organizations. Since these organizations, in particular organizations managing ccTLDs, are often located within the legislative and executive sphere of control of the censor, it is easy for a powerful attacker like a nation state using its legal or executive powers to force these organizations to remove or modify DNS information under their control. The attacker tries to remove these information from DNS or DNS resolvers to make services inaccessible or modify DNS resolution to redirect access to attacker controlled impostor sites. Here the use of stub resolvers forwarding DNS requests to DNS servers often provided by ISPs has negative consequences, since for an attacker it is easy to force an organization like an ISP falling in under the jurisdiction of the attacker to manipulate DNS name resolution.

Well known examples for this approach to limit connectivity and remove information from the Internet are Turkey’s blocking of Twitter and YouTube in April 2014 to prevent access to an audio recording of a phone call made by the Prime Minister [Raw14, HA14], Germany’s attempt to prevent access to child pornography using a stop sign impostor site or the banning of the PirateBay website using *DNS hijacking*. Here attackers manipulate DNS servers or responses to make name resolution fail or to redirect users to an attacker controlled impostor site. These manipulations of the DNS often have side effects like the mandatory “pornography filter” in the UK blocking in addition to pornography also educational and charity sites [Kan14, Cow14], the incident with a Danish policemen accidentally blocking 8,000 legitimate sites including Google and Facebook [Hol12].

DNS is a key service for the functioning of the Internet but was designed without any focus on security. Domain Name System Security Extensions (DNSSEC) is an extension to DNS providing authenticity and integrity for DNS information. With DNSSEC, DNS responses are cryptographically signed by the operator of the respective zone. DNSSEC establishes a *chain of trust* a resolver can follow to verify the authenticity and integrity of DNS responses protected with DNSSEC. But DNSSEC does not protect against an attacker like a nation-state as defined in Chapter 1 using his legal or executive powers to force zone operators to modify name resolution or make name resolution fail. Since DNSSEC does not provide any confidentiality for the name resolution process, it does not protect the users and their privacy against attackers monitoring DNS name resolution to track down users.

A resilient and secure communication infrastructure should be designed to not depend on DNS information to establish connections between participants. Instead it protects itself against DNS manipulation attacks and should use IP addresses to establish connections between participants. To provide a censorship-resistant and privacy-preserving alternative to DNS for users to access and link information on the Internet and to provide secure communication on the Internet, we focus in Chapter 6 on the development of the GNU Name System (GNS), a fully decentralized, censorship-resistant, privacy-preserving name system, which can also double as a replacement for the X.509 public key infrastructure and the Certification Authorities (CAs) this infrastructure relies on.

## 2.5 The X.509 Public Key Infrastructure

Initially, the Internet was designed without a focus on security. With the emerging success of the Internet, secure communication became an important aspect to provide authenticated, confidential and integrity protected communication between entities and services on the Internet. Not all transport layer and link layer protocols deployed on the Internet do provide mechanisms to protect authenticity, confidentiality and integrity of communication. Since modifying existing and deployed protocols is hard to achieve, these security properties had to be added on higher layers. To provide secure authenticated and confidential communication between participants, the exchange of cryptographic key material is required. To provide secure communication often asymmetric cryptographic methods are used, where public keys are used for authentication and to exchange a shared secret. This shared secret is used to protect communication using symmetric cryptography. Since on the Internet entities wanting to communicate in a secure way with each other do not know each other a priori, a security infrastructure to map identifiers to public keys is required.

X.509 is a public key security infrastructure used to bind identifiers to cryptographic public keys. X.509<sup>2</sup> is an ITU Telecommunication Standardization Sector (ITU-T) standard defining formats for public key certificates and methods for key revocation and certificate validation. An essential aspect of X.509 are *certificates*. Certificates provide a cryptographic binding of a *distinguished name* in X.500 format to a cryptographic public key. These certificates are issued by a trusted third party, a so called *Certification Authority (CA)*. A certification authority can issue certificates or *certification authority certificates* to enable other entities to issue certificates. With this approach a trust path between a certificate and a *trust anchor*, the root in the trust chain, can be established. To allow entities to successfully validate certificates, the user must possess the certificate of the CA issuing the certificate and trust this CA. The user is often provided with this

<sup>2</sup> <https://www.itu.int/rec/T-REC-X.509/en>

trust anchor information out of band. In practice, this trust anchor information is often shipped to the user with the operating system or the software used (e.g the web browser). X.509 also provides mechanisms for key revocation. When a certificate is revoked, it is added to a Certificate Revocation List (CRL) and this CRL has to be obtained and evaluated by the client to check if a certificate is revoked. An alternative to CRLs is the Online Certificate Status Protocol (OCSP) [SMA<sup>+</sup>13] and OCSP stapling [3rd11]: OCSP is a protocol clients can use to obtain the revocation status of X.509 certificates from the OCSP *responder*. With OCSP, a responder is typically a server run by the certificate issuer. With OCSP stapling, the certificate owner queries the responder on its own and includes (*staples*) the OCSP response signed by the issuer with the certificate given to client (e.g. in the Transport Layer Security (TLS) handshake). This approach improves performance of connection setup and revocation checking since the client does not have to perform the request, improves privacy for the client since the certificate issuer does not see the requests and reduces load for the responder of the certificate issuer.

X.509 provides a security infrastructure and defines formats and methods, but does not provide a specific infrastructure and is not located on a specific layer of a network model. Therefore, it can be used in various different environments and in network protocols on different layers of the network model. X.509 is today widely used in various protocols to provide secure communication between participants and systems.

Prominent examples for protocols relying on the X.509 infrastructure are TLS and SSL [DA99, DR08] and Datagram Transport Layer Security (DTLS) [RM12], providing secure communication for networking applications on application layer. TLS/SSL and DTLS are located on the session layer of the OSI model and use an underlying transport layer protocol to transparently encapsulate application traffic from higher layers over a secure session. TLS/SSL is designed for reliable transport protocols and uses TCP whereas DTLS is designed to be used with datagram transport protocols like UDP. TLS/SSL is used with HTTP Secure (HTTPS), where it is used to authenticate web servers and establish secure communication between client and server. Another example is S/MIME [Ram99] used for secure e-mail based on X.509 certificates to provide authenticated and confidential mail communication.

## 2.6 Middleboxes

One of the original visions of the Internet was to provide unlimited connectivity between systems, so every system could communicate with every other system connected to the network: the paradigm of end-to-end connectivity as described in [SRC81]. The purpose of the network was to forward data between end systems: the network should be *dumb* and all functionality should be realized in end systems and not in the network itself. All data on the Internet should be treated equally and forwarded with a *best effort* approach: the network was supposed to be *neutral*.

On today's Internet a large variety of systems with various different purposes contradicting this idea of a dumb network exist in the network. These so called *middleboxes* inspect, prioritize, transform or filter traffic routed on the Internet and often have an impact on connectivity and influence communication between end systems [CB02]. In the following sections, we give an overview over approaches deployed on today's Internet and their impact on connectivity and communication.

### 2.6.1 DiffServ

Differentiated Services or *DiffServ*, defined in [NBBB98], [BBC<sup>+</sup>98] et al., is a technique for traffic management in IP networks. With DiffServ Quality of Service (QoS) can be improved by distinguishing between different traffic classes. So time critical services like telephony or video services can be prioritized over traffic without time constraints. DiffServ works on the network link layer of the ISO/OSI network model and therefore provides layer 3 QoS.

DiffServ uses a 6-bit Differentiated Services (DiffServ) field in the 8-bit Type of Service (TOS) field of the IPv4 header and the 8-bit Traffic Class (TC) field of the IPv6 packet header. This DiffServ field contains a Differentiated Services Code Point (DSCP) value encoding the respective traffic class for this packet.

In a *DiffServ domain* (a group of routers implementing the same policies) this traffic classification is used to handle and manage the packets according to their traffic class, called the per-hop behavior (PHB). Common traffic classes are the default traffic class Default Forwarding PHB (DF PHB) for traffic not matching any other class, the Expedited Forwarding PHB (EF PHB) suitable for low latency, low jitter and low loss traffic, handled strictly prioritized in routing queues, Voice Admit PHB (VA PHB) used for telephony handled similar to EF PHB but with a call setup phase and the Assured Forwarding PHB (AF PHB) class assuring priority as long as a specified rate is not exceeded.

DiffServ provides a lightweight, easy to implement traffic management mechanism allowing to define certain traffic classes for prioritized forwarding. By design, DiffServ tries to prioritize traffic and does not try to degrade specific traffic classes. Traffic management attempts can conflict with the best-effort principle demanding to handle all traffic equally and to provide the best-effort service for all traffic classes.

### 2.6.2 Deep Packet Inspection

Deep Packet Inspection (DPI) is a technique used to classify network traffic based on its content and higher layer information by examining the content of data packets passing a network. Contrary to routing or switching, where normally only the header of the protocol of the respective networking layer is inspected to allow decision making, with DPI, the header and the content of the packet is inspected. So classification decisions can be made upon a whole set of information including traffic source and destination, transport layer protocol used, the application generating the traffic (based on transport layer ports) and additional application layer information like application layer protocol, application content transmitted over the network and the use of encryption in the application data. DPI can be used to create statistics about network traffic, for traffic management to handle traffic according to its classification and traffic policy enforcement. Besides network management tasks, DPI can be used to manage traffic to provide quality of services and for lawful interception and copyright enforcement by analyzing traffic for illegitimate content.

DPI is a well adapted and widely used technology and large number of ISPs in various countries are reported to use DPI [DMG<sup>+</sup>10]. The use of DPI is strongly related to traffic management and often to surveillance and censorship attempts as analyzed in [AvEM12]. Since DPI can be used to classify traffic according to certain properties, the use of DPI has impacts on privacy and best-effort handling of service as demanded by the principle of network neutrality and on end-to-end connectivity when applied to traffic management.

### 2.6.3 Packet Filter

A popular approach to secure computer systems or entire network segments is the use of packet filters, commonly also known as *firewalls*. Packet filters are used to restrict the incoming or outgoing traffic to or from a network segment or connected systems. A packet filter can be a dedicated physical network security device or the packet filtering functionality can be realized in software and performed by devices realizing other network functionality like routers or gateways. By using a dedicated device, a whole network segment can be protected by locating this device on a prominent location where all network traffic has to bypass as shown in Figure 2.1. Packet filtering software can also be installed directly on end systems, protecting only these devices. By installing packet filters directly on end devices, mobile devices frequently switching their location can be protected from attacks and traffic flows between systems inside a protected network segment can be controlled. Most modern operating systems ship with some packet filtering solution included to protect the system from potentially dangerous network traffic.

Every connection between systems on a network can be described by several properties like source and destination, transport protocol used, etc. A packet filtering software inspects the incoming and outgoing network traffic based on predefined rules and, if the traffic matches one of these rules, the respective action is executed, or a default rule if none of the explicit rules match. This matching can be performed based on destination or destination address, network protocol, TTL, etc.

Literature commonly distinguishes between stateless network layer filters, stateful network layer filters and application layer filters [IF02].

Network work layer filters work on the lower layers of the Transmission Control Protocol/Internet Protocol (TCP/IP) stack and can be distinguished between stateless and stateful filters. Stateless packet filters are the first generation of packet filters. They are cheaper to implement since they only apply their filter rules on the networking traffic and do not pay respect to the state a network stream is in. A stateful packet filter maintains more information about a connection and traffic history and can prevent more sophisticated attacks. It still uses static rules but these rules are now extended with a state.

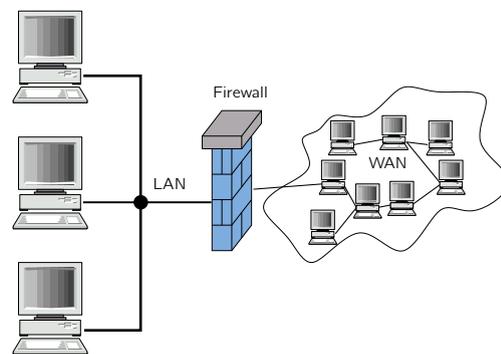
Application layer filters work on the application layer of the network protocol stack and can understand certain application protocols (like for example FTP, DNS or HTTP). These filters inspect network traffic on application layer and apply rules based on actions and the state of the application protocol. Application layer filters can be used to prevent the spread of malware by inspecting the payload of websites or to detect applications trying to bypass a firewall system (e.g. by using port numbers different from the default port number assigned to a protocol blocked by the filter).

Packet filters are mostly used to improve security of network segments or end-systems. Their main purpose is to prevent potentially dangerous network traffic, restricting undesired network traffic or restrict communication to a subset of allowed machines: access to specific application ports can be completely restricted or limited to a trusted set of sources. Specific network protocols can be blocked (e.g. to prevent spam) or specific ports can be blocked (e.g. to prevent the spreading of viruses using security vulnerabilities in networking applications).

The impact packet filters have on connectivity on the Internet is hard to estimate. The limitations caused by packet filters strongly depend on the purpose they are deployed for and how restrictive the filter rule set is defined. Organizations can follow a blacklisting approach and only filter very specific protocols and ports whereas organizations very concerned about security can apply a whitelisting approach, blocking all network traffic

except the traffic explicitly permitted by filter rules.

Packet filters are often used to secure perimeter networks and a filtering in transit networks is less often observed. With a client/server architecture and the client protected by a packet filter and the server not protected by a packet filter, the client can (if the rules set is not explicitly denying this outbound traffic) still initiate a connection to the server. If the server is protected by packet filter, it depends on the configuration of the filter if it is possible to initiate a connection. With decentralized applications where users are often located in perimeter networks, the impact of packet filters can increase depending on the restrictions imposed by the filtering appliance.



**Fig. 2.1:** Packet Filter Firewall Used as Gateway to Protect a Network Segment

Source: wikicommons / Harald Mühlböck 

#### 2.6.4 Network Address Translation

Network Address Translation (NAT) is a technique to modify network information of data transmitted in a network. It is commonly used to *masquerade* and replace (IP) addresses from an *internal* network with address information from an *external* network. A Network Address Translation (NAT) router replaces IP information in the network traffic and masquerades the internal network with an external address. For the masqueraded network, addresses from a private IP address space as defined with [RMK<sup>+</sup>96], can be used. When data are transmitted to an external network, the NAT device removes the internal IP address information and replaces it with the external IP address. When data are transmitted back, the NAT device replaces the external IP address with the internal IP address this data are destined for. To be able to replace IP addresses correctly, the NAT device has to maintain a connection table containing a mapping from the internal host communicating with an external host. A basic property of NAT is that data from an external host can only be forwarded to an internal host when a mapping for this connection exists in the NAT translation table or the NAT configuration. Hence computers in the external network cannot initiate a connection to a computer in the internal network without having the NAT device configured to forward connections properly or using NAT traversal or hole punching techniques.

NAT was initially defined in [SH99]. [SH99] only defines address translation on IP-level: only IP protocol information like IP address and IP header checksum are replaced but no higher layer protocol information. This is called one-to-one NAT and can be used to connect different IP networks.

### 2.6.4.1 Network Address Port Translation

To allow multiple computers, using higher level protocols like TCP or UDP, to share one external address provided by a NAT device, additional technologies are required. To avoid conflicts with these protocols in addition to IP information information from higher layer protocols (in particular port numbers) has to be replaced and the respective information has to be stored in the NAT translation table. This approach is called Network Address Port Translation (NAPT) and is defined in [SH99].

The use of NAT increased with the beginning exhaustion of IPv4 addresses in the mid-1990s: with increasing number of customers, ISPs had to provide more and more IPv4 addresses to their costumers. So ISPs started to assign dynamic IP addresses to their customers only valid for a certain period of time and provided only a single public IP address per customer. Due to this development, NAT is a widely deployed technique in end-user settings since it allows customers to have multiple devices share one Internet connection. [MSF11] states that in their dataset more than 90% of residential broadband networks use NAT gateways to connect to the Internet.

The behavior of NAT implementations depends on the respective implementation and varies between different devices and implementations. [MCK08] and [M13] give an overview how behavior of NAT devices can be categorized with respect to NAT traversal techniques. This work distinguishes between *port binding*, *NAT binding* and *endpoint filtering* classification of NAT behavior. [RWHM03], defining the Session Traversal Utilities for NAT (STUN) protocol used for NAT traversal, tries to classify different NAT approaches based on the possibility to establish connections to the devices located behind NAT devices. [RWHM03] specifies four main variants of NAT implementations, shown in Figure 2.2:

**Full Cone NAT:** Once a port mapping from an internal IP address and port to an external address and port is established with the NAT devices, all traffic from the internal address is sent through the external address.

Any external host can send data to the internal address by sending data to the external address. This behavior is depicted in Figure 2.2(a)

**Restricted Cone NAT** Once a port mapping from an internal IP address and port to an external address and port is established with the NAT devices, all traffic from the internal address is sent through the external address.

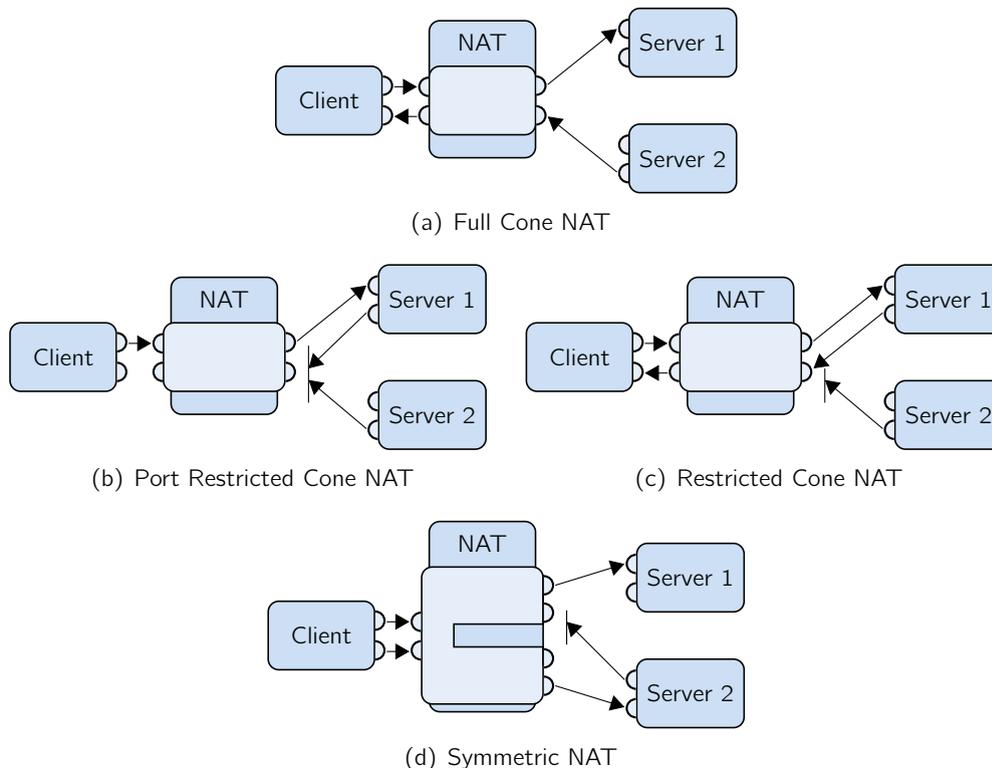
An external host can send data to the internal address by sending data to the external address from any source port if the internal host has sent data to the address of the external host before. This behavior is depicted in Figure 2.2(c)

**Port Restricted NAT** Once a port mapping from an internal IP address and port to an external address and port is established with the NAT devices, all traffic from the internal address is sent through the external address.

An external host can send data to the internal address by sending data to the external address from a specific source port if the internal host has sent data to the address of the external host and the source port before. This behavior is depicted in Figure 2.2(b)

**Symmetric NAT** Every connection from an internal address to an external host is mapped to a unique external address and port. If the same host sends data to a different destination, a different mapping is used.

An external host can only send data back when it received a packet from an internal host. This behavior is depicted in Figure 2.2(d)



**Fig. 2.2:** Classification of Different NAT Types According to [RWHM03]

Establishing connections with systems located behind NAT devices is challenging as we can already see with the four classes of NAT approaches defined in [RWHM03]. The use of NAT devices has major impact on end-to-end connectivity on the Internet and is one of main aspects limiting the possibility to establish connections with devices located in perimeter networks like ISP client access networks. Current networking applications have to put effort in trying to traverse NAT devices and several approaches like STUN [RMMW08], Traversal Using Relays around NAT (TURN) [MMR10, CNP11] and Interactive Connectivity Establishment (ICE) [Ros10] try to provide a systematic framework for NAT traversal.

#### 2.6.4.2 Carrier-Grade NAT

In addition to NAT being used in end-user settings to allow users to use their Internet connection with multiple devices, NAT is also used by ISPs affected by IPv4 address exhaustion. Many ISPs are only provided with a small pool of public IPv4 addresses and therefore cannot provide every customer with a public IPv4 address. An approach used by ISPs is the use of large-scale NAT (LSN) or Carrier-grade NAT (CGN) as defined in [RMK<sup>+</sup>96]: with Carrier-grade NAT (CGN), customers are not provided with a public IP address but instead a private IP address which is then translated by a NAT middlebox in the provider's network. This approach is illustrated in Figure 2.3(a).

With this approach, ISPs can re-utilize their address pools since they can provide customers with private IP addresses and can reserve their limited pool of public IPv4 addresses to be used by the CGN NAT box.

The use of CGN has several drawbacks as it breaks end-to-end connectivity and makes it impossible for hosts affected by CGN to host services. Since CGN devices have to maintain a state about mappings, CGN can introduce scalability and reliability issues.

### 2.6.4.3 NAT and IPv6

For IPv6, the successor of IPv4, the use of NAT was not intended nor specified [TZL10]. Due to the larger address pool provided with IPv6, address translation from private address ranges to public address as provided by NAT is not required anymore. Nonetheless, advocates of IPv6 and NAT emphasize the benefits an NAT implementation for IPv6 can provide: simplified and homogeneous network configuration and renumbering, concealment of network internals and a simple security mechanism. Therefore, an experimental standard for IPv6-to-IPv6 Network Prefix Translation (NPTv6) (or NAT66) [WB11] and implementations for stateful and stateless NAT66 exist<sup>3</sup>.

### 2.6.4.4 NAT and Connectivity on the Internet

As we can see with the analysis of NAT in this section, NAT can have major impact on connectivity for systems especially since it is well adopted and widely used. For servers used with client/server architectures running important and popular services it is in particular important that clients can connect. Therefore, these servers are often operated in networks where NAT is rarely used. NAT is widely used in perimeter networks like ISPs' client access networks or access networks of mobile phone providers. Here NAT is used by end users with NAT home devices to use their Internet access with multiple devices or on a large scale by ISPs using CGN to provide customers with IP addresses.

With a centralized architecture using a classic client/server approach, the impact of NAT is less severe since clients located behind NAT can initiate a connection to servers located on the core Internet. If the server is required to initiate a connection to a client located behind NAT, NAT devices often prevent connections to be established requiring the use of complex NAT traversal techniques not always working dependably.

The impact on decentralized networking applications, where every participant is required to initiate a connection to other participants, is even severe as described for the BitTorrent peer-to-peer system in [LP09]. In the context of this work, we can safely assume that a large number of users will be located in perimeter networks and therefore will be affected by NAT and NAT-imposed limitations to end-to-end connectivity. When NAT is used in end-user settings, advanced users having access to the NAT device may be able to circumvent NAT restrictions using Universal Plug and Play (UPnP) or port forwarding. Users affected by a large-scale NAT (LSN), as described with CGN have to use expensive traversal techniques like TURN, relaying traffic over a third party system.

## 2.6.5 DS-Lite

Another technology employed to counteract exhaustion of IPv4 addresses is Dual-Stack Lite (DS-Lite) as defined in [DDWL11]. Dual-Stack Lite (DS-Lite) is described as an IPv4-IPv6 transition technique but is more commonly used with ISPs which cannot provide an IPv4 address pool large enough to assign global IPv4 addresses to all of their customers. With DS-Lite, a customer's device is provided with a global IPv6 and a private IPv4 address from a private IPv4 network as defined in [WKD<sup>+</sup>12]. When the user attempts

<sup>3</sup> <http://sourceforge.net/projects/nfnat66/>

to access a host on the Internet using IPv4, the IPv4 packet is encapsulated in an IPv6 packet and sent to a CGN device operated by the ISP. This CGN device extracts the encapsulated IPv4 packet, replaces the private IPv4 address with a public IPv4 address and forwards the IPv4 packet to its destination. Since the CGN device maintains a port mapping, it is able to map the response back to the private IPv4 address and returns the response, encapsulated in an IPv6 message, to the client. IPv6 enabled services are accessed directly without using encapsulation in IPv4 and CGN. The use of DS-Lite is illustrated in Figure 2.3(b).

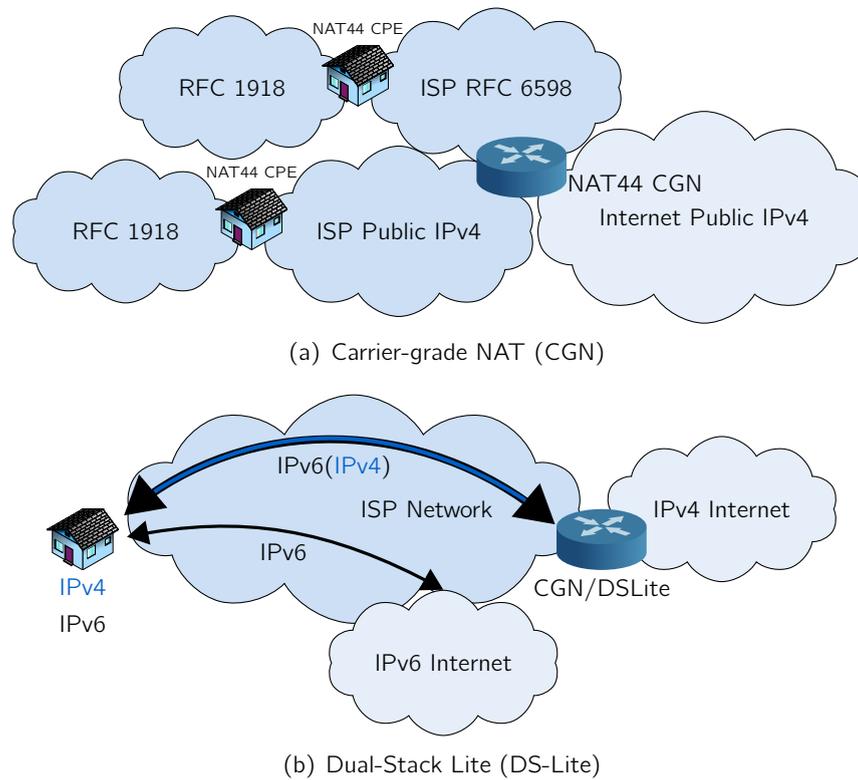
With DS-Lite, a connection to a user affected by DS-Lite can only be established when using IPv6. As described with IPv6 in Section 2.2.3 is IPv6 not yet widely deployed and only few ISPs provide IPv6 to their customers and only few companies, organizations use IPv6. Inbound connectivity for users affected by DS-Lite is therefore restricted to users also provided with IPv6. For their outbound connectivity using IPv4, the same limitations apply as with CGN. So for end-to-end connectivity with DS-Lite, we can assume similar impacts as with NAT: if we assume a classical client/server architecture where the clients initiate the connection to the server, the impact may be low since the client can establish connections with both IPv4 and IPv6 hosts on the Internet. If it is required to initiate a connection to a client, this can be achieved if the initiator has IPv6 capabilities. If the initiator is not able to use IPv6 but only IPv4, DS-Lite prevents a connection from being established.

For decentralized networking applications the impact is more severe, especially for end users: only a few ISPs provide IPv6 to their customers, most of them only provide IPv4 in combination with NAT or CGN. So most end users affected by DS-Lite cannot initiate a connection to these IPv4-only hosts. On the contrary, IPv4-only hosts cannot initiate a connection to users affected by DS-Lite, since users affected by DS-Lite are only provided with a private IPv4 address masqueraded by the CGN appliance.

### 2.6.6 Proxy Servers

Proxy servers are computers acting as intermediaries and gateways in networks. Proxy servers can be implemented on different layers of the network model but are based on the idea to act as an intermediary between the client and the destination of the traffic. Proxy servers can be used for various different purposes like increasing performance by caching data, to filter or monitor traffic, to hide the internal structure of a network for security purposes or to provide anonymous communication by hiding communication participants. With proxy servers, clients initiating a connection do not connect to the destination directly, but instead issue the request to the proxy, which initiates a connection to the target, obtains the desired resources and passes the data to the client. A system retrieving resources for clients from the Internet is commonly called a *forward* proxy, whereas a *reverse proxy* is used to access a resource on an internal network.

Proxy servers can be used to separate networks and provide only controlled communication between networks. This approach is often used in combination with filtering techniques like firewalls described in Section 2.6.3. Communication with a network is limited by using restrictive filtering and is restricted to communication using an enforced proxy server. With this approach communication is often enforced to be unidirectional, so only clients from an internal network can access the Internet using the proxy server whereas it is impossible to establish communication from outside networks with hosts protected by a proxy server. This leads to violations of the end-to-end principle since connections to clients located in networks using proxy servers cannot be established.



**Fig. 2.3:** IPv6 Transition Mechanisms: Carrier-grade NAT (CGN) and Dual-Stack Lite (DS-Lite)

## 2.7 Centralized Client/Server Architectures

The classical design approach for a networking application is the client/server approach. With this approach two different roles exist within a networking application: the server, providing services, storing data and performing computations and the client, consuming services from the server as shown in Figure 2.4(a). This architectural approach originates from the early times of the Internet when computations were performed by big, expensive mainframe computers. Many users facilitated these mainframe servers to perform computations and accessed them from remote using simple, cheap and not very powerful terminals or clients.

This architectural design concept survived the evolution of the Internet and is still the most commonly used design for networking applications. With a centralized client/server architecture, a networking application is running on a centralized server. This approach has advantages for developers and administrators since with a centralized architecture a networking server application (conceptually) runs as a single application on a single server. This simplifies the development, debugging, control and maintenance of such an application. Economically, the application can be accessed as a black box on the server, so control over the application can be assured.

Much effort was put in reducing the impact of a centralized architecture providing a centralized single point of failure: redundant architectures, load balancing, virtualization or fail-over mechanisms to just name a few. To improve performance and reduce latency for the user, redundant servers placed all over the network geographically close to the user and so-called Content Distribution Networks (CDNs) are used. CDNs are distributed networks with a large number of servers located in different data centers around the world interconnected with multiple backbone connections. CDNs are used to provide high

performance and high availability services like streaming or application services to end users. Therefore, CDNs are often directly connected to ISP networks to minimize latency and network distance to users.

From a user's perspective, centralized architectures have drawbacks, since the user has no control over the services and applications he uses. Centralized architectures are operated by a provider and users depend on these providers to continue their services and do not change the terms of usage. Since for users a service is a black box, they have to trust the providers to operate their services reliably and trustworthy. This aspect is in particular relevant when users give their private data to such a centralized services. With respect to concerns about privacy, it is a questionable approach to store user-related, private data on a central server.

## 2.8 Decentralized Peer-to-Peer Networking Architectures

Today computers are not the simple terminals anymore and owning a computer is not the privilege of companies or research institutions. With modern computing devices, most computers, mobile phones and other personal devices have enough resources to perform calculations directly on the users' devices and do not require centralized servers to perform expensive computations. Being almost continuously connected to the Internet, these devices can communicate with each other directly not requiring an intermediary server. This allows to realize networking applications in a decentralized manner no longer requiring centralized services.

The peer-to-peer paradigm states that in a peer-to-peer network participants both provide services and consume services from other participants. No dedicated servers (only providing services) and no dedicated clients (only consuming services) exist. These roles are existing and combined in every participant of the network, called a *peer*. Peers communicate directly with each other on transport layer and do not require a centralized instance coordinating communication in the network. Since peer-to-peer networks define an abstraction over the existing Internet structure with peers connected to each other using an application-specific addressing scheme, these networks are often referred to as *overlay networks* or just *overlays*. An illustration for the structure of a peer-to-peer network is given in Figure 2.4(b).

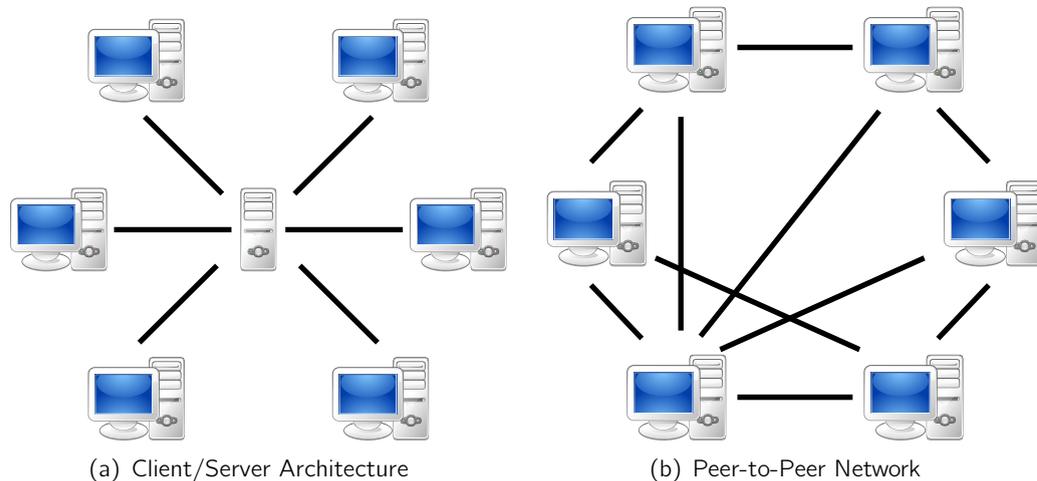
Contrary to a client/server architecture, where the server is supposed to be continuously running over a long time, participants in a peer-to-peer network can join and leave frequently. This effect is called *churn* and requires special attention when developing such a network application.

### 2.8.1 Structured and Unstructured Peer-to-Peer Architectures

Within the class of peer-to-peer networks, different architectures exist:

In *unstructured* peer-to-peer networks, no further structure or hierarchy exists. All participants connect randomly and establish a direct connection with each other. These types of networks are easier to establish, but issues arise from this form of architecture when operations in the network are performed. Due to missing structure of the network, locating information in the peer-to-peer network can be expensive. To locate information in the network, this information may have to be flooded through the network what can cause high CPU load, memory consumption or network load and can be inefficient.

In *structured* networks, the overlay is organized in a topology, where network operations can be performed more efficiently. The most common way to structure a peer-to-peer



**Fig. 2.4:** Comparison of Client/Server and Peer-to-Peer Architectures

Source: wikicommons 

overlay network is to use an overlay addressing scheme as provided by a Distributed Hash Table (DHT).

### 2.8.2 Distributed Hash Tables

Exchanging information between peers in a decentralized system with peer joining and leaving the network without prior coordination, requires a new way to store and retrieve data in a peer-to-peer network. For this reason peer-to-peer applications can use a Distributed Hash Table (DHT). A DHT is a key/value store distributed in the network. DHTs provide a cyclic key address space and each participating peer is assigned a portion of this address space, based on an identifier assigned to peers. A DHT basically offers two operations: a put operation to store a value under a key, and a get operation to obtain a value stored under a key. When a peer is storing a value in the DHT, the peer determines the peer responsible to store this value based on the key the data to store under. The peer sends the data to the responsible peer which stores the data locally. To retrieve a value, a peer again finds the peer responsible for the key and retrieves the value from this peer.

Attacks to disturb the functioning of DHTs exist as described in [SM02]: *Sybil attacks* [MDD02], where an attacker creates a large number of pseudonymous identities to gain disproportionately high influence in the DHT overlay and *eclipse attacks* [SNDW06], where an attacker tries force a peer to peer only with a set of malicious peers by creating selective peer identifiers to surround the victim in the DHT namespace and mediate most or all communication to and from the victim.

Some existing DHT implementations like R<sup>5</sup>N [Eva11] or X-Vine [MCB11] put special focus on restricted communication and churn to improve censorship-resistance by adding redundancy for stored values and randomized routing.

### 2.8.3 The GUNet Peer-to-Peer Framework

GNUet<sup>4</sup> envisions to be the foundation for future decentralized Internet protocols. It is intended to be a versatile platform supporting many different forms of peer-to-peer applications. GNUet is designed with a focus on security, resilience and in particular respects privacy. It is designed to be fully decentralized and does not rely on any centralized services. GNUet tries to provide secure communication between peers by authenticating communication partners. Confidentiality is achieved by ensuring only senders and recipients of messages know about the content of messages strongly relying on link encryption and plausible deniability of traffic exchanged between peers.

The goal of the GNUet project is to create a framework for peer-to-peer applications. GNUet is a free software project and wants to enable developers to easily realize their idea of a new decentralized peer-to-peer application without having to reinvent the wheel. GNUet provides developers with important building blocks required to implement peer-to-peer applications. GNUet provides functionality to establish low-level connectivity between peers, bootstrapping mechanisms, encrypted communication between peers and a DHT implementation for the developers. Based on these building blocks, developers can concentrate on developing the functionality required for their applications. GNUet also provides applications like a file sharing and a telephony application.

In GNUet each node participating in the network is called a *peer*. Each peer is equipped with a cryptographic public/private EdDSA key pair and is identified by the public key, the so-called *peer identity*. EdDSA is a digital signature scheme using Elliptic Curve Cryptography (ECC) [Res00] and Curve25519 described in [BDL+12]. The EdDSA key pair is used to mutually authenticate peers. The peer identity can be represented as a human-readable string using the Crockford base32 [Cro, Jos06] encoded hash of the public key. Mutual authenticated and encrypted communication between peers is realized using Elliptic Curve Diffie Hellman Ephemeral (ECDHE) and the derived ephemeral keys are signed using EdDSA. With ECDHE a shared secret is exchanged between peers and is used to create a pair of sessions keys using the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) key derivation function described in [KE10]. This derived key pair is then used to encrypt communication between peers.

GNUet is written mostly in C and has a strong focus on portability to support a large number of platforms and operating systems. It is realized as a multiprocess architecture encapsulating different functionalities in different (system) processes interacting using Inter-Process Communication (IPC) communication. Using multiple processes improves performance on modern multi-core processors compared to realizing all GNUet components in a single process and increases maintainability and reliability by providing fault isolation between components. By splitting functionality in separate components depending on each other, it creates a layered service architecture. In GNUet, functional components running in separate processes are called *services*. Each service provides an Application Programming Interface (API) for other components to use this component. These APIs are implemented as a shared library. Components can link against this library to use this API and the functionality provided by the service. The service API runs in the process context of the client and interacts with the service using IPC. GNUet's layered service architecture based on services is depicted in Figure 2.5. GNUet's services are controlled by a master service, the Automatic Restart Manager (ARM) starting and stopping services on demand and after a potential crash. GNUet supports multi user environments by distinguishing between *system* services, shared between all users on a system and *user*

---

<sup>4</sup> <https://gnunet.org>

services, providing services for a designated user. In addition to extending functionality by adding services, a second functionality of GUNet is the use of a plugin architecture to extend components with additional plugins. This functionality is for example used with the TRANSPORT service described in Section 4 to extend the transport infrastructure with new transport protocols implemented as loadable plugins.

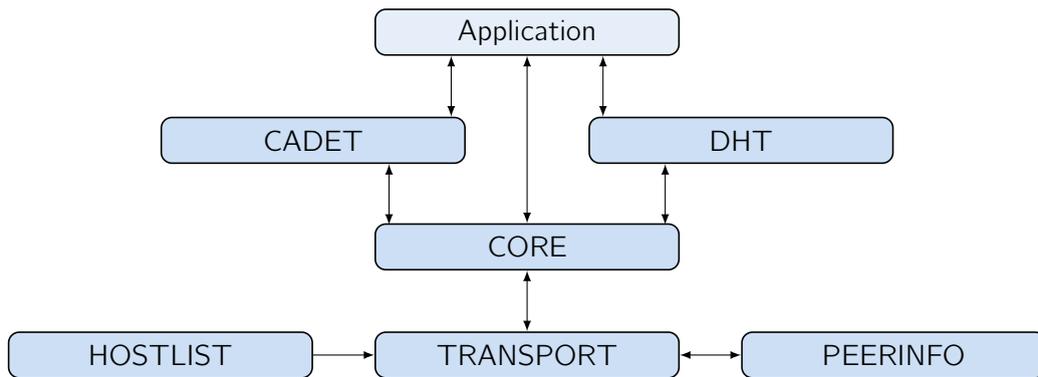


Fig. 2.5: GUNet's Layered Service Architecture

## 2.9 Conclusion and Findings

In this section we provided an overview over the design and architecture of the Internet and introduced important protocols and technologies used on the Internet. We highlighted that on today's Internet free and unrestricted communication between users and systems is limited and restricted for various reasons and pointed out the limitations and restrictions current technical approaches impose on free and unrestricted communication between users on the Internet.

The Internet was originally envisioned as a network to interconnect systems, providing connectivity between systems and give users the possibility to communicate with each other. The paradigm of end-to-end connectivity originally envisioned a *dumb* network only forwarding data between hosts on the Internet with functionality realized in end systems and not the network itself. But on today's Internet a diverse set of middlebox appliances exist inspecting, managing, transforming, shaping and filtering traffic for various purposes. End-to-end connectivity between systems on the Internet is limited for various technical or security reasons by firewalls, packet filters, enforced proxies, NAT appliances, all of them diminishing the possibility for users affected by such approaches to establish connections and communicate with other systems on the Internet.

Decentralized peer-to-peer systems are in particular affected by limitations to end-to-end connectivity as we can assume that a larger number of users of a peer-to-peer system will be located in restricted networking environments like ISP access networks. Providers, network operators and other parties are often suspicious of peer-to-peer traffic and peer-to-peer traffic is often discriminated, affected by network traffic management or prohibited.

A communication infrastructure for decentralized peer-to-peer systems trying to re-establish free and unrestricted communication between users and employing existing communication infrastructure to do so has to be aware of these limitations and be able to cope with restrictions for end-to-end connectivity and antagonize degradation attempts. To establish an overlay network re-establishing the possibility of end-to-end communica-

tion between participants therefore has to focus on increasing connectivity for users in restricted environments, antagonizing degradation attempts and improve overlay connectivity between participants by providing the possibility to route traffic between participants in the peer-to-peer overlay.

### 3. RESILIENCE OF COMMUNICATION ON THE INTERNET

Direct communication between systems on the Internet can be limited for various reasons as we saw with the limitations imposed by middleboxes described in Section 2.6. A peer-to-peer system can circumvent these restrictions to direct end-to-end connectivity by routing traffic to other participants via the peer-to-peer overlay. But before we can design and present a secure and resilient communication infrastructure using the Internet as a foundation, we have to analyze if the Internet can always provide a communication path to other systems and how resilient the Internet itself is against the failure of communication providers and communication links.

In this chapter we analyze how resilient the Internet's communication infrastructure is against failing network infrastructure and how hard it is for an attacker to partition the Internet and separate users from important services and impact the possibility to communicate with these users by routing traffic via the peer-to-peer overlay. For this evaluation we present a new graph partitioning algorithm and use this algorithm to help answer the question of how resilient the Internet is against Byzantine failure of a small set of providers (nodes) or peering links between providers (edges).

To answer this question we analyze the topology of the Internet's backbone by generating graphs from Internet measurement data and use a new heuristic to find the smallest possible set of networks or connections to be removed to split the Internet. We present empirical evidence that the Internet's backbone infrastructure is surprisingly well connected.

Work presented in this chapter is researched together with Christian Grothoff and Ramakrishna Thurimella and was previously published in [WGT12].

#### 3.1 Introduction

Internet connectivity can be surprisingly brittle with respect to failures at physical locations. There are several examples where failures at a single physical location have had a significant impact on connectivity for a whole region or country [Hac09, Par12, Sau08].

The modern Internet consists of Autonomous System (ASes) which are linked together using the Border Gateway Protocol (BGP). A high-level approximation for the Internet topology is thus the AS graph, which models the peering relationships between ASes.

We use a graph-theoretic approach by finding sets of nodes (networks) and edges (connections), the so called separators, to be removed from the network graph. A *separator* is a set of edges or nodes that partitions a graph into sizable connected components. We use the term separator to mean both edge separator and node separator; it will be clear from the context which type of separator is used.

The primary goal of the analysis in this chapter is to characterize the size of the separator that would have to be removed to fragment the AS graph and thereby the Internet. Determining separators for network topologies is useful in various respects, in particular:

- Elements of the separator represent critical infrastructure since the removal of them

would fragment the network and significantly reduce its value. Networks with large separators have higher resilience.

- Elements of the separator are particularly useful for the placement of traffic monitoring (or even manipulation) equipment since a large fraction of long-distance communications must cross those links.
- Elements of the separator are particularly critical for routing in a peer-to-peer network. Peers being elements of a network separator can limit the possibility to route traffic to other participants in the peer-to-peer overlay.

Any separator has to balance two goals: both the size of the separator as well as the size of the largest resulting component should be small. We refer to the size of the largest remaining connected component in relation to the overall size of the graph as  $\chi$ . For this work, we assume that a value for  $\chi$  is given and that the goal is to find a small  $\chi$ -separator.

Using the AS graph creates the problem of obtaining a sufficiently accurate approximation of the actual graph. We address this problem by using two methods to obtain approximations of the AS graph. We then demonstrate that the differences between these two approximations are small (in terms of the size of the separator in relation to the size of the AS graph). We also consider the problem that not all ASes are of equal relevance. By mapping page access statistics from the Alexa Web Information Service<sup>1</sup> to ASes, we obtain a weighted AS graph which we then try to separate as well.

### 3.2 Background and Related Work

Given a graph with  $n$  nodes of total weight  $W$ , a  $\chi$ -separator is a subset of nodes or edges whose deletion partitions a connected graph into connected components where the largest component has no more than  $\chi \cdot W$  nodes, for some fixed  $1/2 \leq \chi < 1$ . In other words,  $\chi$  determines how *balanced* a separator is. In a connected graph of  $n$  nodes and  $m$  edges, one can exhaustively delete every separator of size  $k$ , examine the connected components for balance (e.g. size of the largest connected component as a fraction of the original graph). This gives us an  $O((m+n)\binom{n}{k})$  time algorithm.

For fixed  $k$ , this is a polynomial-time algorithm; otherwise, this algorithm has  $\Omega(n^k)$  complexity. It should be noted that even for constant values of  $k > 3$ , this brute-force algorithm is impractical. Furthermore, if there is a polynomial time algorithm for finding a  $\chi$  separator of size  $k$ , then that algorithm can be used to find a  $k$ -clique in a given graph [Mar06]. In light of this, one can consider approximation algorithms that relax the problem constraints: increase the size of the separator for a given  $\chi$ , or tolerate a slightly less balanced separator in favor of a smaller separator.

Historically, Kernighan and Lin recognized the importance of the graph partitioning problem and its various applications as far back as in 1970 [KL70]. In their landmark paper, they also proposed a heuristic which has served as a benchmark for other graph partitioning heuristics ever since.

Another common class of partitioning heuristics are derived from the technique of graph coarsening: the graph to be partitioned is made smaller by collapsing nodes and edges, the resulting smaller graph is partitioned, and the separator in this smaller graph is extended to the original graph during the uncoarsening step [BJ93, HL95]. Though it

<sup>1</sup> <http://aws.amazon.com/alexa/>

was apparent that these early works were promising, it remained unclear whether these heuristics would consistently produce good quality separators for graphs arising in a wide range of application domains until Karypis and Kumar [KK98] explored the partitioning problem systematically for various versions of the problem. They proposed refinements to the coarsening heuristic and implemented them. Their implementation, which they distribute under the name Metis, is available for free<sup>2</sup>. A remarkable feature of Metis is its fast execution time and its applicability to graphs arising out of varied domains. We compare our results against those produced by Metis.

Kleinberg [KSS08] defined  $(\epsilon, k)$ -failures and proposed an algorithm to monitor the connectivity of a network against such failures. These failures are events in which an adversary deletes up to  $k$  nodes or edges, after which there are two sets of nodes  $A$  and  $B$ , each at least an  $\epsilon$  fraction of the network, that are disconnected from one another. A set of nodes is an  $(\epsilon, k)$ -detection set  $D$  if, for any  $(\epsilon, k)$ -failure of the network, some two nodes in  $D$  are no longer able to communicate. Finding detection sets, though the problem appears superficially related to the problem we consider in this paper, is simpler than finding good separators.

Other related works include finding approximate separators by Feige and Mahdian [FM06], polynomial time approximation schemes for finding most balanced minimum separators that separate two designated vertices  $s$  and  $t$  [Bon10], and finding weak points in networks [BLS10] using semi-definite programming. The last paper is a study on the impact on node and edge failures on connectivity that is quite similar to the one presented in this work. The main difference is that they start with a network where nodes are categorized into clients and servers and where any server is able to provide the required service. They present algorithms to calculate lower and upper bounds on the minimum connectivity (after failures). The results presented in [BLS10] work for graphs that are typically orders of magnitude smaller than those presented in this paper.

### 3.3 Calculating Separators

This section describes the heuristics we use to find node and edge separators in the network graphs. We initially attempted to find separators using Metis [KK98]; however, Metis did not produce good cuts for larger values of  $\chi$  and for weighted graphs the resulting separators were quite large. We believe this is mainly because Metis is not optimized to create  $\chi$ -separators with  $\chi > \frac{3}{4}$ . The separators calculated using our heuristics are sometimes significantly smaller than those computed by Metis, justifying the development of our own heuristics.

#### 3.3.1 Finding Edge Separators

We found that minor variations of the well-known Kernighan-Lin (KL) heuristic for finding edge separators [KL70] give us the best results for the network graphs considered in this paper. Define  $S(A, B)$ , the separator for a given vertex partition  $(A, B)$ , to be the set of edges which have one end point in  $A$  and the other in  $B$ . We then intend to minimize the size of the separator,  $|S(A, B)|$ . The KL heuristic tries to achieve this by starting with an arbitrary partition and applying successive vertex swaps until no amount of swapping helps, i.e. when the heuristic reaches a local minimum. A feasible starting solution is created using a breadth-first traversal from a random starting node. To describe the algorithm, we need the following definition. When a pair of vertices  $a \in A, b \in B$  is swapped, we say

<sup>2</sup> <http://glaros.dtc.umn.edu/gkhome/views/metis>

it results in  $gain(a, b) = |S(A, B)| - |S(A \cup \{b\} - \{a\}, B \cup \{a\} - \{b\})|$ . Note that the gain is negative if swapping  $a$  with  $b$  increases the size of the separator.

When we are seeking to partition the input graph into *unequal* parts (i.e.  $\chi > \frac{1}{2}$ ), we employ the trick suggested by Kernighan-Lin [KL70] of adding an appropriate number isolated “dummy” nodes to the graph.

### 3.3.2 Edge Separators for Weighted Graphs

Let us now consider graphs with node weights where weights model the relative importance of different nodes. Clearly for this version of the problem, dummy nodes cannot be used effectively. Let  $W_A$  represent  $\sum_{v \in A} w_v$  where  $w_v$  is the weight of vertex  $v$ . For brevity

```

1 Alg.: Weighted Edge Separator
   Input:  $G(V, E)$  with node weights, a  $\chi$  partition  $A$  and  $B$ .
   Output: Updated  $\chi$  partition  $A, B$  with potentially fewer edges between  $A$  and  $B$ 
2 Coarsen  $G$  by merging  $u$  with its neighbor  $v$  if  $degree(u)$  is 1 and  $w_u + w_v < t$ ;
3 Unmark all nodes;
4  $max\_gain \leftarrow 0$  ;
5  $cumulative\_gain \leftarrow 0$ ;
6 while unmarked nodes exist in  $A$  and  $B$  do
7    $g_A \leftarrow$  gain from best feasible move from  $A$  ;
8    $g_B \leftarrow$  gain from best feasible move from  $B$  ;
9    $g_{AB} \leftarrow$  gain from best feasible swap ;
   // only unmarked nodes are feasible
10   $g \leftarrow \max(g_A, g_B, g_{AB})$  ;
11   $cumulative\_gain \leftarrow cumulative\_gain + g$ ;
12  switch  $g$  do
13    case  $g_A$ 
14      perform best feasible move from  $A$  ;
15      mark node;
16    end case
17    case  $g_B$ 
18      perform best feasible move from  $B$  ;
19      mark node;
20    end case
21    case  $g_{AB}$ 
22      perform best feasible swap ;
23      mark nodes;
24    end case
25  endsw
26  if  $cumulative\_gain > max\_gain$  then
27     $max\_gain \leftarrow cumulative\_gain$  ;
28    checkpoint steps taken so far ;
29  end if
30 end while
31 undo the steps taken after the last checkpoint ;
32 return  $A, B$ ;

```

the total weight of the graph  $W_V$  is denoted simply as  $W$ . We will refer to  $\chi \cdot W$  as the *threshold*. A  $\chi$  partition,  $\chi > \frac{1}{2}$ , of the node set  $V$  is a partition into two sets  $A$  and  $B$  where both  $W_A$  and  $W_B$  are below the threshold. Note that a  $\chi$  partition always exists as long as the weight of every individual node is under the threshold.

For weighted graphs, we make three significant changes to the KL heuristic. First, we use a simplistic variant of graph partitioning [KK98] where we iteratively merge all nodes of degree one with their neighbor as long as the weight of the resulting node is under the threshold. Especially for some of our larger and sparser graphs, this results in a significant reduction in the problem size without impacting the quality of the solution.

Second, in order to deal with weighted nodes, we introduce the notion of a *feasible* swap; a swap is feasible if after the swap both sides are below the threshold. Only feasible swaps are considered in our innermost loop of Kernighan-Lin.

Our third change is that in addition to swaps, we consider (feasible) *moves*, where a single node is moved from one side to the other side (and both sides are below the threshold after the swap).

The algorithm is summarized in Algorithm “Weighted Edge Separator” (Alg. 1). The initial  $\chi$  partition to our algorithm is found as in the unweighted case. One key implementation technique that we employed is ordering the nodes in the decreasing order of gain and searching on this ordered lists. This gives us the ability to *abort* the search the moment we discover that the gain at the current indices is less than what we already know is possible from the previous considerations. From this point on, none of the remaining possibilities will offer a better gain. Since searching takes place repeatedly in the innermost loop of the heuristic, this simple observation results in significant savings in the run time of the algorithm.

### 3.3.3 Finding Node Separators

Next we consider *node* separators. The discussion given in this section applies to both weighted and unweighted graphs. As in the previous section, for weighted graphs we only consider *feasible* moves or swaps.

A  $\chi$ -node separator  $C$  is a subset of the node set  $V$  such that the weight of every connected component in the subgraph induced by  $V - C$  is under the threshold. The size of the separator is  $|C|$  and it is this size we seek to minimize. It is important to note that we only consider the cardinality of  $C$  and not the sum of the weights of the nodes in  $C$ . Note that if we have an edge separator of size  $k$ , then we can easily construct a node separator of size *at most*  $k$  by picking, arbitrarily, either  $u$  or  $v$  for each edge  $(u, v)$  from the edge separator.

Once we have an initial node separator, we employ simulated annealing to improve the quality of the node separator as shown in Algorithm “Weighted Node Separator” (Alg. 2). For a node  $u$  in the cut  $C$ , let  $N_A(u)$  and  $N_B(u)$  represent the neighbors of  $u$  in  $A$  and  $B$  respectively. In each iteration, we consider two operations for a vertex  $u$  from the cut: push  $u$  and pull  $u$ . Push  $u$  removes  $u$  out  $C$  and places it in  $A$ . In addition, this operation moves  $N_B(u)$  from  $B$  to  $C$  in order to maintain the invariant that  $C$  is a node cut. Pull  $u$ , on the other hand, removes  $u$  out  $C$  and places it in  $B$ . Also  $N_A(u)$  is moved from  $A$  to  $C$  in order to maintain the invariant that  $C$  is a node cut. Clearly, push (resp. pull)  $u$  reduces the size  $C$  by one if  $N_B(u) = \emptyset$  (resp.  $N_A(u) = \emptyset$ ). If  $N_B(u) \neq \emptyset$ , a push  $u$  operation increases the node cut size by  $N_B(u) - 1$ . The cost of a move, either a pull or a push, is the net change in the number of nodes of  $C$ . As before, these moves are performed only if they are feasible, i.e. after the move both  $A$  and  $B$  are below

```

1 Alg.: Weighted Node Separator
   Input:  $G(V, E)$  with node weights, a  $\chi$  node partition  $C$  derived from a good
           quality edge separator.
   Output: Updated  $\chi$  partition  $A, B$  with potentially fewer edges between  $A$  and  $B$ 
2  $current\_best \leftarrow$  cost of  $C$ ;
3 for  $h\_max \leftarrow 2$  to  $30$  do
4   for  $heat \leftarrow h\_max$  down to  $1$  do
5      $c_a \leftarrow$  Best node separator cost from
6       20 iterations of Push( $C, heat$ );
7      $c_b \leftarrow$  Best node separator cost from
8       20 iterations of Pull( $C, heat$ );
9     // Pull is the same as Push, only with  $A$  and  $B$  exchanged.
10    if  $current\_best \geq \min(c_a, c_b)$  then
11       $current\_best \leftarrow \min(c_a, c_b)$ ;
12       $C \leftarrow$  node separator corresponding
13        to  $\min(c_a, c_b)$ ;
14    end if
15  end for
16  if no change in  $C$  in the last 5 iterations then
17    return  $C$ ;
18  end if
19 end for
20 return  $C$ ;

```

the threshold. With simulated annealing, lower-cost moves are performed with higher probability, with the general chance of success being determined by the current amount of heat. The problem with this approach is that if we were to use simulated annealing in the usual fashion — slowly cooling down from ‘infinite’ heat, the initial moves would destroy the good properties possessed by our initial node cut which was derived from a good edge cut.

### 3.4 Graph Generation

For our analysis, we generate graphs representing the Internet from actual measurement data. All measurement data we use represent the state of the Internet in late 2010 or early 2011. Section 3.4.1 describes how we use Border Gateway Protocol (BGP) [RLH06] routing information gathered by the Route Views project<sup>3</sup> to generate a graph representing peering relationships between ASes. Naturally, the resulting graph is only a rough approximation of the actual relationships since the data provide only a local view of the routing topology and do not model policy restrictions that may be present. In Section 3.4.2 we try to compensate for this by combining the BGP generated AS graph with an AS graph based on IP-level forward-path measurement topology information provided by CAIDA<sup>4</sup>.

Finally, we want to differentiate between networks that provide “important” services and networks that are not widely used. For example, AS 56357 is currently an AS for research consisting only of a single router — clearly this network is hardly significant to

<sup>3</sup> <http://www.routeviews.org/>

<sup>4</sup> <http://www.caida.org/>

```

1 Alg.: Push( $C$ , heat)
   Input: Weighted graph  $G = (V, E)$ , node separator  $C$ , two partitions  $A$  and  $B$  of
            $V - C$ , and a positive integer heat
   Output: Updated  $A, B, C$  after possibly moving some nodes  $u$  from  $C$  to  $A$  and
           some neighbors  $u$  from  $B$  to  $C$ .
2  $current\_cost \leftarrow$  cost of  $C$ ;
3 foreach  $u \in C$  do
4   Let  $N_B$  denote  $\{v \mid v \text{ is a neighbor of } u \text{ in } B\}$ ;
5   if  $weight \text{ of } (A \cup \{u\}) \leq \chi \cdot W$  then
6      $new\_cost \leftarrow$  cost of  $(C \cup N_B - \{u\})$ ;
7      $\delta \leftarrow new\_cost - current\_cost$ ;
8      $r \leftarrow$  random number between 0 and  $\delta$ ;
9     //  $r$  can be negative if  $\delta < 0$ 
10    if  $r \leq heat$  then
11      move  $u$  from  $C$  to  $A$ ;
12      move  $N_B$  from  $B$  to  $C$ ;
13       $current\_cost \leftarrow new\_cost$ ;
14    end if
15  end if
16 end foreach

```

**Tab. 3.1:** Characterization of the AS Graphs Generated Using Route View's BGP Snapshot

Monitor Name	LINX	SYDNEY	WIDE
Prefixes	8,414,813	1,543,223	680,980
Nodes	35,872	36,543	36,315
Edges	75,170	67,485	52,885

the Internet as a whole. Section 3.4.4 describes how we use HTTP access statistics as one possible method for evaluating network relevance and assigning weights to the nodes in the graphs.

### 3.4.1 Construction of AS Graphs from BGP Routing Information

We build a first set of AS graphs using the AS routing information collected by the Route Views project. The Route Views project collects BGP information by operating 10 BGP routers in different locations. The routers' routing tables and the received BGP updates are written to disk and made publicly available on a daily basis.

We use the path information from BGP for our graph generation by adding occurring ASes to our graph as nodes. For consecutive ASes in the AS path we add an edge to the graph.

For this work we initially selected three different BGP datasets from three different BGP routers to investigate which impact the number of announced IP prefixes in the routing tables on the resulting graph has: The LINX dataset (London, UK) contained over 8 million announced prefixes, the average size SYDNEY dataset (Sydney, Australia)

**Tab. 3.2:** Characterization of the AS Graphs Generated Using CAIDA's Routed AS Links Dataset

Dataset	Cycle 1249	Cycle 1248	Cycle 1250
AS Sets	63	54	63
MOAS	1,456	1,455	1,263
Nodes	19,376	19,416	19,343
Edges	43,654	44,371	42,273

contains 1.5 million prefixes and the very small WIDE dataset only about 680,000 prefixes all from December 30th 2010. The resulting graphs have almost the same number of nodes with around 36,000 nodes. The number of edges differs, depending if the router is located in a well-connected area of the core Internet or if it is located in a peripheral area. When we merge the resulting graphs, we obtain a graph with 36,697 nodes and 79,464 edges. Based on this modest increase in the graph size from merging multiple points of view, we expect that using a few more vantage points would not add a significant number of additional nodes or edges. Still the resulting AS graph is incomplete; alternative methods, such as the one discussed in the next section, can be used to discover some of the missing edges, such as those that typically are not widely advertised via BGP. Table 3.1 provides some further statistics on the resulting AS graphs.

### 3.4.2 Construction of AS Graphs from Traceroutes

In addition to the graphs from section 3.4.1, we generate AS graphs based on IP-level forward path measurements conducted by CAIDA. CAIDA performs regular path measurements to every routed /24 IP network. Based on this measurements CAIDA provides the "IPv4 Routed /24 AS Links Dataset" [HHA<sup>+</sup>] where they mapped the contained IPs to ASes using Route Views BGP data and extracted the connections between the ASes. This dataset contains pairs of connected ASes and additional information about the processing process. In particular, the data distinguishes between direct and indirect links. Links are marked as indirect if one or more hops on the IP-path cannot be mapped to an AS.

Table 3.2 characterizes the AS graphs generated using the "IPv4 Routed /24 AS Links Dataset". Combining the AS graphs from the three datasets results in an AS graph with 22,271 nodes and 57,867 edges.

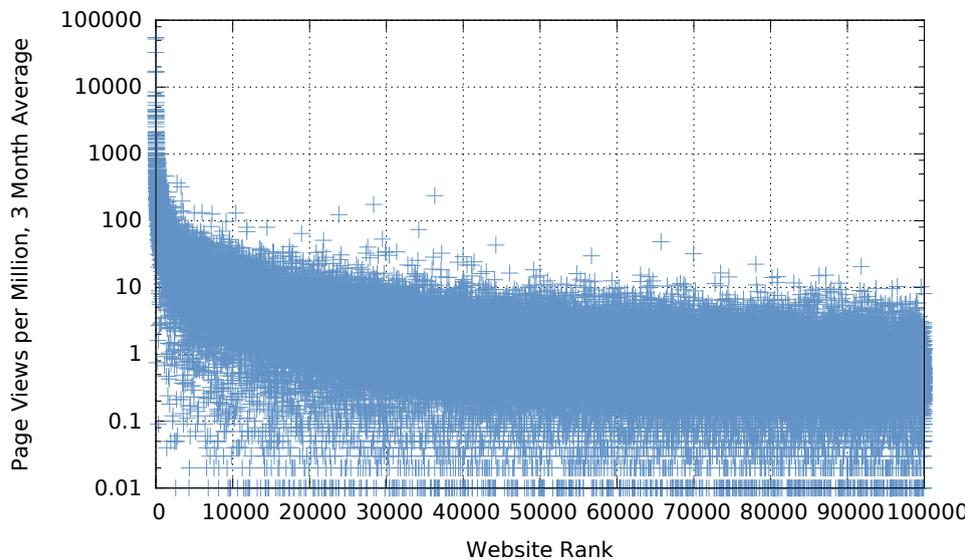
### 3.4.3 Merge of BGP and Traceroute Graphs

To obtain a best possible graph representation of the Internet, we merged all the graphs mentioned before into one big graph. So we can prevent the drawbacks from both graphs and get a graph with a non-local, non-directed view on the Internet.

We merged the BGP-based graphs and the CAIDA AS links based graphs in an resulting graph containing 36,715 nodes and 99,852 edges.

### 3.4.4 Weight Generation

We also want to be able to assign a weight to each AS reflecting its importance, where importance is based on the popularity of the service the AS provides to end-users. Ideally,



**Fig. 3.1:** Weight Distribution for Websites in Alexa's Top 100,000 Websites

the importance rating would reflect the impact of separating the AS from the rest of the network. As a simple approximation of actual importance, we use the popularity of hosted websites as an indicator for the importance of networks. As a basis for the popularity of websites we used a ranked list of websites provided by the Alexa Web Information Service. We obtained a list of the current top 100,000 websites (hostnames) ranked by popularity, including the relative number of page views per million page views for each of those sites. This page view rank in relation to the rank of the website in the Alexa ranking is depicted in Figure 3.1. The plot is this wide because Alexa ranks the top 100,000 websites not only by the number of page views but also the number of distinct users accessing the pages (“reach”). As a result, higher ranked websites do not always have more page views.

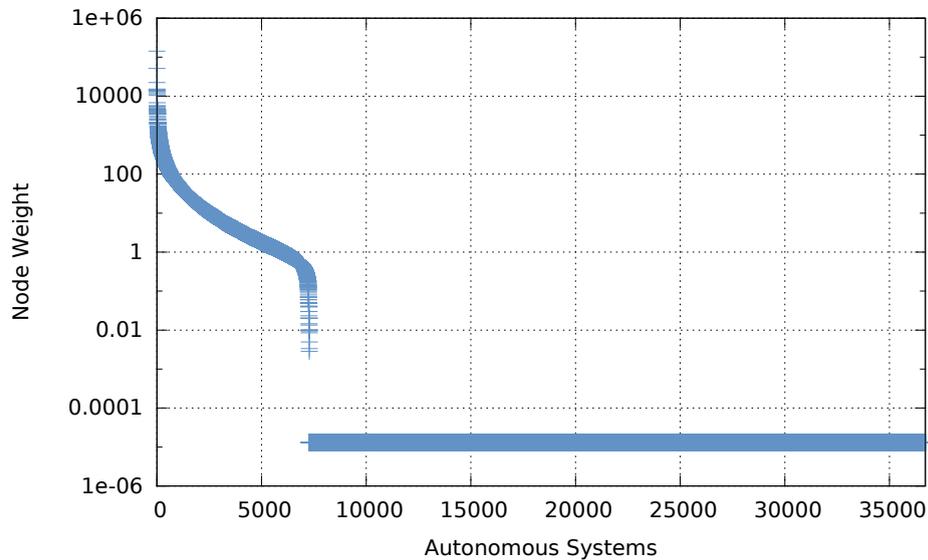
To use this list to assign weights based on the total number of page views to ASes, we had to link DNS names to AS numbers.

First, we map DNS names to IP addresses, which can then be mapped to the respective AS. A simple DNS lookup up for each hostname is not sufficient as this would not consider DNS caching, DNS-based load balancing and ultimately only provides a local view of name resolution from the view of our system. Instead, we obtained a global view of DNS resolution using 89 nodes on PlanetLab from all over the world. Each of the nodes was used to perform DNS lookups for the complete list of the top 100,000 websites. To bypass web server load balancing and DNS caching, three sequential lookups of the whole list were performed.

The resulting dataset consists of 27,801,818 IP addresses; 18,066 IP addresses from invalid private IP blocks were filtered and 1,269,239 lookups failed. After filtering invalid and duplicate addresses, 159,247 unique IP addresses were found.

These IP addresses were then mapped to AS numbers using CAIDA's “Routeviews Prefix to AS mappings Dataset”, a dataset available from CAIDA providing an IP prefix to AS number mapping based on BGP dumps.

Using this mapping, the weight of a node in the AS graph is then sum of the page views of all websites with an IP in this AS. A small default value is used if no website in the Alexa list has an address in the AS. The resulting weight distribution is depicted in



**Fig. 3.2:** Weight Distribution in Terms of Page Views for ASes

Figure 3.2. ASes that did not correspond to any sites within the top 100,000 websites were given a uniform small weight. We believe that the sharp drop in AS weight (around [7000, 0.5] in the plot) is likely due to the artificial limitation of only counting page views of the top 100,000 websites.

### 3.5 Experimental Results

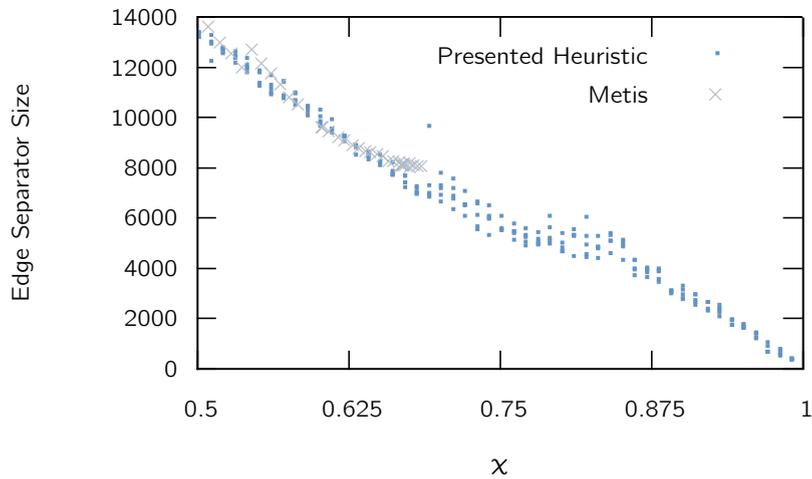
We used the heuristics presented in Section 3.3 to calculate separators for  $\chi \in [\frac{1}{2}, 1)$ . Since the heuristics are randomized, running the heuristic several times typically yields significantly different results. The plots in this section all show the results for five runs (not averaged, each run is represented by a single dot) where the result for each run is the best result obtained during 10 iterations of the original heuristic. On an Intel i7 920, our Java implementation used about 1 GB of memory and took typically around a minute to execute a single iteration of either heuristic.

We provide all experimental results for the AS graph generated by combining the data from BGP routing tables (Section 3.4.1) with the traceroute graph. The combined graph has 36,715 nodes and 99,852 edges.

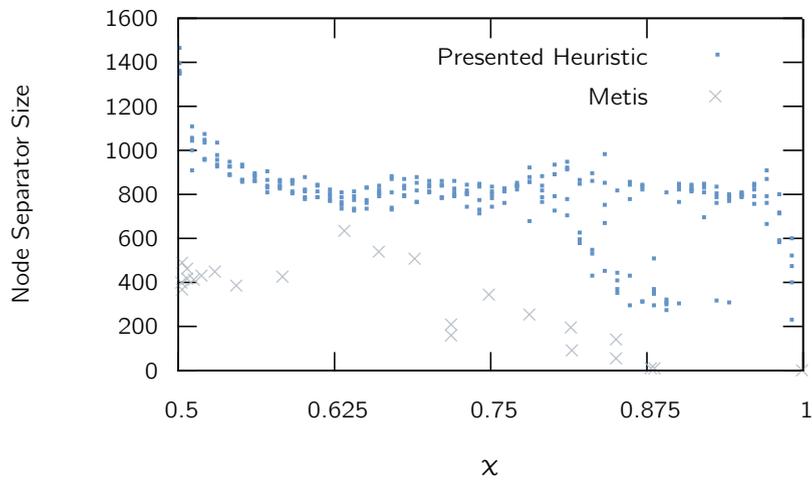
#### 3.5.1 Unweighted AS Graphs

First, we determine the size of the separators for the AS graph without weights. A  $\chi$ -separator in this case simply ensures that the largest remaining connected component would contain at most a fraction of  $\chi$  of the number of ASes. Figure 3.3(a) shows the size of the edge separators found by our heuristic (Section 3.3.1) in relation to  $\chi$ . The relationship between the size of the edge separator and  $\chi$  is linear, suggesting that separating the graph by edge removal requires incrementally cutting of ASes at the corners of the network. When compared to Metis, the main difference is that our heuristic is able to produce good separators for larger values of  $\chi$ .

Figure 3.3(b) shows the size of the node separators in relation to  $\chi$ . As expected, the



(a) Size of the Edge Separator



(b) Size of the Node Separator

**Fig. 3.3:** Size of the Computed Separators for the Unweighted AS Graph.

size of the node separators is significantly smaller. What is surprising is that the heuristic does not seem to always work consistently well for values of  $\chi$  close to 1; after all, the size of any separator should be monotonically decreasing as  $\chi$  increases.

For unweighted node separators, Metis produces significantly better results compared to the heuristic presented earlier. However, the results are somewhat less predictable as well.

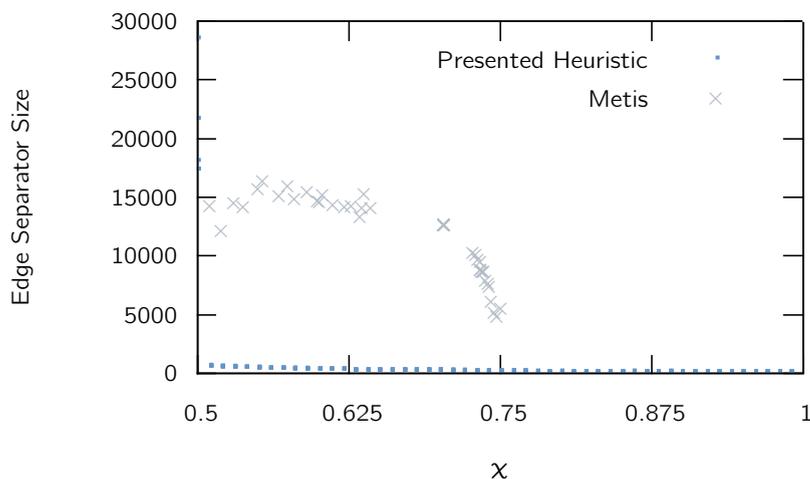
### 3.5.2 Weighted AS Graphs

Figure 3.4(a) shows the size of the edge separator in relation to  $\chi$  for weighted graphs. As expected, it is monotonically decreasing. However, the sharp drop from over 15,000 edges to around 500 edges at only 54% is quite surprising. The reasons for this are likely two-fold. First, achieving near-perfect balance for the weighted graph is harder because it essentially requires solving two knapsack problems, leaving little room for minimizing the size of the separator. Second, the extreme weight distribution of the ASes permits rather

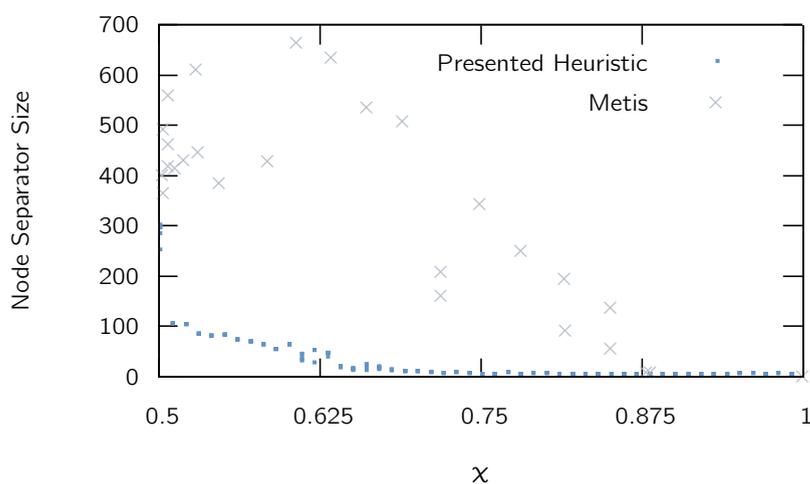
small separators (compared to the unweighted graph) because isolating a few ASes can have a huge impact on the size of the largest connected component.

For node separators, this second effect is even stronger. Figure 3.4(b) shows that removing a handful of ASes could be in theory quite devastating for reachability. These ASes are typically a combination of high-traffic ASes (such as Google) and ASes that are key for connectivity (such as Level 3). The first effect is less strong here, because the ASes that are part of the node separator are not part of any connected component, making it much easier to achieve a separation where the remaining connected components have less than  $\chi$  weight. As a result, for node separators,  $\chi$  close to  $\frac{1}{2}$  is not as restrictive.

Compared to Metis, the edge separators for the weighted graphs are significantly smaller using our heuristic. Metis seems to be unable to take advantage of the significant weight differences between the nodes. Again, Metis also does not produce separators for values of  $\chi > \frac{3}{4}$ .



(a) Size of the Edge Separator



(b) Size of the Node Separator

**Fig. 3.4:** Size of the Computed Separators for the Weighted AS Graph.

For node separators, this effect is less pronounced (Figure 3.4(b)). This is likely

because targeting ASes with high weights or high degree is equally effective in both cases. As a result, the increase in the size of the node separator for the combined AS graph is typically modest, with the exception of  $\chi$  values close to 0.5.

For node separators on the weighted graph, Metis performs again significantly worse than the presented heuristic; furthermore, Metis' results are widely dispersed and lack a clear trend towards smaller separators for increasing values of  $\chi$ .

### 3.6 Discussion

This research followed the common approach of combining traceroute and BGP data to obtain a reasonable approximation of the AS graph. Similar to existing studies that attempt to characterize AS graphs [RMRW10], the resulting AS graphs are not complete [CCG<sup>+</sup>02, CR06, RTM08]. The AS graphs could theoretically be augmented to be more accurate, for example using additional or simply more accurate measurements such as those described in [AKW09, VAC<sup>+</sup>08].

In an AS graph critical infrastructure is often not represented according to her significance for connectivity. Internet Exchange Points (IXPs) might be treated as sets of highly-connected nodes whereas a single undersea-cable might be represented as multiple logical edges which are much harder to separate. So applying our heuristic to a physical map of the Internet could bring interesting insights how to disconnect a certain region and partition the Internet.

We believe that the calculated separators are applicable to the various applications stated in the introduction:

- The elements of the separators do represent critical infrastructure; while additional edges in the AS graph may make them less critical and a smaller separator would point to even more critical systems, the separators do point to organizations and relationships that are key to global connectivity. The weighted AS graphs are useful in this application domain.
- The locations calculated would be useful for the placement of traffic monitoring equipment; additional edges might allow some traffic to escapes surveillance (but many applications, such as early warning systems, do not require completeness). Even smaller separators may reduce the cost; nevertheless, the heuristic represents an advance over existing algorithms.

### 3.7 Conclusion and Findings

This chapters characterized small edge and node separators in AS graphs and provided extensive data on the size of  $\chi$ -separators for various values of  $\chi$ . Analyzing the AS graphs, we found that allowing only a slight imbalance between the resulting connected components can significantly reduce the size of the separator. However, it would take the failure of 20–30 ASes to make make about 40% of Hypertext Transfer Protocol (HTTP) accesses fail for the AS graphs used in this study (Figure 3.4(b)). Furthermore, this number represents only a lower bound, as the observable AS graph is naturally a subset of the actual AS graph. As Byzantine failure of more than 20–30 ASes seems unlikely, separating the Internet on the AS-level is thus unlikely to work; future work in this area will thus have to include additional information about BGP routing policies or geographical knowledge about physical connections.

The goal of this chapter was to analyze if the Internet and its communication infrastructure can provide the foundation for a resilient communication infrastructure for decentralized peer-to-peer networks. The focus was to answer the question if the Internet can provide resilient communication paths between systems for peer-to-peer networks and if an attacker can partition the Internet in distinct partitions by making a set of communication links or providers fail and therefore impact the peer-to-peer network's ability to provide connectivity between systems both directly or via the peer-to-peer overlay.

As a result of this analysis we can say, that the Internet's communication infrastructure is surprisingly resilient against failing links and networks. An attacker trying to partition the Internet in fractions of roughly equal size would have to remove hundreds of networks and thousands of communication link to have an impact on the peer-to-peer networks possibility to connect peers and allow peers to communicate via the peer-to-peer overlay with each other (Figure 3.3(b)). So the Internet's backbone can provide a sufficient foundation to establish a peer-to-peer overlay with a decentralized peer-to-peer application. So when designing a resilient communication infrastructure, a special focus still has to be put on perimeter networks (like ISPs' access networks) only connected with few network links to the Internet which an attacker can make fail more easily.

## 4. RESILIENT AND SECURE COMMUNICATION FOR DECENTRALIZED NETWORKS

In this chapter we present the design and implementation of a resilient and secure communication infrastructure for the GNUUnet peer-to-peer framework. GNUUnet's philosophy is to be a reliable, open, non-discriminating, egalitarian, unfettered, and censorship-resistant system of free information exchange. GNUUnet is designed to protect the privacy of its users and to guard itself against attack or abuse.<sup>1</sup>

The foundation of such a system is the communication and the ability to exchange information between participants in the network. The communication infrastructure presented in this chapter is the foundation for GNUUnet to provide such a free and reliable system. The goal of GNUUnet's communication infrastructure is to connect participants in the peer-to-peer network and provide resilient and secure communication between these participants.

Direct communication between systems can be limited for various reasons as we saw with middleboxes in Section 2.6 and various parties try to influence and control free and unrestricted communication between users on the Internet as described in Chapter 1. With the communication infrastructure presented in this section, we try to increase connectivity between systems and antagonize degradation and censorship attempts.

### 4.1 Objectives

#### **Increase Connectivity:**

In case no communication infrastructure is available or existing communication infrastructure is failing, GNUUnet should still be able to connect users and allow them to communicate.

Here GNUUnet should provide support to create ad hoc networks using wireless technologies like Bluetooth or WLAN to connect peers directly without relying on deployed infrastructure. To increase connectivity between peers in the peer-to-peer overlay, GNUUnet's transport infrastructure provides a Distance Vector (DV) routing component to route traffic between peers not connected directly via intermediate peers in the peer-to-peer overlay.

The communication infrastructure should improve connectivity for participants located in restricted environments or affected by limitations to end-to-end connectivity as described in Section 2.6.

To improve connectivity between peers, GNUUnet's transport infrastructure employs techniques like collaborative and non-collaborative NAT traversal, connection reversal, forward and reverse proxies.

#### **Resilience against Degradation:**

Since peer-to-peer applications are particularly often affected by traffic management

---

<sup>1</sup> <https://gnunet.org/philosophy>

and filtering attempts as described in Section 1, GNUnet's communication infrastructure should be resilient against and antagonize service degradation attempts.

GNUnet's approach to counteract such attempts is to support multiple transport protocols and continuously monitor communication channels for degradation attempts. If a communication channel is effected by degradation, GNUnet can switch to a different communication channel providing better performance properties.

#### **Authenticated and Secure Communication:**

GNUnet's communication infrastructure has to authenticate a communication partner before establishing an underlay connection with a peer. The communication infrastructure has to validate the identity of the peer and verify that the set of addresses provided to establish a connection with the remote peer can really be used to communicate with this particular peer. This validation is required to prevent *man-in-the-middle* attacks and attackers trying to measure, analyze or manipulate traffic exchanged between peers.

Communication between peers in the peer-to-peer overlay must be confidential, integrity protected and authenticated. Peers establishing a connection in the peer-to-peer overlay must authenticate the communication partner and establish a secure connection with the communication between the peers being confidential and protected against modification.

GNUnet's communication infrastructure distinguishes between *validated* and *unvalidated* peer information. All information received from external sources (e.g. via bootstrapping mechanisms) is treated as unvalidated and has to be validated before it is used to establish a connection with a remote peer. With this validation process, a peer cryptographically proves its identity and that it can be reached using the provided set of addresses. This validation process ensures that a communication partner is really the peer it purports to be and prevents *man-in-the-middle* attacks.

Link encrypted communication in the peer-to-peer overlay between directly connected peers is provided by GNUnet's CORE service including perfect forward secrecy and integrity protection. Communication is encrypted using two symmetric block ciphers.

## **4.2 Scope and Limitations**

#### **A Fully Decentralized Network:**

GNUnet is designed to be a fully decentralized peer-to-peer application not requiring or relying on any centralized entities. This requirement is also true for the communication infrastructure designed in this chapter. For GNUnet's communication infrastructure, we do not rely on any centralized entities to coordinate communication between peers in the peer-to-peer network and the setup of the peer-to-peer overlay.

#### **An Unknown and Dynamic Environment:**

For GNUnet, we assume that both participants in the network and the number of participants are not a priori known and can change over time. This implies that a peer does not know the peers it can communicate with and has to learn this information when joining the network. All peers can join and leave the network and establish connections to other peers without prior coordination.

**No Trusted Entities and No Trust Between Entities:**

Besides having no centralized entities in the network, we assume as an extension to this assumption that there are no trusted entities and no trust between participants in the network. A peer cannot trust any other peer or information provided by other peers. All information obtained from other entities in the network has to be validated before it can be used by a peer. This extends to the assumption that any set of peers in the network may be malicious and peers even collaborate to attack a peer and we do not assume an upper limit on the number of malicious peers in the network.

**No Trust in Underlying Infrastructure:**

GNUnet's communication infrastructure uses existing communication infrastructure as a foundation to allow peers to communicate. GNUnet should rely as little as possible on these technologies and assume the existing infrastructure to be unstable or compromised. GNUnet's infrastructure has to ensure authenticity, integrity and confidentiality of communication when exchanging data between participants and combine this with the idea that information cannot be trusted without prior validation.

**Integrity within a Peer:**

Contrary to the assumption that there is no trust between peers, we assume that the integrity of a peer, integrity of information stored with a peer and integrity between collaborating components of a peer is not compromised.

An objective this work does not focus on is traffic hiding or traffic obfuscation of peer-to-peer traffic. These techniques are often used to make traffic less suspicious for network filters. Traffic obfuscation tries to modify an application's traffic in a way that it is not recognizable as traffic from an particular application or by trying to make the traffic look like traffic originating from a "benign" application. While this sounds like a reasonable approach to achieve resilience against service degradation, we think it is an arms race with the advantage on the side of the censor, therefore hard to win. Works like [HJ10] show that many protocols, even when obfuscated, have statistically measurable properties allowing it to detect an obfuscated protocol easily by analyzing just a some hundreds of bytes of application traffic. In this work, we therefore focus on providing alternatives when service degradation is detected and do not rely on traffic obfuscation for censorship resistance.

### 4.3 Design and Implementation

GNUnet's transport infrastructure consists of several components. These components are implemented in independent *services* or *daemons* collaborating with each other via IPC according to GNUnet's design principles described in Section 2.8.3. Each component provides a distinct functionality important for the functioning of the transport infrastructure. GNUnet's transport infrastructure consists of the following components also depicted in Figure 4.1:

HOSTLIST daemon:

Download peer information from bootstrapping servers and provide bootstrapping server functionality

PEERINFO service:

Persistent storage of validated peer information

TOPOLOGY daemon:

Management of the peer-to-peer overlay topology and friend-to-friend connections

NAT library:

NAT traversal and hole punching, UPnP support, management of IP addresses available with the local peer

ATS service:

Management of addresses and active sessions, providing address suggestions to TRANSPORT, resource allocation to peers

TRANSPORT service:

Validating address information, establishing underlay connections to remote peers, managing incoming connections from remote peers, manage peer's address information, manage transport mechanisms, enforce resource restrictions

CORE service:

Encrypted communication between peers, cryptographic key exchange and link encryption between peers, Ethernet-like semantics for communication including possible out-of-order delivery and unreliable communication

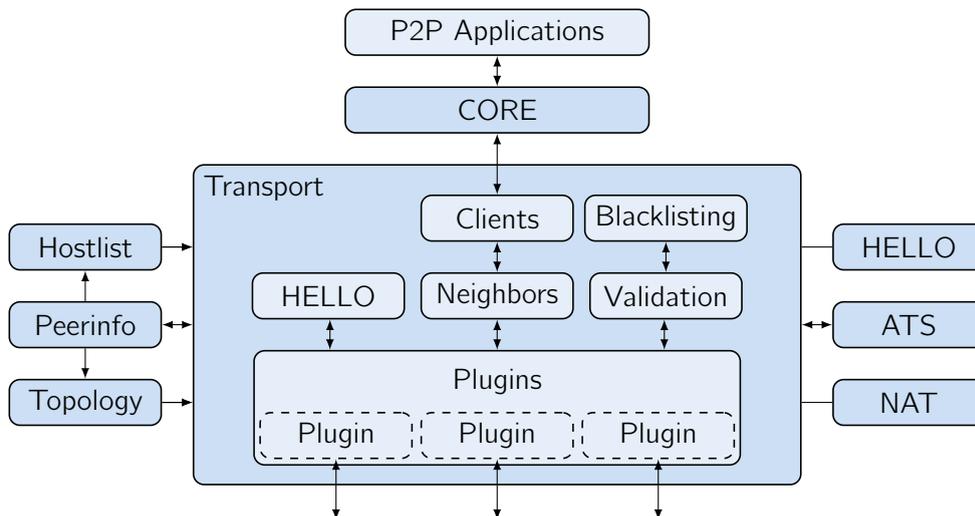


Fig. 4.1: Components of GNUet's Transport Infrastructure

In this Section, we present the design and implementation of GNUet's PEERINFO, TOPOLOGY, HOSTLIST, HELLO and NAT components and discuss in detail the TRANSPORT service responsible to maintain underlay connectivity with remote peers. We briefly describe the CORE service and the ATS service which is presented in detail in Chapter 5 discussing address selection and resource allocation for decentralized peer-to-peer networks.

#### 4.3.1 Peers and Peer Identities

In GNUet, every peer is equipped with a public/private key pair and a peer's *identity* is the public key of this public/private key pair. Within the peer-to-peer overlay this identity

is used to refer to a particular peer and route traffic between peers.

With respect to cryptography, GUNet relies on Elliptic Curve Cryptography (ECC) using Ed25519, specifically the EdDSA digital signature scheme and Curve25519 [BDL<sup>+</sup>12]. Every peer provides an EdDSA public/private key pair used for authentication. An Ed25519 public key has a size of 256 bit (32 byte), stored in GUNet using the Ed25519 compact format and an Ed25519 signature has a size of 64 bytes. Within the GUNet framework, peer identities are represented using a `GNUNET_PeerIdentity` struct used as a wrapper around the EdDSA public key. Using such a wrapper makes other components using peer identities independent from the specific cryptographic implementation used. By using the public key as a peer identity, we can assume with a very high probability that the identity of this peer is unique in the network, since otherwise another peer would have generated the same Ed25519 key pair.

To provide confidential communication, GUNet's CORE service uses Elliptic Curve Diffie Hellman Ephemeral (ECDHE) [Res99] to provide a secure key exchange over the insecure communication underlay. This shared secret is used to derive a shared session key for symmetric encryption using the HKDF derivation function defined in [KE10]. This derived key is used to encrypt communication with other peers using both the Advanced Encryption Standard (AES)-256 [DR02] and Twofish[SKW<sup>+</sup>98] encryption ciphers.

For representation purposes, the binary representation of the public key can be converted to a human readable representation using Crockford Base32 ASCII encoding [Cro, Jos06]. Peer identities in GUNet are usually represented using this ASCII encoding, representing peer identities as string with 52 characters or in a short form with only 4 characters:

- Printable version of a peer identity:  
R8TTJ9GAL5VIFOFNM8KNT3D83BVQPBNRHJSSD0IME63V821906EG
- Shortened version of a peer identity:  
R8TT

GUNet supports developers with an extensive cryptographic utility library providing functionality to create and verify signatures, derive keys from existing cryptographic key material, encrypt and decrypt data using symmetric block ciphers, hash data blocks etc.

### 4.3.2 Plugin Specific Address Formats

GUNet's transport infrastructure tries to antagonize service degradation and to improve connectivity between peers by supporting multiple transport mechanisms in parallel and switching between transport mechanisms in case a degradation attempt or failing communication infrastructure is detected. By supporting a set of different transport mechanisms, we have to cope with a large number of alternative address formats varying from mechanism to mechanism. In GUNet's transport infrastructure every transport mechanism is implemented in a so called *transport plugin* as described in Section 4.3.11.3 and can define its own address format used internally suitable for the communication mechanism it relies on: an IP based transport mechanism may store address information in form of a socket address struct, while a plugin using MAC addresses may choose a byte array to store this MAC address. Only the plugin itself knows about the structure of the address format and the semantics. This format cannot be interpreted by other components. When a transport plugin passes such an address to other components, it uses an untyped pointer and a variable specifying the length of an address.

The address format is plugin specific, but common properties shared between the plugins are defined: each address contains the address information required to establish the connection and in addition an option bit field. This option bit field is used to indicate special options associated with an address which cannot be encoded in the address itself. The encoding of options in the bit field is plugin specific and is defined with the address format.

An example for such an option is the `HTTP_OPTIONS_VERIFY_CERTIFICATE` flag used with the `HTTPS` plugin. The `HTTPS_SERVER` plugin sets this flag in the address if it can provide a valid `HTTPS` certificate and wants to instruct the `HTTPS_CLIENT` plugin to validate this certificate when establishing a connection. This information cannot be included in the Uniform Resource Locator (URL) used to establish the connection and is specific to the `HTTPS` plugin and is not useful to any other plugin.

Using a plugin specific address format simplifies address handling and management since plugin specific addresses do not have to be converted to a generic format when used outside the transport plugin and do not have to be converted back to the plugin specific format for internal operations. Since address handling is only implemented in the plugin itself, other components of the transport infrastructure do not have to be adapted when new transport plugins are added.

Since the transport infrastructure is agnostic of the address format, the transport plugins provide functions for address handling used by the rest of the infrastructure to process addresses. This covers conversion from plugin specific format to a human readable representation and parsing human readable addresses and convert them to the plugin specific address format. The plugins provide special conversion functions with the plugin API as described in Section 4.3.11.2.

To achieve a common format for conversion of plugin specific addresses to a human readable representation, a common format for human readable addresses is defined. This format is defined as a triplet containing information about the plugin, the address specific options and the address itself, separated by dots:

```
<plugin>.<address specific options as unsigned integer>.<address>
```

So a `TCP` address with the connection endpoint `203.0.113.1`, port `2086` and without address specific options (option value is `0`) is represented as:

```
tcp.0.203.0.113.1:2086
```

### 4.3.3 Generic Address Format

Within the transport infrastructure, additional information are required to manage connections with other peers. In `GNUnet`, we therefore introduce a `GNUNET_HELLO_ADDRESS` struct used as a generic address format to handle plugin specific addresses outside of the plugins. This address format is called a `HELLO` address. A `HELLO` address contains information about the peer an address belongs to, the transport plugin this address has to be used with, the address itself in plugin specific format, the length of the address and an option field containing additional information associated with the address on the local peer. One example for such a flag is the `GNUNET_HELLO_ADDRESS_INFO_INBOUND` flag, indicating that an address cannot be used to establish an outbound connection, for example for an incoming `TCP` connection containing a `TCP` private port (as described in Section 2.3.1) as source address which cannot be used to connect to the remote peer. This address format supports state independent handling of addresses and encapsulates all information related to an address.

HELLO addresses are handled by the GUNet HELLO library. This library provides functionality to manage HELLO addresses and provides functionality to allocate, copy, compare and free HELLO addresses.

A HELLO address contains the identity of the peer this address belongs to and the name of the transport plugin this address can be used with as a string. To store plugin specific addresses in a HELLO address, the format provides an untyped (void) pointer to store the plugin specific address and a length field to store the size of the plugin specific address. The local information bit field can contain additional information but is only valid in the context of the local peer. This information is not exchanged with other peers.

#### 4.3.4 Transport Sessions

Both generic HELLO addresses and plugin specific addresses only designate an address and how to communicate with a remote peer. To refer to a specific connection established with a peer using a particular address, we introduce the notion of a *session*: a session refers to a specific connection established with a remote peer using an address. A session can be seen as an “instantiation” of an address.

When a peer wants to connect to a remote peer, it selects an address to use and requests the respective plugin to use this address and connect to this peer. If not already connected to this address, the plugin creates a connection and returns a session reference to the transport infrastructure, which can be used to refer to this specific connection. In GUNet’s transport infrastructure sessions are realized with pointer references: a plugin internally allocates a plugin-specific session object and passes an opaque pointer to this session to other components. Other components use this pointer as a handle to the session, but cannot access information stored in this session.

#### 4.3.5 HELLO messages

To establish connections with remote peers and bootstrap the peer-to-peer overlay, peers have to exchange information allowing other peers to connect them and authenticate the connection. For this purpose the GUNet peer-to-peer framework uses so called HELLO messages containing all information required to establish a secure connection with a peer. HELLO messages are similar to business cards containing information about a peer’s identity and the different addresses to contact this peer.

A peer’s HELLO message contains all addresses belonging to this peer as reported by the different transport plugins in the generic HELLO address format described in Section 4.3.3. A validity lifetime is added for each address to ensure stale address information is eventually removed from the network. In addition, the HELLO message contains the peer’s public key to allow remote peers to verify the identity of the peer by authenticating the connection to prevent man-in-the-middle attacks and peer’s pretending false identities. A remote peer receiving such a HELLO message can extract the addresses contained in this message and the peer’s public key and use this information to connect to the peer and cryptographically authenticate the peer’s identity.

The functionality to convert addresses to the HELLO format which can be transmitted over the network is provided by the HELLO library described with HELLO addresses in Section 4.3.3. This library can create and parse HELLO messages and extract addresses contained in HELLO messages.

The HELLO library can convert HELLO messages to a human readable format represented as a Uniform Resource Identifier (URI). This HELLO URI can be used to exchange peer information out-of-band between users of the network. The HELLO library uses the

transport plugins to convert plugin specific addresses to a human readable format as described in Section 4.3.2. A HELLO URI contains the peer's public key and all addresses available with this peer. For every address, the URI contains the expiration time of the address, the transport plugins this address has to be used with and the address itself in the human-readable format as described with plugin-specific addresses. The generic format for a human-readable HELLO URI is:

```
gnunet://hello/<peer identity>+<expiration address 1>+<plugin address
1>+<address 1>+...+<expiration address n>+<plugin address n>+<address n>
```

So a HELLO URI for peer R8TT... being reachable using TCP address 203.0.113.1 port 2086 would look like:

```
gnunet://hello/R8TTJ9GAL5VIFOFNM8KNT3D83BVQPBNRHJSSD0IME63V821906EG+0+tcp+tcp.0.203.0.113.1:2086
```

### 4.3.6 Bootstrapping and Neighbor Discovery

To bootstrap the peer-to-peer overlay network and allow peers to connect to each other, information about peers available in the network has to be exchanged between peers in the network. This information is contained in the HELLO messages described in Section 4.3.5 and GUNet provides different techniques to spread information about peers in the network.

#### 4.3.6.1 Peer Information Included in the Software

The first technique used is to include peer information in the peer-to-peer software itself and to ship this information with the software. With GUNet, HELLO messages of peers known to be running reliably and to be reachable from the Internet are shipped directly with the distribution of the GUNet software. The expiration time for HELLO messages shipped with GUNet is overwritten to prevent these HELLO messages from expiring. With this approach a new peer can connect directly to the network without having to use any other bootstrapping mechanism.

#### 4.3.6.2 Out-Of-Band Exchange of Peer Information

HELLO messages can also be exchanged directly between users using an out-of-band mechanism like e-mail, instant messaging or Quick Response (QR) codes. The HELLO library provides the functionality to generate a human readable URI version of a peer's HELLO message as described in Section 4.3.5. Users can export and import peer information using the PEERINFO Command Line Interface (CLI). The PEERINFO CLI exports peer information as an URI and these URIs can be exchanged between users or converted to QR codes. A QR code can be imported to GUNet using the `gnunet-qr` tool able to scan a QR code using a webcam.

#### 4.3.6.3 Bootstrapping Mechanisms

The third approach provided are so called *hostlist* servers. Peers joining the network can connect to these servers to learn about peers existing in the network. In GUNet, these bootstrapping servers are called *hostlist* servers. A GUNet hostlist server is a HTTP or HTTPS server, peers can connect to and download a list of HELLO messages. These

hostlist servers are “normal” GUNet peers and every peer can enable the functionality to provide a hostlist to other peers. In the GUNet framework, several hostlist servers are pre-configured to allow new peers to join the network. This functionality does not contradict to the assumption of a fully decentralized network without trusted authorities since every peer in the peer-to-peer network can provide a hostlist to other peers, users are free to add and remove hostlist servers and information obtained from hostlist servers is treated as untrusted and checked using an validation process before used to communicate with remote peers.

In GUNet, the hostlist functionality is provided by the HOSTLIST daemon. This is a GUNet daemon, providing the functionality to serve a hostlist to other peers and download hostlist information from hostlist servers. The HOSTLIST daemon is collaborating with the PEERINFO service, responsible for persistent storage of peer information, described in Section 4.3.7. The HOSTLIST daemon functionality is split in a client and a server part:

The HOSTLIST server is responsible to provide the hostlist information to other peers. It obtains the HELLO information from the PEERINFO persistent storage and makes it publicly available acting as a HTTP web server other peers can contact to download the list of HELLO messages known by this peer.

The HOSTLIST client component is responsible to obtain hostlist information from hostlist servers. The HOSTLIST periodically contacts HOSTLIST servers to download information about other peers. Based on the set of hostlist servers known to the HOSTLIST daemon on a peer, it connects to one of the hostlist servers and downloads the HELLO messages from the server. The HELLO messages downloaded from the server are treated as untrusted and have to be checked using TRANSPORT’s address validation process described in Section 4.3.11.5 before storing this information with the PEERINFO service.

To make bootstrapping more resilient, the HOSTLIST daemon also provides the functionality to distribute and learn information about hostlist servers using the peer-to-peer overlay. Peers can enable *hostlist learning* to learn about hostlist servers from other peers and enable *hostlist advertising* to advertise their hostlist to other peers. Both hostlist learning and advertising are realized by exchanging information via the peer-to-peer overlay using the CORE service described in Section 4.3.19. When a peer with learning enabled connects to peer with hostlist advertising enabled on CORE-level, the HOSTLIST daemon notifies the connecting peer about the URL of the hostlist server available. A peer with hostlist learning enabled receiving such an URL can add the URL to the set of hostlist servers available and later on download HELLO messages from this bootstrapping server. The functionality of the HOSTLIST daemon is illustrated in Figure 4.2.

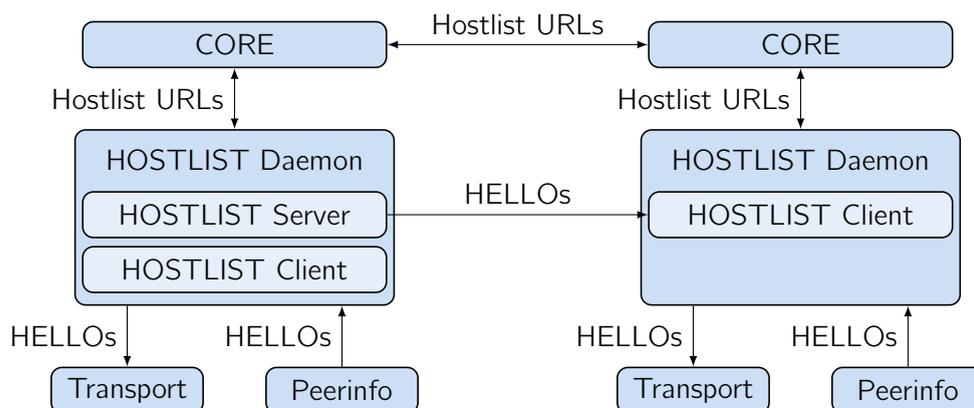


Fig. 4.2: The HOSTLIST Daemon

#### 4.3.6.4 Neighbor Discovery Mechanisms

Besides the mechanisms enabling new peers to join the network, GUNet provides additional techniques to autonomously discover peers. Peers can use *neighbor discovery* provided by the UDP transport plugin described in 4.3.14, the Bluetooth plugin described in 4.3.17, and the WLAN plugin described in 4.3.16 to learn about additional peers. These plugins periodically broadcast the peer's HELLO message as *beacons* in the network and other peers on the same network can receive these beacons. HELLO messages received are forwarded to the transport infrastructure which validates the address information contained in the HELLO messages and uses this information to establish a connection to this peer.

Peers can also use so called *gossiping*. With gossiping, peers connected to each other in the peer-to-peer overlay exchange HELLO messages about other peers. This is realized by the TOPOLOGY daemon described in Section 4.3.9 responsible to maintain the topology of the peer-to-peer overlay.

#### 4.3.7 Persistent Storage of Peer Information

To store validated peer information in a persistent way, to ensure these information are available after a restart of a peer and to provide transport components with peer information, GUNet provides a central storage component for peer information in a dedicated service. GUNet components can interact with this service to store and obtain peer information.

The PEERINFO service is a persistent storage to store information about peers and addresses using HELLO messages as described in Section 4.3.5. This storage component provides the functionality to store validated information obtained from bootstrapping and neighbor discovery mechanisms in a persistent way and provides other services with this information. Addresses stored with the PEERINFO service are treated as validated since they were verified using the validation mechanism described in Section 4.3.11.5. With the PEERINFO service, components collaborating with the service can be supplied with validated addresses directly after the peer starts without having to obtain peer information from other peers using the bootstrapping and neighbor discovery mechanisms first. The PEERINFO service stores HELLO message in a directory on the system's hard disk and parses this directory during start up.

Users can interact with PEERINFO using a CLI tool to list peer information available on the system and to export and import peer information using HELLO message URIs. The PEERINFO service interacts with different other components: it is provided with peer information to store by the bootstrapping and neighbor discovery mechanisms when information was successfully validated by address validation. PEERINFO provides validated address information to the transport infrastructure to establish connections with remote peers and to bootstrapping and neighborhood discovery mechanisms to exchange peer information with remote peers.

#### 4.3.8 Address Management and NAT Support

GUNet tries to increase connectivity for peers located in restricted environments using both collaborative NAT traversal techniques like UPnP and the Internet Gateway Device Protocol (IGDP) [For03] and non-collaborative NAT traversal techniques like ICMP and UDP hole punching as described in [MEGK10]. Increasing connectivity for peers affected by NAT middleboxes consists of both determining the external IP address (and port)

remote peers can use to contact a peer and to interact with the NAT device to make the device forward data from the external address to the internal address of the peer. A functionality closely related to interacting with NAT middleboxes and determining the public IP of a peer affected by NAT is therefore the management of IP addresses available on a local system and provide transport components and plugins with these addresses. Due to this close relation, GUNet realizes both functionalities in a single component, the NAT library.

GUNet's NAT library supports to collaborate with NAT devices to establish port forwarding using UPnP and the IGDP [For03]. With IGDP, a peer located behind a NAT device can ask the NAT device to create a forwarding from the external address of the NAT device and a port to an internal address and a port. As a result, IGDP will return the external IP address and the port to use to contact to client requesting this mapping. This approach is particularly useful in home environments, where routers support UPnP and UPnP can be used to automatically configure a dynamic port forwarding.

In addition to establishing port forwarding dynamically, NAT devices commonly support *static port forwarding*, where the mapping from port on the NAT device to internal address and port is manually configured by the user on the NAT device. GUNet's NAT library supports this approach by allowing the user to configure information about the port forwarding (i.e. the external IP address and port number) in the configuration and give this information to other GUNet components and peers.

Besides collaborative methods to establish port mappings, the NAT library also supports NAT traversal methods like Internet Control Message Protocol (ICMP) and UDP based hole punching described in [MEGK10]. With this approach, certain NAT devices can be tricked to establish a port mapping without actively supporting to establish port forwarding.

A task closely related to the task of traversing NAT devices is the management of IP addresses available on a system including information about external IP addresses provided by NAT devices and dynamic IP addresses. With GUNet supporting multiple IP based transport mechanisms, information about available IP addresses has to be provided to all IP based transports. The NAT library provides functionality to determine both IPv4 and IPv6 addresses available on a system by periodically scanning the local network interfaces. Besides scanning available network interfaces, the NAT library also includes information about external public IP addresses obtained from NAT traversal techniques and can interact with an external server to obtain the public IP address of a peer. To support dynamic IP addresses often used by ISPs, the NAT library can be configured to resolve an external DNS hostname configured by the user to obtain the public IP address to use. This approach is useful with dynamic IP addresses and Dynamic DNS (DDNS), as defined in [VTRB97], often used to dynamically update DNS names and allow systems provided with dynamic IP addresses to host services.

The NAT library realizes NAT traversal and management of IP address in a single component and other GUNet components requiring this information can register with the NAT library to be notified about changes to the IP addresses available for a peer. This is in particular important for IP based transport plugins. These transport plugins use this information to update their internal list of available addresses and notify the transport service to update the peer's HELLO message based on this information as described in Section 4.3.5.

### 4.3.9 Overlay Topology Management

To establish and maintain an unstructured mesh peer-to-peer overlay topology, GNUnet's transport infrastructure provides special component responsible for maintaining overlay connectivity with other peers. The TOPOLOGY daemon interacts with the PEERINFO service to obtain information about peers available to establish overlay connections. Based on the information provided by PEERINFO, TOPOLOGY maintains a list of known peers and tries establish overlay connections with a pre-configured number of peers to achieve a resilient overlay topology. TOPOLOGY tracks pending, successful and failed connection attempts. It also interacts with the CORE service to get notified about successful overlay connections established with other peers. The TOPOLOGY daemon instructs the TRANSPORT service to establish an underlay connection and triggers reconnect attempts with the TRANSPORT service when a peer was disconnected in the peer-to-peer overlay. Here TOPOLOGY applies an adaptive reconnect frequency to throttle reconnect attempts when a peer is currently not available and connection attempts fail.

#### 4.3.9.1 Friend-To-Friend Topologies

TOPOLOGY can also restrict connections. GNUnet can operate in a so called *friend-to-friend* (*f2f*) mode, where a peer is only allowed to connect to a set of designated, explicitly *white-listed* peers. This list of white-listed peers is configured by the user and can be used by the user to create a topology only including explicitly white-listed peers. Friend-to-friend connections are controlled by the TOPOLOGY daemon which manages and issues requests to TRANSPORT service to establish underlay connections to remote peers. If the peer is running in friend-to-friend mode, the TOPOLOGY daemon only issues connection requests to peers in the list of white-listed peers and instructs TRANSPORT to block all other peers.

With respect to HELLO addresses, addresses of *f2f* peers must not be leaked to (public) bootstrapping mechanisms due to privacy issues. The HELLO library therefore supports the notion of a *friend-to-friend HELLO* message. If a peer operates in friend-to-friend mode, a specific friend-only HELLO message is created. This friend-only HELLO is not given to components like bootstrapping or neighbor discovery mechanisms which may leak a HELLO to peers.

#### 4.3.10 Managing Active Addresses and Session

Besides having to manage validated addresses, GNUnet's transport infrastructure also has to manage information about active and established connections available to communicate with remote peers. To manage this information, a dedicated component is used to manage information about addresses and established sessions. The automatic transport selection service (ATS) is responsible to manage information about available addresses and sessions which can be used by TRANSPORT to communicate with remote peers. ATS can suggest these addresses to the TRANSPORT service and distributes bandwidth resources among peers according to bandwidth restrictions defined by quotas as described in Section 4.3.11.13.

When an address is successfully validated or an incoming connection is established by a remote peer, the respective address and session information is given to the ATS service. This service maintains a list of addresses and sessions which can be used to exchange data with a remote peer. When the TRANSPORT service is instructed to establish an underlay connection to a peer, it requests an address from ATS. When TRANSPORT requests an address from ATS for a peer, the ATS service evaluates which address and session is the

“*best*” address currently available to communicate with this peer and suggests this address to TRANSPORT. As long as the transport infrastructure does not cancel this request, ATS can reconsider this decision and suggest a different address or change the amount of resources assigned. It is important to note that ATS does not decide with which peers to establish connections (since this is the focus of the TOPOLOGY daemon), but ATS can decide to disconnect a peer if no suitable address or insufficient resources can be provided.

The ATS service and its functionalities to determine the *best* address to communicate with a peer and to distribute resources among peers is the focus of Chapter 5 and is discussed in detail in this chapter with a special focus on address selection and resource allocation approaches suitable for the use in fully decentralized peer-to-peer networks. In this chapter we focus on the design and implementation of GNUnet’s transport infrastructure and on the ATS functionality to manage and suggest addresses for the transport infrastructure and assign bandwidth resources to addresses.

#### 4.3.11 The Transport Service

A central and important component, realizing several main functionalities required for GNUnet’s transport infrastructure is the TRANSPORT service. The TRANSPORT service is the central component of GNUnet’s transport infrastructure responsible to establish underlay connections with remote peers and manage incoming connections from remote peers. GNUnet’s TRANSPORT service provides functionality for:

Address Validation

Module: *transport\_validation*

Described in Section [4.3.11.5](#)

Peer’s Cryptographic Information

Module: *transport*

Described in Section [4.3.11](#)

Peer’s HELLO message

Module: *transport\_hello*

Described in Section [4.3.11.4](#)

Interaction with Higher Layer Applications and Clients

Module: *transport\_clients*

Described in Section [4.3.11.14](#)

Connections with Remote Peers

Module: *transport\_neighbours*

Described in Section [4.3.11.7](#)

Blacklisting Connections with Remote Peers

Module: *transport\_blacklist*

Described in Section [4.3.11.6](#)

Transport Plugins

Module: *transport\_plugins*

Described in Section [4.3.11.1](#), Section [4.3.11.2](#), Section [4.3.11.3](#),

Enforcing Resource Restrictions

Described in Section [4.3.11.13](#)

The TRANSPORT service is a GNUnet service and orchestrates the functionalities described above and the collaboration between these components. During startup, the

TRANSPORT service parses the peer's configuration settings, loads the peer's public/private key pair, initializes the service and starts the different TRANSPORT components. It controls the different components and provides them with additional functionalities like access to the peers configuration settings, peer statistics and the public/private key pair. In the following, we will introduce the different components and describe the detailed functionalities, their interaction with internal and external components. The internal structure of the TRANSPORT service with its components is depicted in Figure 4.3.

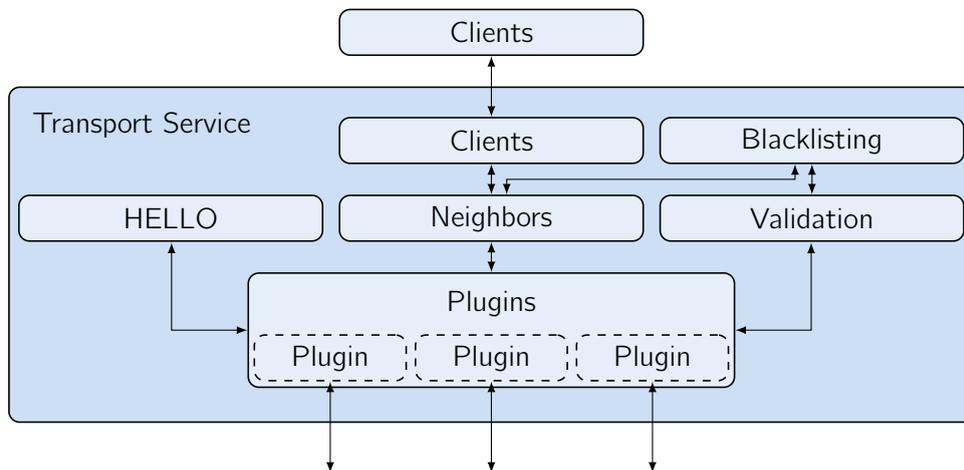


Fig. 4.3: The TRANSPORT Service

#### 4.3.11.1 The Transport Service Plugin Architecture

GNUnet's transport infrastructure supports the use of multiple different transport mechanisms to be resilient against service degradation and to provide connectivity in case of missing or failing infrastructure. To provide an extensible design allowing new transport plugins to be added in the future, the TRANSPORT provides an extensible plugin architecture for transport mechanisms and supports to implement transport mechanisms as loadable plugins. These transport plugins are self-contained and hide their implementation from the TRANSPORT services. The TRANSPORT service and plugins interact using an API. This design allows to add new plugins without modifying the transport service and to modify the plugin implementation without having to update to the TRANSPORT implementation. The API is designed to allow both the plugin to access functionality provided by the TRANSPORT service as well as the TRANSPORT service to access functionality provided by the plugins.

Each peer can load multiple transport plugins, but does not have to support all plugins. Which plugins a peer loads is configured by the user. Two peers trying to communicate with each other only need to have one transport plugin in common. There is no dedicated negotiation mechanism to find the common set of plugins between two peers. Addresses using plugins not available on a peer are filtered by the address validation process described in Section 4.3.11.5.

To manage transport plugins, the TRANSPORT service contains the PLUGINS module. This module is initialized during TRANSPORT startup and parses the configuration of the TRANSPORT service for the plugins configured and initializes each plugin. Each transport plugin is implemented as a self-contained module, which can be loaded by the TRANSPORT service if the plugin is configured to be active. To load the plugin, the TRANSPORT service relies on the plugin functionality provided by GNUnet's utility library. This functionality

requires the plugin to provide two functions using a fixed nomenclature to load and unload the plugin. During start up, the PLUGINS component initializes the plugin using the initialization function. As an argument for this function, it passes an *environment* object, containing data structures provided by the TRANSPORT service, including a configuration and a logging handle and references to the callback functions provided by the TRANSPORT service. These callback functions are used by the plugins to interact with the TRANSPORT service. After initialization, the plugin returns an `api` structure to the PLUGINS component containing information about the API functions provided by plugin. This `api` object is stored by the PLUGINS module and other components of the TRANSPORT service can obtain these objects to interact with the plugins. During TRANSPORT shutdown, the PLUGINS module instructs all plugins to shutdown and free all data structures using the shutdown function required for every plugin.

#### 4.3.11.2 Transport Service Plugin API

The TRANSPORT plugin API provides the interface for the TRANSPORT service to access functions provided by the transport plugins and for the plugins to access TRANSPORT service functionality. The plugins provide functions to the TRANSPORT service to initiate and terminate connections, send data to other peers and convert addresses from and to the plugin specific address format used by the plugin as as described in Section 4.3.2. The functions required to be provided by all plugins are:

```
struct Session * get_session (void *cls,
                             const struct GNUNET_HELLO_Address *address);
```

Initiate a new connection to a peer using a specific address. The plugin checks if a session is established for the address and if so returns the existing session. If no session is established with this session, the plugin establishes a new session to this address and returns the new session.

```
ssize_t send (void *cls,
              struct Session *session,
              const char *msgbuf,
              size_t msgbuf_size,
              unsigned int priority,
              struct GNUNET_TIME_Relative to,
              GNUNET_TRANSPORT_TransmitContinuation cont,
              void *cont_cls);
```

Send data to a peer using a specific session. TRANSPORT passes a message buffer where the data to send are stored and passes the size of this buffer. Priority is not supported at the moment. TRANSPORT can specify a send timeout and a continuation which is called when the send operation was completed. The continuation is called with the result of the operation indicating if the operation was successful or failed (e.g. due to a timeout). The `send` operation returns the amount of bytes accepted by the plugin or `GNUNET_SYSERR` to indicate immediate failure of the operation.

```
void disconnect_peer (void *cls,
                     const struct GNUNET_PeerIdentity *target);
```

Disconnect all sessions for the given peer. The plugin will lookup all sessions related with this peer, disconnect underlay connections if supported by the plugin and cleanup all session information.

```
int disconnect_session (void *cls,
    struct Session *session);
```

Disconnect a specific session for a peer. The plugin will lookup the session, terminate an underlay connection if supported and frees all information stored with the session. The function returns GNUNET\_OK on success.

```
void update_session_timeout (void *cls,
    const struct GNUNET_PeerIdentity *peer,
    struct Session *session);
```

Reschedule disconnect timeout for a session. If TRANSPORT received data with a session it notifies the respective plugin about this event so the plugin can update the inactivity timeout to prevent the session to be removed due to inactivity.

```
unsigned int query_keepalive_factor (void *cls);
```

Get information how often keep alive messages are required for this plugin. The default idle timeout configured for TRANSPORT is divided by the value returned. If a plugin requires to send keep alive messages more frequently it increases the value. This is true for example for the UDP plugin since UDP does not provide explicit disconnect events.

```
void update_inbound_delay (void *cls,
    const struct GNUNET_PeerIdentity *peer,
    struct Session *session,
    struct GNUNET_TIME_Relative delay);
```

Updates the delay for the next read operation for a given session. The delay can change if e.g. the inbound bandwidth assigned by ATS changes. This function is used to notify a plugin about a read delay that changed.

```
enum GNUNET_ATS_Network_Type get_network (void *cls,
    struct Session *session);
```

Returns the network scope a HELLO address is located in for the session passed. The network scopes available are defined with ATS including scopes for Local Area Network (LAN), Wide Area Network (WAN), loopback, Bluetooth and WLAN.

```
int check_address (void *cls,
    const void *addr,
    size_t addrlen);
```

Check if an address belongs to this peer and this plugin. A plugin-specific address is passed to the plugin and the plugin checks if this address is included in the list of addresses belonging to this peer and provided by this plugin and if the format of the address is valid. The plugin returns GNUNET\_OK if the address belongs to this peer or GNUNET\_SYSERR if the address does not belong to this peer.

```
const char * address_to_string (void *cls,
    const void *addr,
    size_t addrlen);
```

Convert a plugin-specific binary address to a human-readable string representation. TRANSPORT passes the address in binary format in combination with the address length. The plugin returns a string containing the human-readable string.

```
int string_to_address (void *cls,
    const void *addr,
    size_t addrlen);
```

Convert a human-readable string representation to a plugin-specific address. This function is used to convert addresses obtain from URLs to the plugin-specific binary representation. The plugin returns GNUNET\_OK if the address could be converted or GNUNET\_SYSERR if conversion failed.

```
void address_pretty_printer (void *cls,
    const char *type,
    const void *addr,
    size_t addrlen,
    int numeric,
    struct GNUNET_TIME_Relative timeout,
    GNUNET_TRANSPORT_AddressStringCallback asc,
    void *asc_cls);
```

Converts a plugin-specific to a human readable string. TRANSPORT passes the address and its length and specifies if a numerical conversion or a lookup for the hostname with DNS should be performed. A continuation is passed to be called with the result of the conversion.

The TRANSPORT service provides functions to the plugins to allow the plugins to notify the service about connect or disconnect events, and data received from remote peers. Plugins can retrieve the peer's identity and HELLO message from the TRANSPORT service and handles to the peer's configuration, the statistic service and the logging functionality. Functions provided by the TRANSPORT service with the plugin API are:

```
void session_start (void *cls,
    struct GNUNET_HELLO_Address *address,
    struct Session *session,
    const struct GNUNET_ATS_Information *ats,
    uint32_t ats_count);
```

Notify TRANSPORT about a new incoming session. This information is given to ATS and can be selected by ATS to communicate with this peer. The plugin passes additional performance information about the address.

```
struct GNUNET_TIME_Relative receive (void *cls,
    const struct GNUNET_HELLO_Address *address,
    struct Session *session,
    const struct GNUNET_MessageHeader *message);
```

Pass data received from remote peers to TRANSPORT. The plugin received a message from a remote peer using the given address and session. As a return value, TRANSPORT notifies the plugin about the delay for the next read operation for the session.

```
void session_end (void *cls,
  const struct GNUNET_HELLO_Address *address,
  struct Session *session);
```

Notify TRANSPORT about a terminated session. The session for the given address terminated and cannot be used anymore. TRANSPORT notifies ATS about this event and if this session is the active session currently used has to obtain a new session from ATS.

```
const struct GNUNET_MessageHeader * get_our_hello (void);
```

Returns the peer's current HELLO message to the plugin.

```
void notify_address (void *cls,
  int add_remove,
  const struct GNUNET_HELLO_Address *address);
```

Notify TRANSPORT about a new address available or an address removed with this plugin. The addresses are added to the peer's HELLO message by the HELLO component.

```
struct GNUNET_ATS_Information get_address_type (void *cls,
  const struct sockaddr *addr,
  size_t addrlen);
```

Return the ATS network scope of an address to the plugin. TRANSPORT uses NAT to figure out if a given IP address is a loopback address or located in the local LAN or in the WAN.

```
void update_address_metrics (void *cls,
  const struct GNUNET_HELLO_Address *address,
  struct Session *session,
  const struct GNUNET_ATS_Information *ats,
  uint32_t ats_count);
```

Notify TRANSPORT about updates for address properties. The plugin gets notified about performance changes for a session or an address using ATS specific data structures.

```
void register_quota_notification (void *cls,
  const struct GNUNET_PeerIdentity *peer,
  const char *plugin,
  struct Session *session);
```

Function to be called by the plugin to be notified about changes to the quota for a specific peer, plugin and session. This information is required to allow the plugin to reschedule delayed read events for sessions when quotas change.

```
void unregister_quota_notification (void *cls,  
    const struct GNUNET_PeerIdentity *peer,  
    const char *plugin,  
    struct Session *session);
```

Function to be called by the plugin to stop notifications about changes to the quota for a specific peer, plugin and session.

#### 4.3.11.3 Transport Plugins

The specific implementation of a TRANSPORT plugin depends on the transport mechanism it implements, but plugins share a common design approach to be able to inter-operate with the TRANSPORT service. Transport plugins are loaded by TRANSPORT service during TRANSPORT's startup and initialization phase. To load a plugin, TRANSPORT calls the initialization function provided by the plugin and passes an *environment* data structure containing references to the API functions provided by the TRANSPORT service and additional information like configuration and service handles. During this initialization, the plugin loads required configuration parameters from the configuration using the configuration handle passed by TRANSPORT, initializes required data structures and sets up the network communication. The plugin returns a data structure to TRANSPORT containing references to the API functions provided by the plugin. Plugins based on IP can register with the NAT library described in Section 4.3.8 to obtain information about IP addresses available on the system. The plugins notify TRANSPORT about addresses remote peers can use to connect to this peer and TRANSPORT uses these addresses to update the peer's HELLO message exchanged with other peers. Whenever the plugin is notified about a change in the set of addresses by NAT, TRANSPORT is notified about addresses added or removed.

When TRANSPORT requests a new session to a remote peer, it passes the HELLO address to use with the `get_session` function. First the plugin checks if a session for this address is already established and returns the existing session. If no session is established, the plugin creates a new session and returns a reference to this session to TRANSPORT service. The TRANSPORT service then uses this reference to pass data to the plugin to send to the specific peer. If an incoming connection is established by a remote peer, the plugin creates a new session for this connection and notifies TRANSPORT about the new session. TRANSPORT passes this information to ATS service which can decide to use this session instead of establishing an outbound session. To send data to remote peers, TRANSPORT service uses the plugin's `send` function and passes the data together with the session to use and a callback function to call with the result of the send operation to the plugin. The plugin attempts to send the data to the peer and notifies the TRANSPORT service about success or failure of this operation by calling the callback function with the result of the send operation. When the plugin receives data from a peer, it notifies TRANSPORT service about the data using TRANSPORT's `receive` function. When a session is terminated by a remote peer, the plugin notifies TRANSPORT about the terminated session so TRANSPORT service can switch to a different address or stop communicating with this peer. If the peer decides to disconnect a particular session, the TRANSPORT service instructs the plugin to disconnect this specific session and the plugin will terminate the connection corresponding to this session (for connection-oriented transport protocols) and clean up all session specific information. If the peer decides to stop communicating with a particular peer and to disconnect from this peer, the TRANSPORT service instructs the plugin to disconnect from the peer and to terminate all sessions established with this peer. Disconnecting from a

peer may occur during shutdown and when TRANSPORT is instructed to disconnect from a peer by a higher layer component. Depending on the transport mechanism implemented by the plugin, the plugin may have to implement a timeout mechanism to keep track of unused and outdated sessions. This is required to notify the TRANSPORT service to dismiss these sessions. This is in particular important for datagram-based transport plugins like the UDP plugin since these protocols do not notify the plugins with explicit disconnect events.

When the plugin has to be unloaded, TRANSPORT service calls the respective shutdown function provided by the plugin. The plugin disconnects from all remote peers and terminates all active sessions. It shuts down network communication, notifies TRANSPORT about all addresses in the set of active addresses to be removed, and frees all data structures used to store information.

#### 4.3.11.4 HELLO Management

Every peer maintains a HELLO message as described in Section 4.3.5. This HELLO message contains peer specific information including the peer's public key and information about the peer's addresses. This HELLO message is distributed to other peers using bootstrapping and neighbor discovery mechanisms. Due to the support of multiple transport mechanisms and the possibility of addresses changing over time, a central instance collecting this address information and keeping the HELLO information up-to-date is required. In GNUnet's transport infrastructure, this centralized maintenance of the HELLO message is provided by TRANSPORT's HELLO component. This component collects all information required to maintain the peer's HELLO message in a central place; other components can obtain this HELLO message from the HELLO component.

When initialized during TRANSPORT startup, the HELLO component creates a new HELLO message containing the peer's the public key provided by the TRANSPORT service. The addresses to add to the HELLO message are provided by the TRANSPORT plugins as described in Section 4.3.11.3. The plugins notify the TRANSPORT service about the address they provide and regularly update their addresses (e.g. when notified by the NAT library about a changing IP address). TRANSPORT notifies the HELLO component about any change and the HELLO component updates the HELLO message accordingly. The HELLO message is given to the PEERINFO service responsible to store validated peer information as described in Section 4.3.7.

The HELLO message contains an expiration date how long the current HELLO message and the addresses included are valid. This value is defined by a constant<sup>2</sup> and currently defined as 12 hours. The HELLO message is refreshed by the HELLO component in a regular interval to prevent the HELLO from expiring when no addresses are added or removed by the transport plugins. This interval is defined in the constant HELLO\_REFRESH\_PERIOD, currently defined with 6 hours. To refresh the HELLO, the component iterates over the list of addresses provided by the plugins and creates a new HELLO message containing these addresses with an extended expiration time.

#### 4.3.11.5 Address Validation

Before a peer can use addresses obtained in a HELLO message from other peers, the peer has to ensure that the addresses contained in this message are valid (i.e. can be used to establish a connection) and the communication partner it can connect to with these

---

<sup>2</sup> GNUNET\_CONSTANTS\_HELLO\_ADDRESS\_EXPIRATION

addresses is the peer (with the identity) it purports to have. This process is called *address validation*. When an address is validated, the peer checks that the address can be used to connect to a peer and authenticates the communication partner using a challenge/response mechanism using the public key included in the HELLO message. All addresses obtained from other peers are initially treated as *unvalidated*, no matter if they are obtained from a hostlist server operated by a (trustworthy) organization or a from a peer directly discovered using neighbor discovery or other methods mentioned in Section 4.3.6. An exception is peer information included in the GUNet software, which is by default treated as validated. Addresses are revalidated in regular intervals to ensure that they are still valid.

This address validation process is implemented in the TRANSPORT service's VALIDATION module. Other components like the HOSTLIST daemon or transport plugins supporting neighbor discovery mechanisms like the UDP or Bluetooth plugin notify the VALIDATION module about new HELLO messages received and pass the VALIDATION module the HELLO messages. When VALIDATION obtains a new HELLO message, it starts with parsing the HELLO message and extracts the public key and addresses contained in the message. For each address, VALIDATION checks the expiration time of the address and the address is discarded if the expiration time is exceeded. If the address is not expired, a validation attempt for this address is started. When an address is to be validated, VALIDATION first checks if the transport plugin required for this address for is available. When the required plugin is not available, the peers skips the address since it cannot be used with the peer's current local configuration. Before establishing a connection with this address a blacklist check, as described in Section 4.3.11.6, is performed to ensure that connecting to this peer is allowed. If blacklisting permits a connection, the peer initiates a connection to the respective address and sends a so called PING message. A PING message contains the public key of the remote peer to be validated, the address in validation and the transport plugin this address belongs to, the HELLO message of the peer sending the validation request and a 4 byte nonce value to use with the response to prevent replay attacks.

When a peer receives such a PING message it extracts the address to validate from the message and checks if the required transport plugin is available on the peer. If the plugin is available, it checks with the plugin if the address to validate is a valid address belonging to this peer. This check is done by giving the address to the plugin using the `check_address` function of the plugin API as described with the transport plugin API in Section 4.3.11.2. Each plugin internally maintains a list of addresses valid with this plugin and checks the address against this list. If the plugin confirms the validity of the address, the peer creates a validation PONG message to send to the requesting peer. This PONG message contains as a response the address and the plugin of the address to validate and the nonce included in the PING message. The PONG message contains a timestamp how long this signature is valid and is cryptographically signed with the private key of the peer creating the PONG message to ensure the authenticity of the message. This signature contains the address, the transport plugin, and the expiration time of the signature. The nonce is not included in the signature. The validation component can cache signatures and reuse them if the peer sending the validation request only attempts to re-validate an address. This reduces the number of required signing operations and can prevent Denial of Service (DoS) attacks. If any of the checks or any operation fails, the validation process is aborted and the connection with the remote peer is terminated without sending a response.

When a peer receives a PONG message, it first of all ensures the correctness of the message received and checks if a validation request for this address is pending. If no

PONG message is expected, the peer discards the message without any response. If a PONG message is expected, the peer checks the expiration time of the message and verifies the signature included in the PONG message using the public key of the peer creating the message. If the validation check of the signature was successful, the address is treated as validated. Since signature verification is expensive, the peer can cache the signature received during a successful validation process and can on subsequent validation attempts compare the cached signature with the signature received in the PONG message. This is possible since the nonce used is not included in the signature created by the peer sending the PONG. Address validation cannot protect against replay attacks from an adversary able to intercept network traffic between peers and by doing so having the possibility to obtain the required nonce for a replay attack. VALIDATION passes validated address information to the PEERINFO service responsible to store peer information in a persistent way and in addition to other components of the TRANSPORT service and the ATS service to use this information to communicate with this peer.

#### 4.3.11.6 Blacklisting Peers

In addition to explicitly *white-list* peers in friend-to-friend mode as described with the TOPOLOGY daemon in Section 4.3.9.1, it is also possible to explicitly prevent connections with peers using *blacklisting*. Before a connection to a peer is established or an incoming connection request is accepted, TRANSPORT service performs a *blacklist* check to ensure a connection with this respective peer is not explicitly denied. Blacklisting entries are maintained by TRANSPORT service's BLACKLIST component and can be configured statically by adding an explicit blacklisting entry in the peer's configuration, or dynamically at runtime, allowing clients to dynamically add new blacklisting entries. A connection to a peer can be blacklisted based on the peer identity denying all connections to this peer or by specifying a transport plugin not allowed to be used with this peer.

Every time TRANSPORT is instructed to initiate a connection, a check against the blacklist is performed. If the peer is blacklisted, a connection to this peer is denied and the connection attempt aborted. The same applies to incoming connections from other peers: if a new inbound connection is established from a remote peer, a blacklist check is performed before the connection is accepted and established with the remote peer. If the connection is denied by BLACKLIST, the connection is terminated and the peer does not respond to the other peer.

#### 4.3.11.7 Neighbor Connections

The most important functionality for the transport infrastructure and the TRANSPORT service is to establish and maintain connections with remote peers. In the TRANSPORT service, the NEIGHBOR component is responsible to establish, maintain and disconnect connections in close collaboration with the TRANSPORT plugins. The NEIGHBORS component is responsible for establishing outbound connections with remote peers as well as for accepting incoming connections. The NEIGHBOUR component interacts with the BLACKLISTING component (described in Section 4.3.11.6), to check for every connection being established or accepted if a connection with this peer or using this transport plugin is allowed or has to be denied. To maintain a connection, the NEIGHBOUR component sends periodic keep-alive messages to ensure a connection stays established. The basic notion in this component is the idea of a *neighbor* a higher layer application wants to communicate with and NEIGHBOR's task is to establish a connection to exchange data with this communication partner. Here the higher layer application does not have to know *how* (i.e. using

which transport plugin or address) the connection is established, the application is only interested that it is possible to exchange data with this peer. The NEIGHBORS component therefore provides the abstraction between communication with a peer and the specific mechanism used. A second aspect of this notion is that the NEIGHBORS component only establishes connections with peers higher layer applications explicitly want to communicate with but it does not decide which peers to communicate with. This decision is met by the higher layer applications and in particular the TOPOLOGY daemon, described in Section 4.3.9. In addition, the NEIGHBORS component does not decide *how* to communicate with a remote peer. This decision is found by the ATS service responsible to manage available addresses and to find a decision which communication mechanism to use. ATS is responsible for automatic transport selection and is described in detail in Chapter 5.

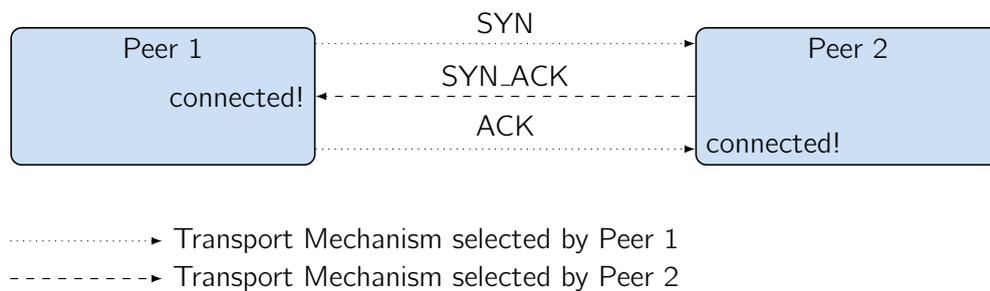
The focus of TRANSPORT's NEIGHBOR component is to establish connections to a remote peer on request of external components or higher layer applications using the transport mechanism suggested by the ATS service. For every neighbor peer a connection is requested with, NEIGHBOR creates a neighbor instance used to store information related to this peer, for example the address provided by the ATS service to communicate with this peer, an additional alternative address to allow a fast switching between addresses and the current state of the connection.

According to the peer-to-peer paradigm, peers in a peer-to-peer network can establish and terminate connections with each other without coordination. Peers can act autonomously, so two peers can begin to connect to each other at the same time. GUNet's transport infrastructure only uses local decisions in combination with a *sender decides* approach. Every peer decides locally how to communicate with a remote peer and chooses the *best* transport mechanism and address available. This has the implication, that peer *a* can send data to peer *b* using transport mechanism *x* while peer *b* can decide to send data to peer *a* using transport *y*. In addition, both peers can even switch mechanisms and addresses while communicating with each other. The TRANSPORT NEIGHBOR component therefore employs a three-way connection establishment mechanism to ensure that connections are established correctly in combination with a state machine representing the current state for every neighbor the component is interacting with.

Such a three way handshake is required since every peer can decide on its own which address to use and since it is not assured that the address is working at the moment and the peer is willing to accept a connection using this address or mechanism. The three way handshake is depicted in Figure 4.4. With this three way handshake, the initiating peer sends a SYN message to the remote peer to indicate that it wants to establish a connection using this address. A peer receiving a SYN message checks with the blacklist if it accepts a connection from this peer and decides on its own which mechanism and address to use to respond to the SYN message. When it accepts the connection attempt, it replies with a SYN\_ACK message using the address suggested by ATS as the best address available. When the initiating peer receives a SYN\_ACK from a peer, it knows that the remote peer received the connection attempt and accepts the attempt using this address. The initiating peer now has to acknowledge the SYN\_ACK message using an ACK message to indicate that it received the SYN\_ACK from the remote peer and that it accepts the address selected by the remote peer.

#### 4.3.11.8 The Neighbor State Machine

Establishing and maintaining connections to remote peers, switching addresses when a connection fails, reconnecting peers and performing a controlled disconnect with a peer



**Fig. 4.4:** The TRANSPORT Service Three Way Handshake

while at the same time accepting incoming connection attempts from a remote peer is a complicated task to manage for the NEIGHBOR component.

The TRANSPORT NEIGHBOR component implements a state machine to track the state of every neighbor it communicates with. This state machine allows the NEIGHBOR component to react to every network and application event according to the peer's current state. This state machine is realized as a Deterministic Finite Automaton (DFA): the state machine provides a finite set of states and a dedicated starting state (NOT\_CONNECTED) when the peer is created and a final state (DISCONNECT\_FINISHED), reached when the peer is fully disconnected. Every transition between states is caused by an event (like a network or application event or a timeout) and triggers an action (like sending a message or terminating the connection).

Each neighbor in the NEIGHBOR component provides its own state machine and every neighbor is during its lifetime in a specific state of the state machine. When a neighbor transitions to a new state in the state machine, specific methods exist to perform a state transition and modify the state of the neighbor. This allows extensive monitoring of the transition events in the NEIGHBOR component for every single neighbor. The API to monitor such neighbor activity is described in Section 4.3.11.15

To check for timeouts, NEIGHBOR uses a *master* task, checking every neighbor in regular intervals. If a timeout occurred for a peer, the master task triggers the required action to respond to timeout situations depending on the current state. The NEIGHBOR state machine is depicted in Figure 4.5 and provides the following states:

**NOT\_CONNECTED:** is the start state of the DFA. When a NEIGHBOR entry is created due to a connection request or a connection attempt from a remote peer, the neighbor is initialized with this state.

**INIT\_ATS:** when a peer attempts to initiate a connection to a remote peer, it requests an address to use from the ATS service and waits in this state for the address suggestion. Since ATS only suggests an address if a suitable address is available, this state can time out when no address is available.

**SYN\_SENT:** when the peer tries to establish an outbound connection and an address was suggested by ATS, the peer sends a SYN message to the remote peer using the suggested address. After sending this message the neighbor changes to this state and waits for a reply from the remote peer.

**SYN\_RECV\_ATS:** after receiving a SYN message from a remote peer, the receiving peer requests an address from ATS to communicate with the remote peer according to the *sender decides* principle. This state indicates that for this neighbor an address suggestion to continue is expected.

**SYN\_RECV\_ACK:** when a peer after receiving a SYN message acknowledges this message using a SYN\_ACK message to indicate that it received the SYN message, it changes to SYN\_RECV\_ACK and expects the ACK to complete the three way connection establishment.

**CONNECTED:** When a peer initiating an outbound connection has sent a SYN and received a SYN\_ACK message, the peer connection is established and the peer is connected. For a peer receiving an inbound connection and receiving the ACK, the connection attempt is successful and the peer is connected.

**RECONNECT\_ATS:** When the active session used to communicate with a peer is terminated, the peer tries to reestablish the connection as described with fast reconnect in Section 4.3.11.11: after the active session was terminated the peer now expects an address suggestion from ATS in this state.

**RECONNECT\_SENT:** When reconnecting and ATS provided an alternative address to use, the peer sends a SYN message to indicate a connection attempt and waits in this state for a response.

**SWITCHING\_CONNECT\_SENT:** When a peer is connected and ATS suggests to switch to a new addresses, the peers sends a SYN message and waits in this state for the response from the remote peer.

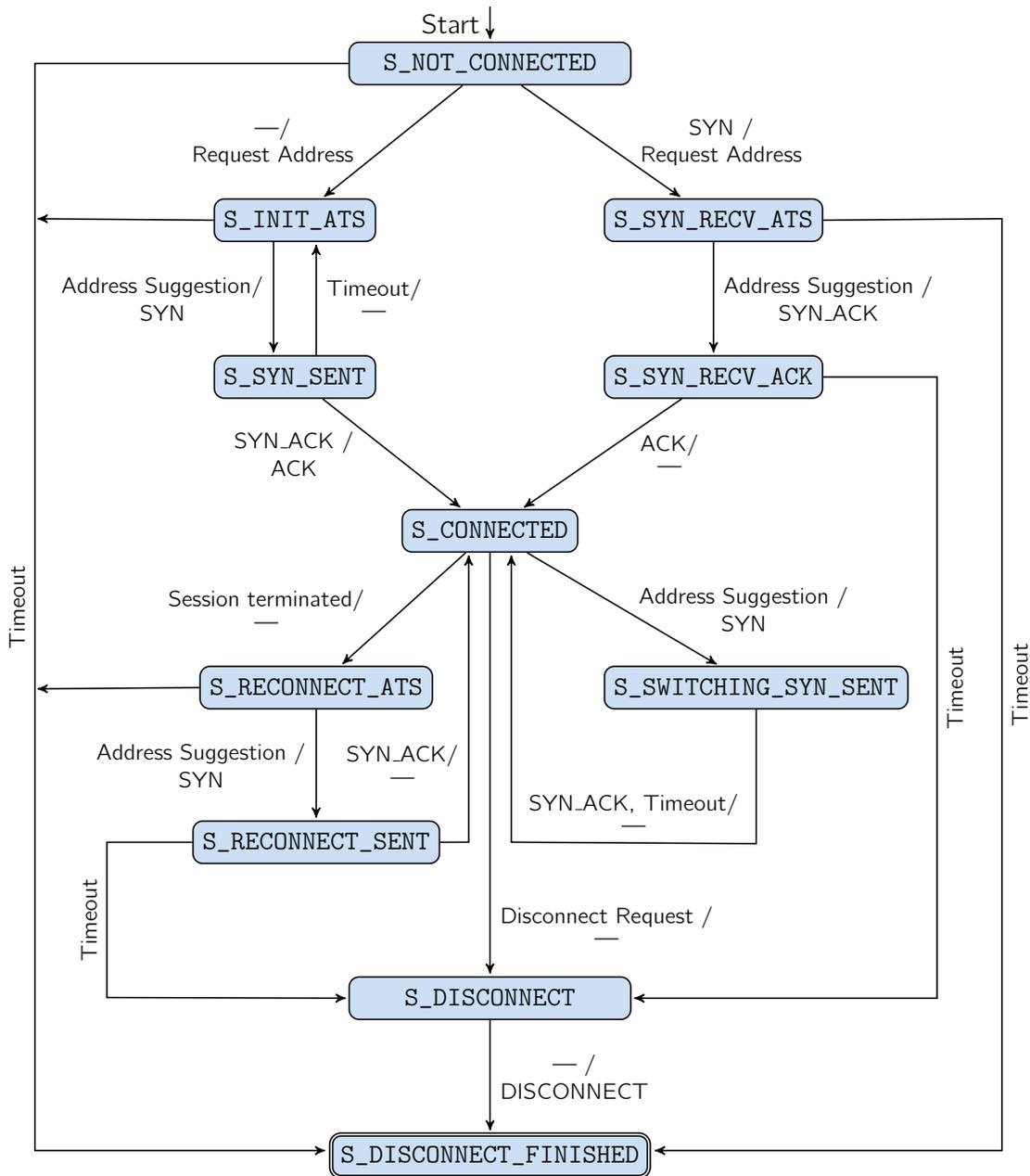
**DISCONNECT:** This state indicates a disconnect operation in progress, as described in Section 4.3.11.12, and the peer tries to notify the remote peer by sending a DISCONNECT message.

**DISCONNECT\_FINISHED:** The disconnect process for this peer is finished and in the next step the neighbor entry will be freed. This is the terminal state of the life cycle for a neighbor.

#### 4.3.11.9 Establishing Outbound Connections

When an external component like the TOPOLOGY daemon instructs the TRANSPORT service to establish a connection with a peer, TRANSPORT first checks if an overlay connection is already established and if not in the next step performs a blacklist check to ensure that establishing a connection to this peer is permitted. If a connection to the peer is permitted, the NEIGHBOR component is notified about the request and a new neighbor object for the respective peer is created and initialized. To establish the connection, an address is requested from the ATS service. When ATS provides an address, NEIGHBOR tries to establish a connection with the peer using the address provided from ATS. Since connections can be blacklisted based on the transport plugin used, a second blacklist check is performed for the suggested address. If the blacklist check for the suggested address permits a connection, the NEIGHBOR can initiate the connection with peer. If the use of this address is denied by the BLACKLIST component, the address is removed, ATS is instructed to delete this address and TRANSPORT waits for an alternative address to be suggested by ATS.

If ATS provides an address including a session, an underlay connection was already established before (due to a validation event or an incoming connection attempt). If no session is provided, the NEIGHBORS component asks the respective transport plugin to create a new session for the provided address.



**Fig. 4.5:** The TRANSPORT Service State Machine

To establish an outbound connection the NEIGHBORS component follows the three-way handshake protocol described in 4.3.11.7: it first sends a SYN message using the plugins send API function and passing the session to use with this call. When the remote peer receives this session, it acts as described in Section 4.3.11.10 to handle incoming connection requests. The peer sending the connect request waits for the SYN\_ACK message as an acknowledgment from the remote peer. When it receives the SYN\_ACK message, it marks the peer as connected and responds to the remote peer with a final ACK to indicate that it received the SYN\_ACK message. When the neighbor is marked as connected, the NEIGHBOR component notifies the TRANSPORT service about the successful connection

event which notifies the clients using the TRANSPORT CLIENT component described in Section 4.3.11.14 about the peer being connected.

#### 4.3.11.10 Accepting Inbound Connections

When a peer receives an incoming connection attempt from a remote peer and is not yet connected to this peer, in the first step a new neighbor entry is created and a blacklist check is initiated to check if an incoming connection from this peer is allowed. If the connection attempt is denied by BLACKLISTING, the NEIGHBOR component terminates the connection without responding to the remote peer. If BLACKLIST permits the connection, an address to communicate with this peer is requested from ATS service. When ATS suggests an address to use, a second blacklist check for the address is performed. This is required since the first check was only for the peer and this additional check ensures that a connection with this specific transport plugin is allowed. If the check for the peer and the plugins permits the connection, a SYN\_ACK messages is sent to the remote peer to acknowledge the connection attempt. Otherwise the address and the session are destroyed with the ATS service, the plugin is instructed to disconnect the session and NEIGHBORS expects ATS to suggest an alternative address. The remote peer receiving the SYN\_ACK message responds with an ACK to acknowledge that it received the SYN\_ACK message. When the peer receives this ACK message, the peer is marked as connected and the TRANSPORT services notifies higher layer clients about this event using the CLIENT component.

#### 4.3.11.11 Switching Connections and Fast-Reconnect

GNUnet's TRANSPORT service relies on the *sender decides* principle to have every peer locally select the best transport mechanism to communicate with a remote peer. Based on this principle the sender decides which plugin, address and session to use and can reconsider this decision and switch to a new transport address if it has evaluated that an alternative mechanism provides better properties.

When a connection to a remote peer is established and the peer (and ATS in particular) decides to switch to an alternative address, TRANSPORT first checks with the blacklist if a connection with this address is allowed. If BLACKLIST permits this connection, a SYN message is sent to the remote peer using the new address. The remote peer performs a blacklist check for the new connection based on the address. If the blacklist check permits the use of this address, the remote peer sends a SYN\_ACK message to acknowledge the SYN message and waits for a ACK as an acknowledgment from the peer initiating the address switch that the switch operation was successful. Now the active address used by the peer initiating the address switch is switched to the new address and both peers are still connected. To prevent a disconnect in case the address switch fails, the NEIGHBOR component stores the previous address used: when the switch operation fails, it discards the alternative address and reactivates the address used before.

If the session used to communicate with a peer is disconnected (e.g. due to a network error), the peer is not immediately marked as disconnected and clients are not immediately notified about the disconnect event. Instead a process called *fast reconnect* takes place with the goal to re-establish the connection between the peers: the peers removes the disconnected address, notifies ATS to delete this address and waits for ATS to suggest an alternative address. If ATS provides can provide an alternative address it reconnects to the peer without notifying clients about this event using the three-way handshake process to initiate an outbound connection. This fast-reconnect approach provides the illusion

of stable connections for higher layer components even under conditions with unstable network connections.

#### 4.3.11.12 Disconnecting Connections

Besides peers getting disconnected due to errors and network failures or ATS decisions, peers can also decide to actively disconnect from a remote peer. When a peer is to be disconnected, NEIGHBORS instructs the plugins to disconnect all sessions established with this peer. For the address used to communicate with the remote peer, an active disconnect message with a notification for the remote peer is transmitted to prevent the remote peer from re-establishing the connection (e.g. using fast reconnect): the disconnecting peer sends a DISCONNECT message to the remote peer containing a cryptographically signed timestamp. This signed timestamp provides authenticity for the remote peer and prevents DoS attacks and adversaries disconnecting peers by sending arbitrary DISCONNECT messages to other peers. A peer receiving a DISCONNECT message checks the signature and the timestamp and disconnects the peer if the check was successful. If for this peer an active address is available, TRANSPORT instructs ATS to delete this address and the respective plugin to disconnect from this peer and to terminate all sessions. This active disconnect process immediately notifies a remote peer about a disconnect and prevents the remote peer from performing expensive reconnect operations.

#### 4.3.11.13 Enforcing Resource Constraints

GNUnet may not be the only application running on a system and may have to share network resources with other applications. To prevent other applications from being cannibalized by GNUnet trying to maximize its performance with other peers, the user may want to limit the resources consumed by GNUnet. GNUnet's transport infrastructure therefore allows the user to configure restrictions on the amount of bandwidth used to communicate with other peers. The user can configure *quotas* for available *network scopes* (WAN, LAN, WLAN, ...), specifying how much bandwidth GNUnet is allowed to consume both for inbound and outbound communication.

Resource restrictions are enforced in the TRANSPORT service on a per-peer basis. Every peer gets a certain amount of resources assigned by ATS and when this share is depleted, TRANSPORT delays future send and receive operations with this peer.

Outbound quotas are enforced within the TRANSPORT API used by clients to send data to peers: a client requests to send a certain amount of data to a peer and the API calls a callback function to notify the client when it is ready to send. This callback mechanism is coupled with a bandwidth tracker, delaying the callback when too much traffic is sent to enforce the bandwidth constraint.

For incoming traffic the inbound quota is enforced by the TRANSPORT service delaying the next receive operation for the plugins. When plugins pass data received to TRANSPORT, TRANSPORT notifies the plugins how long to delay the next receive operation. For transport mechanisms using protocols providing flow control (e.g. TCP) the next receive call can be easily delayed to throttle the amount of data received. For plugins using protocols not providing flow control (e.g. UDP) this is not possible without peers collaborating since reads cannot be delayed.

For mechanisms not providing flow control, we provide a collaborative approach: the receiving peer notifies the sender about the current receive rate it is willing to accept. The sender has to obey this rate or otherwise the receiver can drop messages exceeding the receive rate. Please note, that this approach does not violate the objectives to not

share internal information with other peers and not require other peers to collaborate: the receiving peer can drop messages if the sender does not collaborate and the decision about the receive rate is found locally without collaborating with a remote peer to find this decision. Therefore, this is a voluntary collaboration of the sender who will be otherwise punished if it does not obey the receiver's transmission rate constraint. This approach leaks only little information to other peers: the send rate provided to the remote peer only refers to the resources assigned to this peer and not the amount of resources the peer can overall provide.

#### 4.3.11.14 Client Management

The management of higher layer clients is done by the TRANSPORT's `client` component. Clients wanting to interact with the TRANSPORT service use the TRANSPORT API described in Section 4.3.11.15 to interact with the TRANSPORT service. The CLIENT component manages incoming connections from higher layer clients and maintains a client context for each client. It forwards client requests to the respective TRANSPORT component responsible to process the request and returns the result of the operation to the client.

#### 4.3.11.15 Interacting with the Transport Service

To allow other GUNet components to access the functionality of the TRANSPORT service, the service provides an extensive API. This TRANSPORT API provides functionality to use the TRANSPORT service to send and receive data from remote peers and in addition provides functionality to retrieve information from the service, its internal state and to access extended functionality of the service. Other GUNet services and daemons as well as CLI tools can use this API and the library to interact with the service. The library establishes an IPC connection to the service, sends the requests to the service and receives the responses.

Applications using the TRANSPORT service to communicate with remote peers use the TRANSPORT API to get notified about peers connecting and disconnecting and to send data to a remote peer. Such a client uses the API to connect to the TRANSPORT service and TRANSPORT notifies the client about all peers currently connected and whenever a peer connects or disconnects. The client can use this connection to instruct TRANSPORT to connect to a peer, to transmit data to a peer and the client receive data received from remote peers. Clients interacting with TRANSPORT to communicate with remote peers only use the peer's peer identity to refer to a remote peer. Here the abstraction between the transport underlay and the overlay is achieved since clients are not aware which kind of underlay communication is used. GUNet peer-to-peer applications are not intended to use the TRANSPORT service directly, but use instead the CORE service providing link encryption between peers. The API provides functionality to pass HELLO messages to the service for validation, a functionality commonly used by bootstrapping mechanisms like the HOSTLIST component, and functions to add new blacklist entries for peers and to manipulate traffic properties for certain peers for testing purposes.

The TRANSPORT API provides extensive monitoring functionality to obtain information about the state of TRANSPORT. It can be used to obtain information about peers currently connected and the state and address of the current connection as well as information about validation processes pending for addresses for remote peers. Additional functions provide address-to-string conversion to provide human-readable address strings to applications.

#### 4.3.12 The UNIX Domain Socket Transport

On UNIX systems, UNIX domain sockets are an IPC mechanism for inter-process communication within the same host operating system. UNIX domain sockets provide both connection-oriented (*stream*) and connection-less (*datagram*) communication between processes. UNIX domain sockets exist on all POSIX compliant operating systems. Addresses and endpoints for UNIX domain sockets are represented as files within the file system but communication takes place directly in the kernel of the operating system. Linux systems in addition support *abstract unix domain sockets*. These abstract sockets use an *abstract* namespace independent from the file system, not restricted by permissions in the file system and accessible also from `chroot` environments. For testing purposes, we provide a UNIX domain socket TRANSPORT plugin to connect peers on the same system.

The UNIX domain socket plugin creates a new UNIX domain socket in datagram mode and uses this socket to communicate with other peers on the same system. To use UNIX domain sockets, the Berkeley `socket` interface is used and therefore communicating with UNIX domain sockets is similar to TCP or UDP. The address of an UNIX domain socket is represented as a file in the namespace of the file system. The maximum length of an address for a UNIX domain socket is defined by `UNIX_PATH_MAX`, 108 chars on a Linux system. An abstract socket is represented by an '@' as the first char of the name. The name of the socket to be used with the plugin is configured in the plugin configuration and the socket is created in datagram mode during startup. The plugin supports the use of abstract UNIX domains sockets if enabled in the configuration. The address of this socket is reported to the TRANSPORT to be exchanged with other peers. To send and receive data, the plugin runs in a select loop, notifying the plugin when it is possible to send data or when data are received from remote peer. Messages transmitted with the UNIX domain socket plugin are encapsulated in a plugin specific message format containing the peer identity of the originating peer to allow the receiving peer to demultiplex the messages according to the origin of the data. When a peer receives data from a peer and a UNIX domain socket for the first time, the plugin creates a new address session, identified by the peer identity and the address of the UNIX domain socket and notifies the transport infrastructure about the new session. To send data to a peer, the TRANSPORT can also explicitly request a session based on a HELLO address of a remote peer. Since the plugin uses connection-less datagrams, the plugin needs to implement a timeout logic to notify the transport infrastructure about inactive sessions to be removed due to inactivity. The benefit of UNIX domain sockets is very high throughput. This plugin is intended for testing purposes only since it allows to communicate only with other peers running in the context of the same operating system.

#### 4.3.13 The TCP Transport

To establish connections between peers not located on the same system, GUNet provides the TCP plugin. TCP, introduced in Section 2.3.1, is with UDP one of the standard transport protocols used on the Internet. It provides connection-oriented and reliable communication between systems. TCP protects against packet loss using acknowledgments for transmitted data fragments and in addition supports flow and congestion control. For GUNet, the TCP plugin provides TCP based communication between participants in the peer-to-peer network supporting both IPv4 and IPv6.

To allow remote peers to establish a connection, TCP plugin interacts with the NAT library to obtain the set of IP addresses available on the system and notifies the TRANSPORT service to include these addresses with the peer's HELLO message. When

a new connection is initiated, the plugin first sends a `WELCOME` message containing the peer's identity to allow the remote peer to demultiplex the data received based on the peer's identity and the source address the data originate from. Since with TCP a connection is originating from a randomly selected high port, which cannot be used to establish a connection to the respective peer, incoming connections are marked with the `GNUNET_HELLO_ADDRESS_INFO_INBOUND` flag in the option field of the `HELLO` address passed to the `TRANSPORT` service to prevent the `TRANSPORT` to use this incoming address to initiate a connection to the peer. In the TCP plugin, an address is the socket address provided by the plugin and a session is identified by peer identity and the socket address. Since TCP is connection-oriented, the plugin is explicitly notified about network-level disconnects and therefore only timeouts for unused sessions have to be implemented.

To cope with the restrictions imposed by NAT and techniques limiting connectivity, the TCP plugin supports several techniques to improve connectivity between peers. The plugin attempts to circumvent NAT restrictions using the NAT library and its ICMP and UDP hole punching mechanisms as described in Section 4.3.8. If a remote peer indicates in his address that it is located behind a NAT device, the peer tries to apply the hole punching mechanisms to connect to this peer. The plugin also supports connection reversal, where the peer behind NAT is instructed by a peer reachable on the Internet to establish a connection to this unrestricted peer. With these approaches, the TCP plugin can increase connectivity even for peers with restricted connectivity due to NAT devices or other techniques making it normally impossible to connect to these peers. The plugin also supports a listen-only mode for clients not able to accept any connections. When using this mode the plugin does not provide any addresses to `TRANSPORT` and is restricted to only initiate connections and as it does not listen for inbound connections.

With the TCP plugin, we can provide a reliable data exchange between peers using a high performance communication protocol. By relying on the functionality of the NAT library providing NAT traversal support and connection reversal, we can provide increased connectivity for clients in restricted environments. Since TCP requires a separate network socket for each connection, it is not suited for systems with limited network resources.

#### 4.3.14 The UDP Transport

Besides TCP, providing connection-oriented communication, the second widely used transport protocol on the Internet is UDP introduced in Section 2.3.1. With respect to its properties, UDP is the counterpart to TCP, providing connectionless and unreliable communication without retransmissions, flow or congestion control. So when peers exchange data with UDP, no connection is established, UDP packets are sent with a *fire and forget* attitude without acknowledgments for successful transmissions, re-transmission when packets are lost or congestion and flow control. With UDP, a datagram can only have a specific maximum size, so the size of the payload transmitted with a UDP datagram has to obey this limit. Since UDP is not connection-oriented, a plugin using UDP requires only a single socket on a system, making UDP suitable for the use on systems with scarce resources and in cases where a peer has to communicate with a large number of peers, for example in network experiments and to test scalability of the peer-to-peer system.

The UDP plugin provides UDP-based communication for GNUnet. Like the TCP plugin, the UDP plugin interacts with the NAT library to obtain the set of available addresses on the system. The plugin supports both IPv4 and IPv6. The restriction for the maximum UDP datagram size, the MTU, is about 1500 bytes, whereas GNUnet supports by default a maximum message size of 64 KiB. To circumvent this limitation of UDP,

the plugin implements a transparent fragmentation and defragmentation mechanism for messages with a size larger than the UDP MTU. Datagrams smaller than the MTU are transmitted without fragmentation. Together with (de-)fragmentation, the plugin implements an adaptive acknowledgment mechanism for message and provides automatic retransmissions when messages are lost. Since UDP provides neither congestion nor flow control, a sender can overload a receiver by sending faster than the receiver can read and process the data. Therefore, the UDP plugin implements a congestion control mechanism to communicate to sender to send faster or slower by including a delay value for the next transmission process in the acknowledgments for the sender. To prevent the communication link and partner from overloading and prevent congestion, the UDP plugin implements a congestion control mechanism. The retransmission mechanism responsible to retransmit lost message in case of missing acknowledgment messages uses adaptive retransmission interval, decreasing the retransmission frequency when messages are not acknowledged.

In addition, the UDP plugin provides neighbor discovery to discover new neighbors in the network. The UDP plugin regularly broadcasts the peer's HELLO message to other peers. UDP obtains the peer's current HELLO from the HELLO component using the API function `get_our_hello` provided by TRANSPORT. Neighbor discovery with UDP uses IPv4 broadcast and IPv6 multicast. With IPv4 broadcasts, the plugin sends the peers HELLO beacon on all active network interfaces (except the loopback interface) to the network's broadcast address using the port configured for the UDP plugin. Other peers configure their listen socket to receive broadcast messages on the respective port. To receive beacon messages, it is required for all peers to use the same UDP port. Peers listening for broadcast messages receive these beacon messages and forward them to the TRANSPORT service, which validates the information in the HELLO message and may then use the HELLO to establish a connection to remote peer.

With IPv6, the IPv6 protocol defines special multicast addresses with a special *scope* as defined in [HD06]. The scope of the address restricts the *range* of multicast messages. Four scopes are defined with [HD06]: interface-local, link-local, admin-local and site-local. For GUNet's neighbor discovery, we registered with IANA the variable scope multicast address "FF0X::13B"<sup>3</sup>. Peers with IPv6 and neighbor discovery enabled configure their UDP socket to join the respective multicast group to send and receive messages with this multicast group. The UDP plugins sends in regular intervals the peer's HELLO beacon to the multicast group using the site-local address FF04::13B. Other peers join this multicast group to receive these beacons. As with IPv4, beacons are forwarded to the TRANSPORT services for validation and can then be used to establish connections to the remote peers.

### 4.3.15 The HTTP(S) Transport

To make GUNet traffic resilient against filtering and traffic inspection and to improve connectivity for clients in environments requiring the use of proxy servers to access the Internet, we provide the HTTP- and HTTPS-based transport plugins. HTTP and its secure variant HTTPS are the protocols used with the World Wide Web (WWW) to transfer data between clients and web servers. HTTP is an application layer protocol and relies on an underlying transport protocol. The most common transport protocol is TCP, but it is possible to implement HTTP over other transport protocols like UDP [Gol99] or SCTP [NALB10] as described in [FGM<sup>+</sup>99]. HTTP over UDP is sometimes referred

<sup>3</sup> <https://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml>

to as *HTTTPU* and HTTPU is used as part of the UPnP implementation. HTTP is a client/server based request-response protocol: a client requests a resource from a server and the server responds with the result of the operation. A server provides resources and a client can access these resources by addressing resources using an URL. HTTP is intended to be a stateless protocol: a server does not maintain a state for a client over multiple requests. HTTP defines different methods to indicate the required action to the server. The most common methods are the GET method, used to request a resource to be transferred to the client, POST and PUT, used to transfer data from the client to the server. To indicate the status of an operation HTTP defines *status codes*<sup>4</sup>: the first, most significant digit of the three-digit code represents the class of code, followed by two digits identifying the specific code. Two versions of the HTTP protocol are currently used on the Internet: HTTP 1.0, defined in [BLFF96] and HTTP 1.1, defined in [FR14], [Res99] et al.. HTTP 1.1 added five new methods and additional important features such as *chunked transfer encoding* to transfer data in *chunks* without specifying the length of the content to transfer in advance and *pipelining* to re-use an existing connection over multiple requests.

With HTTP, traffic is transferred in plain text and is not encrypted nor authenticated, so everyone able to intercept the traffic can read the content transferred and perform man-in-the-middle attacks to alter the data. To counteract this issue, HTTPS encrypts communication using HTTP on top of Secure Sockets Layer (SSL)/TLS. SSL/TLS encrypts communication channels based on the X.509 security infrastructure. SSL/TLS uses X.509 certificates and asymmetric cryptography to provide authenticity and to perform a key exchange of a symmetric session key used to encrypt communication between partners. SSL, defined in [FKK11] and TLS, defined in [DR08], are located on layer 5 and 6 of the ISO/OSI layer: they use a transport layer protocol for transmission, most commonly TCP, and are used by application layer protocols for secure communication. Due to the layered approach used with HTTPS, all HTTP traffic including headers, cookies and status code is encrypted when using HTTPS: no HTTP information is leaked from the HTTPS session.

To make a peer-to-peer communication infrastructure resilient against degradation and censorship attempts, using HTTPS as a transport protocol has interesting benefits: one the hand, communication is encrypted, so it does not make sense for an eavesdropper to intercept traffic transmitted between peer-to-peer communication partners. On the other hand is HTTPS unsuspecting since it is one of most commonly used protocols used on the Internet. Depending on the country, HTTPS traffic will be with a high probability not suppressed, filtered or degraded by an ISP, since few ISPs want to get on their customers bad side by censoring their online banking traffic. In environments enforcing the use of a proxy server to access the Internet, as described in Section 2.6.6, the use of HTTPS can also improve connectivity since HTTPS traffic is commonly not handled and cached by proxy servers. Both the HTTP and HTTPS plugins support the use of reverse proxies making filtering of peer-to-peer traffic is even harder, when peer-to-peer traffic relayed over a well-known web site to a peer in the peer-to-peer network.

But HTTPS is not an universal remedy against filtering and traffic management attempts. HTTPS relies on an underlying transport protocol and a censor can make assumptions about the traffic looking at the connection information of the transport protocol. For example if the connection uses a transport protocol or protocol port commonly used by a disagreeable application, a censor can prevent this traffic and track down a user based on these connection information. And even without having the possibility to know the

<sup>4</sup> <https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml>

data encrypted in the traffic, a possible censor can conclude on the kind of traffic included in the encrypted data based on traffic analysis and statistical analysis. Various different properties of the encrypted traffic like packet inter-arrival times, connection duration, traffic direction can be analyzed and compared with known traffic patterns of agreeable or disagreeable applications. Using this traffic analysis an eavesdropper can even obtain information about the information encrypted in the traffic, as for example described for Secure Shell (SSH) and Skype. [AZH09, HJ10]

GNUnet provides a HTTP(S) transport plugins to enable transmission of peer-to-peer traffic encapsulated in HTTP and HTTPS traffic and benefit from the features described before. Since HTTP(S) is a client/server based request-response protocol, the functionality of the plugin is split into a client part, the `http(s)-client` transport plugin and a server part, the `http(s)-server` plugin. The `http(s)-client` plugin is responsible to initiate connections to a server and to request resources from the server. The `http(s)-server` plugin is listening for incoming client connections and serves resources to the client. A peer can run both plugins at the same time. Both parts of the plugin are realized using well-tested HTTP libraries: the `http(s)-client` plugin uses a `libgnurl`<sup>5</sup>, a fork of the well-known `libcurl`<sup>6</sup> library and the `http(s)-server` plugin uses GNU `libmicrohttpd`<sup>7</sup>, a minimal web server library written in C optimized for performance. Both the HTTP and HTTPS plugin use HTTP 1.1 since this version of the protocol provides functionality like *chunked encoding* and *connection pipelining* and provide full IPv4 and IPv6 support.

To initiate a connection with the HTTP(S) plugin, a peer running the `http(s)-client` plugin asks the plugin to initiate a connection to a peer running the `http(s)-server` plugin. Since HTTP is a request/response based protocol with unidirectional communication on HTTP layer, the plugin needs to provide a bidirectional way to exchange data between client and server. Therefore, the client sends two requests to the server: a HTTP GET request to allow the client to receive data from the server and a HTTP PUT request to send data from the client to the server. The client initiates both the GET and the PUT request and includes the peer's identity in the request URL and a unique tag, to enable the server to match GET and PUT to a single connection. When establishing multiple HTTP requests to a single server, the plugin benefits from HTTP pipelining provided by HTTP 1.1, allowing the client to send multiple HTTP requests over a single TCP connection. For both the GET and PUT request, the client uses chunked encoding to send the data in chunks since the amount of data to be transferred is not known a priori.

The `http(s)-server` initially accepts all connections if the current number of connections is below a configured threshold. HTTP requests have to contain a well-formed request URL containing the client's peer identity and the tag to match GET and PUT request. If the URL is not well-formed, the connection is terminated with HTTP status 404 ("Not Found"). If the URL is well-formed, the server waits for both the PUT and GET connection to be established. If the both requests are not established within a reasonable time (`HTTP_SERVER_NOT_VALIDATED_TIMEOUT`  $\cong$  15 seconds), the requests are terminated with HTTP status 404 ("Not Found"). If both requests are successfully connected, the timeout is increased to (`HTTP_SERVER_SESSION_TIMEOUT`  $\cong$  5 minutes). If no data are transferred within this time out with either the GET or the PUT request, both requests are closed and the transport session is terminated due to inactivity. This timeout is restarted for both requests when data are send or received using one of the

---

<sup>5</sup> <https://gnunet.org/gnurl>

<sup>6</sup> <http://curl.haxx.se/libcurl/>

<sup>7</sup> <https://www.gnu.org/software/libmicrohttpd/>

requests. When both connections are established, the peers can exchange data from the client to the server using the PUT request and from the server to the client using the GET request. The client plugin, initiating the connection, has similar timeouts: (`HTTP_CLIENT_NOT_VALIDATED_TIMEOUT`  $\cong$  15 seconds) for both requests to connect to the server and (`HTTP_CLIENT_SESSION_TIMEOUT`  $\cong$  5 minutes) for idle connections. If one of the peers disconnects or the transport service requires a session to be terminated, the client will terminate the connection using a disconnect, the server will end the chunked-transfer sending a “Thank you!” message with HTTP status 200 (“OK”) to the client indicating a successful transmission.

The `http(s)-server` module can be configured to run with IPv4, IPv6 or both in dual-stack mode. When started, the server plugin obtains from the NAT library the addresses it is reachable under and reports these addresses to the `TRANSPORT` service. Here `http(s)-server` addresses are manipulated to not contain the `http(s)-server` plugin as origin, but to contain the `http(s)-client` plugin to allow clients to use this address to connect to other peers. An address contains the URL the `http(s)-server` plugin is reachable under. In URLs only IP addresses are used to prevent censorship based on making DNS names unavailable. An exception is when the user explicitly configures a hostname to use, for example when a reverse proxy is used to access a peer or a valid HTTPS certificate is available.

A peer running the `http(s)-server` HTTPS plugin automatically creates a self-signed X.509 certificate during startup. A peer in possession of a valid X.509 issued by a valid CA can provide this certificate to clients to prevent man-in-the-middle attacks. To enable certificate validation, only the name used as common name in the certificate is used as an address. This external hostname can be configured in the `http(s)-server` plugin configuration. In the plugin specific address, the `HTTP_OPTIONS_VERIFY_CERTIFICATE` flag is set to signal other peers that the certificate has to be validated when connecting. When initiating a connection to a peer providing a valid certificate, the `http(s)-client` validates the certificate when connecting and terminates requests if validation fails.

To prevent censorship based on transport protocol information, the `http(s)-server` provides HTTP(S) reverse proxy support. Contrary to forward proxies, described in Section 2.6.6, HTTP(S) reverse proxies are used to forward requests from a HTTP(S) server to a secondary server to handle the request. Reverse proxies are commonly used to distribute load between HTTP(S) web servers or to have a front end server directing expensive requests to back end (*downstream*) servers. Many web servers like Apache<sup>8</sup> or Nginx<sup>9</sup> can be configured to act as a reverse proxy forwarding requests to downstream server. If a client requests a resource not forwarded on the server, this request is handled by the server itself. If a forwarded resource is requested, the web server itself initiates a connection to the downstream server and forwards the data it receives to the client. For GUNet, reverse proxies can help to improve censorship resistance as shown in Figure 4.7: a web server can be configured for a particular URL to act as a reverse proxy and forward the traffic to a downstream GUNet peer running the `http(s)-server` plugin. This approach is even more effective if a well-known web server is used hosting a prominent service and only forwarding a specific resource to the peer. A reverse proxy can also increase network performance and prevent service degradation. Some ISPs detect network benchmarks to measure network throughput (“*speed tests*”) based on the URL requested and as a consequence increase bandwidth available for this test. This fact can be utilized

---

<sup>8</sup> <https://www.apache.org>

<sup>9</sup> <http://nginx.org>

by using a resource identifier like “/speedtest” to forward the traffic<sup>10</sup>.

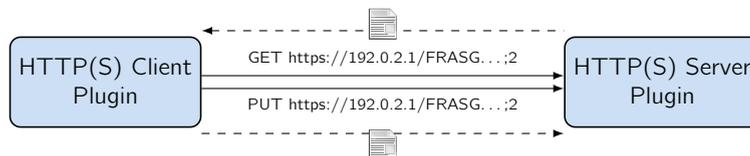


Fig. 4.6: GUNet's HTTP(S) Transport

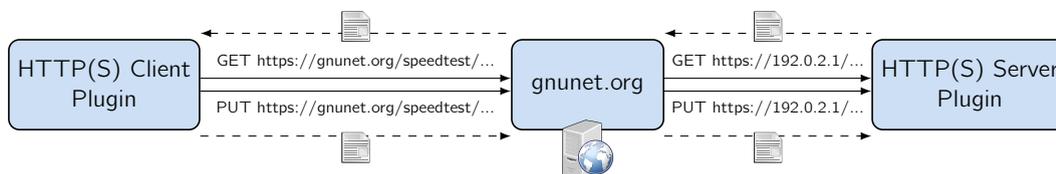


Fig. 4.7: GUNet's HTTP(S) Transport Using a Reverse Proxy

#### 4.3.16 The WLAN Transport

To enable peers to communicate in the situation of missing or failing communication infrastructure, one of the possibilities the GUNet transport infrastructure provides is to establish ad hoc communication networks based on WLAN communication. Ad hoc networks do not depend on a deployed, centralized communication infrastructure and participants communicate directly with each other.

WLAN networks are based on the IEEE 802.11 [IEE12] standard. This standard contains the media access control and physical layer specifications to establish wireless local area networks using the 2.4, 3.6, 5 and 60 GHz frequency bands. This technology is often referred to as *Wi-Fi* networks, but we refrain from using this term since *Wi-Fi* is a registered trademark of the Wi-Fi Alliance<sup>11</sup> and not related to the specifications. WLAN networks based on the IEEE 803.11 standards can operate in different frequency bands and the frequency band used is divided into smaller frequency blocks, so called *channels*. The 2.4 GHz band is divided 14 channels spaced 5 MHz apart. Each WLAN network operates on a single channel and is identified by a human readable identifier, the so called Service Set Identification (SSID). The IEEE 802.11 standard supports two operation modes for WLAN networks: *infrastructure* and *ad hoc* mode. In infrastructure mode, coordination between the participants of a WLAN network is managed by a centralized device, often referred to as *access point*. The set of nodes and the access point are called a Basic Service Set (BSS) in IEEE 802.11 terminology. This access point coordinates admission to the network, security parameters and manages medium access between participants. In ad hoc mode, no centralized infrastructure exists and the participants coordinate themselves. Nodes in the same WLAN ad hoc network have to use the same WLAN channel and SSID. This set of nodes create an Independent Basic Service Set (IBSS) in IEEE 802.11 terminology.

With the WLAN transport plugin, GUNet allows peers to communicate using WLAN ad hoc networking. The WLAN transport communicates directly on IEEE 802.11 physical layer with other peers on a pre-configured WLAN channel in 802.11 ad hoc mode. The WLAN plugin consists of a TRANSPORT service plugin and a small, privileged helper process

<sup>10</sup> <http://blog.fefe.de/?ts=b1032e75>

<sup>11</sup> <http://www.wi-fi.org>

used to configure and interact with the network interface. The WLAN transport plugin implements the TRANSPORT plugin API and is loaded by the TRANSPORT service. When the plugin is loaded, it starts the privileged helper process. Communication between the plugin and the helper is realized using the standard output and standard input of the helper process. When the helper process is started, it initiates communication with the WLAN interface using the radiotap<sup>12</sup> interface. The helper initializes the network card, checks that the networking interface is in 802.11 ad hoc mode and configures the network interface to forward all WLAN frames received to the helper process using the so called *promiscuous* or *monitoring* mode.

When TRANSPORT service wants to transmit data to a peer, it gives the data to the plugin. The plugin applies fragmentation to the messages if messages are too big to be transmitted in one WLAN frame and forwards the fragments to the helper process. The helper process directly injects IEEE 802.11 frames into the network card which are transmitted to other peers on the configured WLAN channel using a GNUnet-specific SSID ("132233445566"). To receive data, the WLAN helper process accesses the WLAN hardware using monitoring mode to receive WLAN frames without involving central access points. When the helper receives WLAN frames, it forwards these frames to the plugin, which reassembles the fragmented messages to forward them to the TRANSPORT service.

#### 4.3.17 The Bluetooth Transport

In addition to the WLAN plugin, we provide a Bluetooth based plugin for ad hoc networking. Bluetooth is a wireless technology intended to create *personal area networks*. Similar to IEEE 802.11, it operates in the 2.4 GHz short range radio ISM band. Bluetooth was designed to connect devices over a short distance. Bluetooth devices are commonly differentiated by their transmission power: a class 1 device has a maximum power consumption of 100mW and a maximum range of about 100m, a class 2 device has a maximum power consumption of 2.5mW and a maximum range of about 10m and a class 3 device has a maximum power consumption of 1mW and a maximum range of about 1m. Bluetooth was originally standardized as IEEE 802.15.1 but this standard is no longer maintained and Bluetooth specifications are instead provided by the Bluetooth Special Interest Group (SIG)<sup>13</sup>. Bluetooth was designed as a layered protocol stack with functionality implemented in so called *profiles*. The profiles available depend on the implementation of the respective protocol stack, but each protocol stack has to provide the mandatory profiles Link Management Protocol (LMP) to set-up and control radio links, Logical Link Control and Adaptation Protocol (L2CAP) to provide logical links between devices, and Service Discovery Protocol (SDP) to detect services provided by Bluetooth devices.

GNUnet provides a Bluetooth TRANSPORT plugin to allow peers to communicate with each other using the Radio Frequency Communications (RFCOMM) Bluetooth profile. RFCOMM provides communication between Bluetooth devices using a virtual serial data stream. With RFCOMM, applications can communicate over Bluetooth using a TCP-style reliable communication channel. Similar to the WLAN plugin, the Bluetooth plugin is split in a TRANSPORT plugin and a small helper process. The TRANSPORT plugin interacts with the TRANSPORT service and on the other hand employs the helper service to implement the Bluetooth-dependent functionality. The Bluetooth helper process implements the RFCOMM communication with other devices. During startup, the helper initializes the Bluetooth device configured in the configuration by retrieving the list of devices available

<sup>12</sup> <https://www.radiotap.org/>

<sup>13</sup> <http://www.bluetooth.org>

from the Bluetooth stack, configures the device to be discoverable, opens a new RFCOMM socket and registers new *GNUnet Bluetooth service* with the Bluetooth SDP service. The helper now scans for other Bluetooth devices with the GNUnet service enabled and when finding such a device connects to this device using RFCOMM. Initially the Bluetooth helper transmits the peers HELLO message to the remote peer. The peer receiving such a HELLO message forwards this message to the TRANSPORT service which will validate information contained and use this information to establish a connection to the peer. The Bluetooth plugin in addition supports to discover new neighbors by scanning for new Bluetooth devices supporting the GNUnet Bluetooth service.

#### 4.3.18 The Distance Vector Routing Transport

To improve connectivity between peers in the peer-to-peer overlay, GNUnet provides a DV component enabling peers to exchange data even if they are not directly connected in the transport underlay [Eva11]. With DV, traffic between peers not directly connected in the transport underlay is forwarded by intermediate peers via the peer-to-peer overlay. DV enabled peers connected in the peer-to-peer overlay exchange information about the peers they are connected to. A DV enabled peer can therefore create a routing table containing information which peer it can communicate with by sending data over a next-hop peer.

GNUnet's DV component consists of two parts: the DV service and a DV transport plugin. The DV service is responsible to communicate with other peers in the peer-to-peer overlay, exchange routing information with these peers, maintain the routing table, forward data send with DV to the next hop peer and receive data forwarded from other peers with DV via the overlay. The DV plugin is a transport plugin and is responsible to notify the TRANSPORT service about peers reachable via DV and the overlay. The plugin accepts data from TRANSPORT to be send with DV and passes this data to the DV service to have the data forwarded to the destination via the overlay. When the DV service receives data via the overlay, it passes the data to the plugin, which gives the data to TRANSPORT as if the data were received directly from the source peer. This approach is depicted in Figure 4.8.

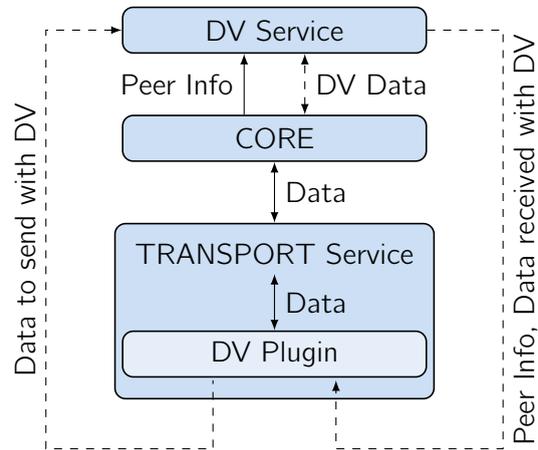
The DV service communicates with other peers using the peer-to-peer overlay and the CORE service described in Section 4.3.19. When DV-enabled peers connect on CORE-level, the DV services on both peers exchange information about the peers they are connected with. Here they use GNUnet's SET service to exchange the set of peers efficiently. This information is used to create a routing table containing which peer can be reached over which next-hop peer. Information about peers reachable via DV is given to the DV plugin which notifies TRANSPORT about available sessions to communicate with this peer. TRANSPORT is agnostic about the DV functionality and only knows that it can send data to a certain peer using the DV plugin and a certain session.

When TRANSPORT has to send data to a DV peer, it passes the data to the DV plugin using the given session. The DV plugin forwards this information to the DV service. The DV service performs a lookup for the next-hop peer in the routing table and sends the data to the next hop router via the peer-to-peer overlay and using the CORE service.

When the DV service receives data from a DV peer with the local peer as destination, it passes the data to the DV transport plugin which passes the data to TRANSPORT which forwards the data to the TRANSPORT clients to process the data. If the local peer is not the destination, DV performs a lookup in the routing table and forwards the data to the next-hop peer via the peer-to-peer overlay and CORE.

The DV component is an effective way to improve connectivity between peers in the overlay. This allows to bridge between separated parts of the network if only a single peer

can talk to both partitions and DV helps to alleviate the effects of restricted end-to-end communication.



**Fig. 4.8:** GNUet's Distance Vector Routing Transport Plugin

#### 4.3.19 Secure Communication Between Peers with CORE

The TRANSPORT service's main focus is to provide communication with remote peers, to increase connectivity between peers and to be resilient against service degradation attempts. But TRANSPORT does not provide confidential or integrity protected communication for applications. In GNUet's transport infrastructure, the CORE service is responsible to provide link-encrypted communication between peers directly connected via the transport underlay. GNUet's CORE service builds on the TRANSPORT service and uses TRANSPORT to provide confidential, authenticated, integrity protected communication between connected peers. The CORE service employs a key exchange to exchange a shared secret between peers used to encrypt traffic exchanged between peers. Peer-to-peer applications should therefore use CORE service to communicate with remote peers and should not use TRANSPORT service directly.

By using ECDHE and Curve25519 [Ber06] to exchange a shared secret to encrypt traffic with both the AES-256 [DR02] and Twofish [SKW<sup>+</sup>98] symmetric block ciphers, CORE achieves confidentiality with perfect forward secrecy. The ephemeral key used with ECDHE is generated during startup and never stored, written to disk and is refreshed in periodic intervals. Communication is authenticated using the Ed25519 signature scheme [BDL<sup>+</sup>11], the EdDSA digital signature scheme using ECC and the twisted Edwards curve Curve25519. Integrity of communication is protected using authenticated encryption with Secure Hash Algorithm (SHA)-512. CORE communication is protected against replay attacks using ephemeral keys, nonces, timestamps, challenge-response and message counters.

CORE can only communicate with directly connected peers and does not perform any routing. In addition, CORE does not provide flow control and expects applications to process data at line speed. Communication with CORE is unreliable and does not ensure in-order delivery of messages. Applications requiring these features should use the CADET system [PG14].

CORE is implemented as a separate GNUet service and is a TRANSPORT client. When CORE is started, it connects to TRANSPORT and gets notified by TRANSPORT about connecting and disconnecting peers and it can use TRANSPORT to exchange data with con-

nected peers. During startup, CORE creates a new cryptographic ephemeral key pair. This ephemeral key pair is used with the ECDHE key exchange and is renewed in regular intervals ( $\sim 12$  hours). CORE's key exchange provides *perfect forward secrecy* since ephemeral keys are never written to disk or stored anywhere else. The ephemeral key is signed with the peer's long-term EdDSA private key described with peer identities in described in Section 4.3.1. These signed ephemeral keys are exchanged between peers using *ephemeral key messages*, containing the ephemeral public key, the signature, a creation and expiration time and additional information. When CORE is notified about a peer connecting, the peers exchange their ephemeral key messages. If a peer receiving an ephemeral key did not confirm the authenticity of the sender before, it sends an encrypted PING message to the sender containing a challenge the receiver has to decrypt and return to the sender in an encrypted PONG message. If the sender can verify the correctness of the PONG message, it treats the remote peer as authenticated. The Hash-based Message Authentication Code (HMAC) used to protect data integrity, is calculated over the encrypted payload of the message to send. Now both peers are provided with the required cryptographic information to exchange encrypted messages. When encrypted messages are exchanged with CORE, CORE messages include the HMAC of the encrypted payload in the header and an initialization seed, a sequence number, a timestamp and the encrypted payload. To determine which application can process which type of message, CORE uses *typemaps*: applications using CORE register the message types they are interested in when connecting to CORE. This allows CORE to pass only messages to applications, that these applications are interested in and can process.

## 4.4 Evaluation

### 4.4.1 Methodology and Setup

To evaluate the performance of the TRANSPORT service and the different TRANSPORT plugins, we use a profiler tool benchmarking the performance of the different TRANSPORT plugins. For a first evaluation, we measure the performance with peers communicating on a local system to analyze the performance limitations imposed by local systems without the limitations caused by networking equipment and network links. We use two computer systems to compare TRANSPORT performance on different platforms: a 4 year old high-end desktop workstation and a recent notebook equipped with up-to-date computing hardware and optimized for low power consumption. In a second evaluation, we connect peers over a real network and measure the performance of the different plugins over a network connection to analyze the limitations imposed by networking equipment and network links.

### 4.4.2 Experimental Setup

To benchmark performance on a local system, we use for the experimental setup a Dell Precision T3500 workstation<sup>14</sup> manufactured in 2009 system running Ubuntu 14.04 64-bit equipped with an Intel Xeon W3520<sup>15</sup> Central Processing Unit (CPU) with 4 cores running at 2.67 GHz supporting hyperthreading. The system is equipped with a Broadcom NetXtreme BCM5761 gigabit Ethernet controller and 24 GB DDR3-1333 MHz memory. Due to GUNet's design using a separate process for every GUNet service, we can

<sup>14</sup> <http://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dell-Precision-T3500-Spec-Sheet.pdf>

<sup>15</sup> [http://ark.intel.com/de/products/39718/Intel-Xeon-Processor-W3520-8M-Cache-2\\_66-GHz-4\\_80-GTs-Intel-QPI](http://ark.intel.com/de/products/39718/Intel-Xeon-Processor-W3520-8M-Cache-2_66-GHz-4_80-GTs-Intel-QPI)

ensure that the TRANSPORT services of both peers can run on a dedicated core and the services of both peers can run at full speed without interfering with each other. As a drawback, the Xeon W3520 does not yet support Intel's AES-NI technology<sup>16</sup> adding new CPU instructions for AES hardware acceleration. These instructions can impact the performance for TRANSPORT plugins using an additional layer of encryption as for example the HTTPS plugin. We therefore conducted additional experiments to analyze the impact of AES hardware acceleration and to analyze the impact of the CPU clock rate on TRANSPORT performance.

For the additional experiments, we use a recent Lenovo T440s notebook<sup>17</sup> manufactured end 2013 running Ubuntu 14.04 64-bit. This notebook is equipped with an Intel Haswell Core i5-4210U 1.6 GHz dual core CPU<sup>18</sup> supporting a maximal turbo frequency of 2.7 GHz with Intel's Turbo Boost Technology, hyperthreading and the AES-NI technology. The T440s is equipped with an Intel I218-V gigabit Ethernet controller and 12 GB of DDR3L-1600 memory.

According to the specifications, the CPUs in both systems can provide a maximum memory bandwidth of 25,6 GB/s. But the CPU in the Dell Xeon system supports DDR3-800/1066 memory and provides 3 memory channels whereas the T440s is equipped with faster DDR3L-1333/1600 memory and only provides 2 memory channels.

To analyze the impact of AES hardware acceleration, we use the gnutls<sup>19</sup> command line tool to benchmark crypto performance on both the Dell and the Lenovo system. As we can see from Table 4.1 provides a CPU with hardware acceleration a much higher performance for cryptographic operations. To benchmark the performance of the HTTPS plugin on such a machine, we conducted additional HTTPS experiments with the Lenovo T440s notebook. To counteract impacts caused by the lower number of CPU cores, we increased the number of iterations in the experiment from 5 to 10 to reduce the impact of measurement outliers caused by scheduling conflicts.

To compare the impact of CPU speed on TRANSPORT performance, we in addition benchmarked the performance of the HTTP plugin on the Lenovo T440s to be able to compare it with the performance of the workstation.

To benchmark performance in a real networking environment, we use two notebooks connected over gigabit Ethernet. The first notebook (sending data to the other peer) is the Lenovo T440s described before. The second notebook responsible to receive data is a Lenovo T61 with an Intel T8300<sup>20</sup> dual core CPU running with 2.4 GHz without hyperthreading support and AES hardware acceleration. The system is equipped with an Intel 82556M gigabit Ethernet controller and 4 GB of RAM and running Ubuntu 14.04 64-bit.

All systems used for the experiments run GUNet in the development version from September 1st 2014 (Revision 34252) compiled with gcc 4.8.2 and optimization level -O2.

---

<sup>16</sup> <https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set>

<sup>17</sup> <http://www.lenovo.com/shop/emea/content/pdf/ThinkPad/TSeries/T440sDSEN.pdf>

<sup>18</sup> [http://ark.intel.com/products/81016/Intel-Core-i5-4210U-Processor-3M-Cache-up-to-2\\_70-GHz?q=4210U](http://ark.intel.com/products/81016/Intel-Core-i5-4210U-Processor-3M-Cache-up-to-2_70-GHz?q=4210U)

<sup>19</sup> <http://www.gnutls.org>

<sup>20</sup> [http://ark.intel.com/products/33099/Intel-Core2-Duo-Processor-T8300-3M-Cache-2\\_40-GHz-800-MHz-FSB?q=t8300](http://ark.intel.com/products/33099/Intel-Core2-Duo-Processor-T8300-3M-Cache-2_40-GHz-800-MHz-FSB?q=t8300)

**Tab. 4.1:** Performance of Cryptographic Cipher Suites on an Intel Xeon W3520 and i5-4200U

Cipher	Intel Xeon W3520 2.67 GHz /wo AES-NI	Intel i5-4200U 1.6 GHz /w AES-NI
AES-128-CBC with SHA1	101.74 MB/s	296.96 MB/s
AES-128-CBC with SHA256	76.21 MB/s	184.27 MB/s
AES-128-GCM	46.42 MB/s	2191.36 MB/s
SHA1	368.64 MB/s	604.16 MB/s
SHA256	158.54 MB/s	276.48 MB/s
SHA512	256 MB/s	235.52 MB/s
3DES-CBC	13.96 MB/s	18.53 MB/s
AES-128-CBC	141.90 MB/s	563.2 MB/s
ARCFOUR-128	149.89 MB/s	286.72 MB/s

#### 4.4.3 Methodology

In our setup, we have two peers connected with each other using a particular TRANSPORT plugin. The profiler tool ensures that the peers are connected with each other and then starts to benchmark TRANSPORT and plugin performance sending data from one peer to the other peer. The profiler measures the time required to send a fix amount of messages with a particular message size from one peer to the other peer. The profiler tool can be configured with the message size, the amount of messages to send, and the number of iterations to perform. The profiler keeps statistics about the iterations and uses this data to calculate the data rate for each iteration, the average data rate over all iterations and the standard deviation for both duration and data rate. Based on these values we in addition calculate the average message rate of the sending peer.

In our experiment, we start two GUNet peers with TRANSPORT configured to only load and use the TRANSPORT plugin to benchmark, connect the peers with each other and use the profiler tool to send messages from one peer to the other peer. The profiler tool acts as a TRANSPORT service client and uses the TRANSPORT service to exchange with the other peer. With this approach we cannot only evaluate the performance of the plugins but the overall transport infrastructure. The profiler measures TRANSPORT performance sending messages with different message sizes from one peer to another peer: it starts with a minimum message size of 1,000 bytes per message and increases the message size in steps of 1,000 bytes up to 65,000 byte per message which reflects the maximum message size of a GUNet message of 64 KiB. For every message size it sends 10,240 messages from one peer to the other peer and repeats this in four iterations. The message size represents the amount of application data sent by the profiler. The amount of data sent *on the wire* can be higher due to messages sent by TRANSPORT service and plugin specific overhead like message wrappers, control messages or protocol specific overhead.

The UDP plugin uses a maximum send rate to limit the maximum data rate to prevent overloading the network and the other peer. By default this value is configured with 1 MiB/s. We modified this value for the performance benchmarks to 1 GiB/s to be able to evaluate the maximum performance with this plugin. To analyze the impact of fragmentation on UDP, we conducted a separate experiment where we transmitted 20,240 messages using UDP starting with a message size of 10 bytes and increased the message size by 10 bytes in each run up to a message size of 2,000 bytes. By default the MTU

for the UDP plugin before messages get fragmented is configured with 1,400 bytes for the overall UDP message (including the size of the wrapper used by the UDP plugin). Therefore, packets with a payload slightly smaller than 1,400 byte will get fragmented.

For the performance benchmark of the WLAN plugin, we had to reduce the number and the size of messages exchanged between peers. The results obtained with the WLAN plugin are created by sending 10 messages with the given message size in 4 iterations to the other peer. In this experiment we used WLAN channel 6 (2.473 GHz) in the 2.4 GHz Industrial, Scientific and Medical (ISM) band. The results for the WLAN plugin are depicted in Figure 4.23.

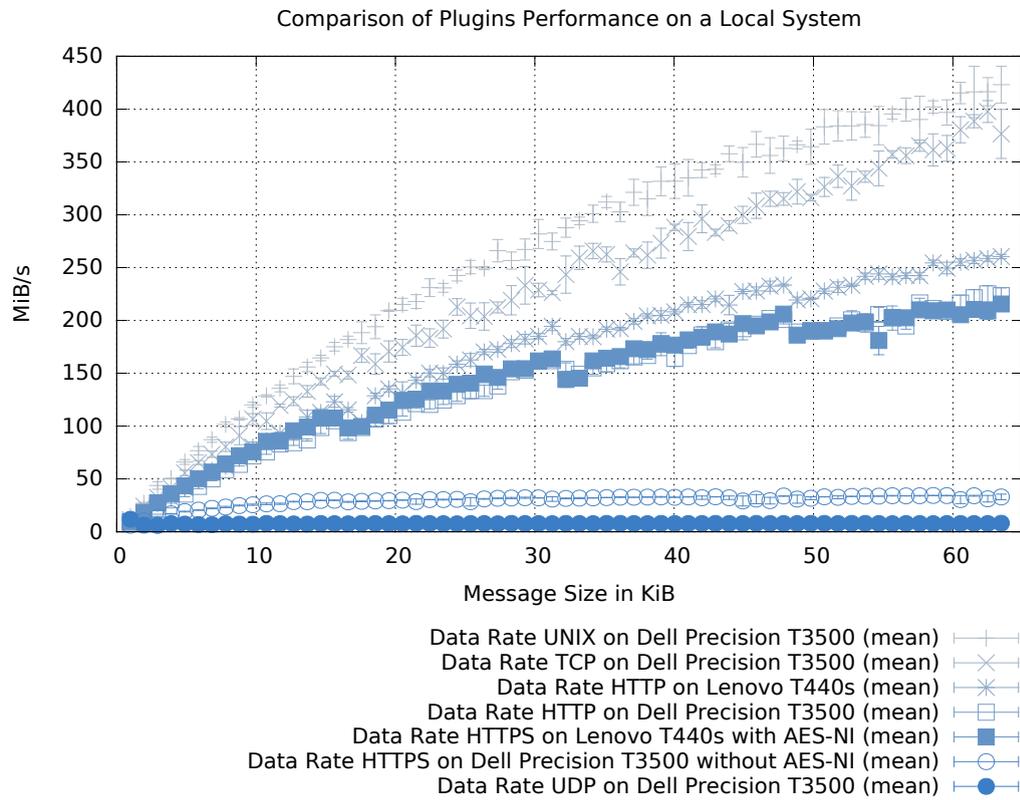
The Bluetooth transport plugin was developed by a student in a Google Summer of Code project. While the TRANSPORT plugin itself and basic functionality of the privileged Bluetooth helper process work it was not possible to perform benchmarks within the evaluation done for this thesis. While peers successfully connect using the Bluetooth service discover as described in Section 4.3.17, the helper stops exchanging messages when a large number of messages is sent. This issue requires further investigations not possible within this evaluation.

#### 4.4.4 Results on Local Performance

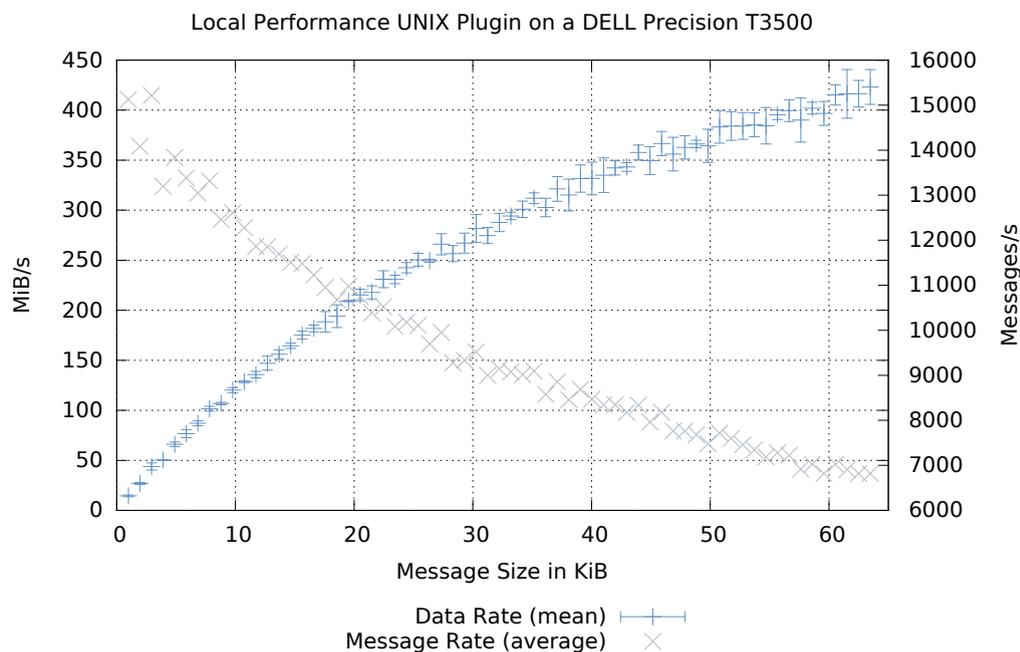
To evaluate the performance on a local system, we benchmarked TRANSPORT service performance in combination with the UNIX domain socket, TCP, UDP, HTTP and HTTPS plugin. A performance comparison for the different plugins is depicted in Figure 4.9. The data and message rates for the different TRANSPORT plugins are depicted in:

- Figure 4.10 for the UNIX domain socket plugin
- Figure 4.11 for the TCP plugin
- Figure 4.12 for the UDP plugin
- Figure 4.13 showing the impact of fragmentation with UDP
- Figure 4.14 for the HTTP plugin on the DELL system
- Figure 4.14 for the HTTP plugin on the Lenovo system
- Figure 4.16 for HTTPS on the DELL system without AES hardware acceleration
- Figure 4.17 for HTTPS on the Lenovo system with AES hardware acceleration

With the performance analysis on the local system we can see that the data rate achieved largely depends on the message size of the messages transferred between the systems. Here the possible data rate achieved between peers is limited by the CPU of the system. In our experiments we observed that 2 out of the 4 CPU cores were running at 100% CPU load with the GNUnet TRANSPORT service. This is caused by the fact the messages are not aggregated before being sent over by instead every message is handled independent causing computational overhead. For the different transport plugins we can see that we can achieve very high data rates for both the TCP and the UNIX domain socket plugin with only slight overall performance difference. Performance for the HTTP plugin is lower than the performance of the TCP and UNIX plugin. For the HTTPS plugin performance is limited by the Xeon CPU having to encrypt and decrypt network traffic. Here both CPU cores running TRANSPORT where constantly under 100% load. On

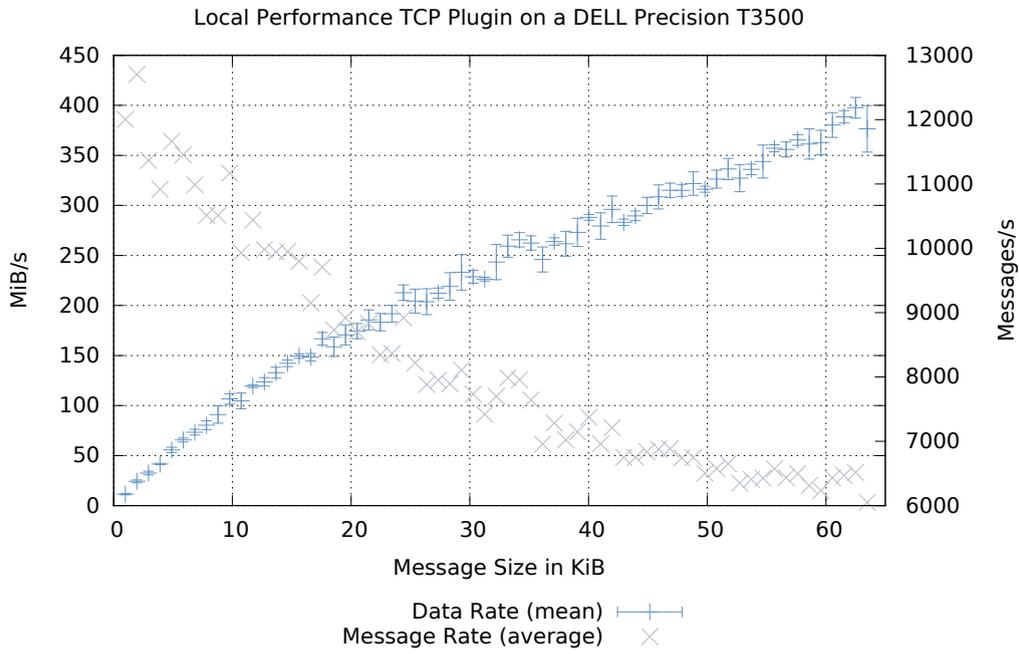


**Fig. 4.9:** Comparison of TRANSPORT Performance with Different Plugins on a Local System

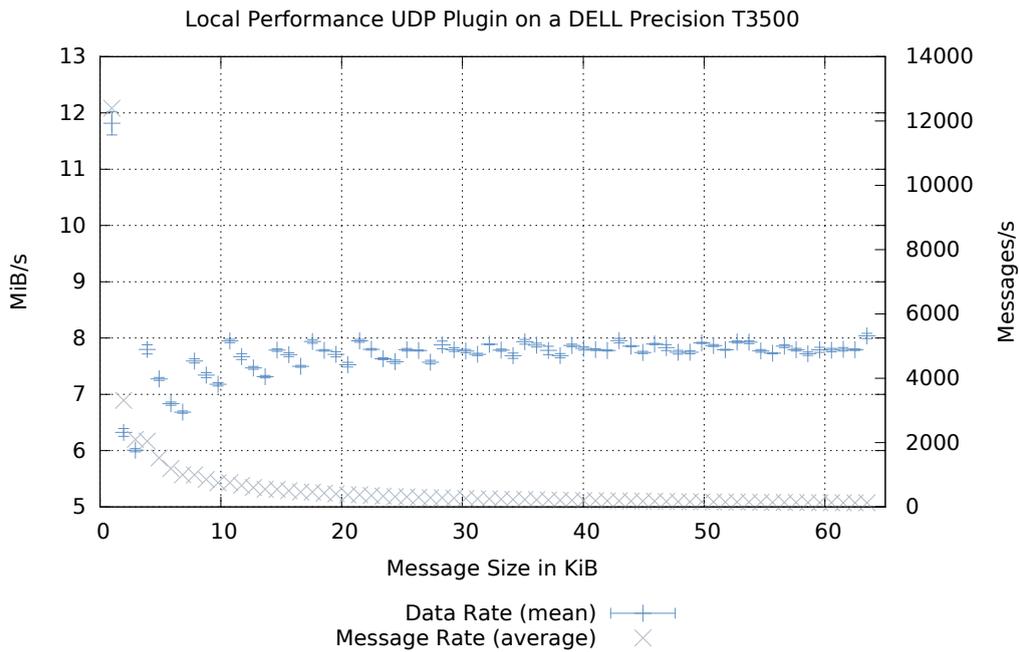


**Fig. 4.10:** Local Performance UNIX Plugin on a DELL Precision T3500

the Lenovo T440s system with AES hardware acceleration, we can achieve significantly higher performance with the HTTPS plugin. With AES hardware acceleration HTTPS



**Fig. 4.11:** Local Performance TCP Plugin on a DELL Precision T3500



**Fig. 4.12:** Local Performance UDP Plugin on a DELL Precision T3500

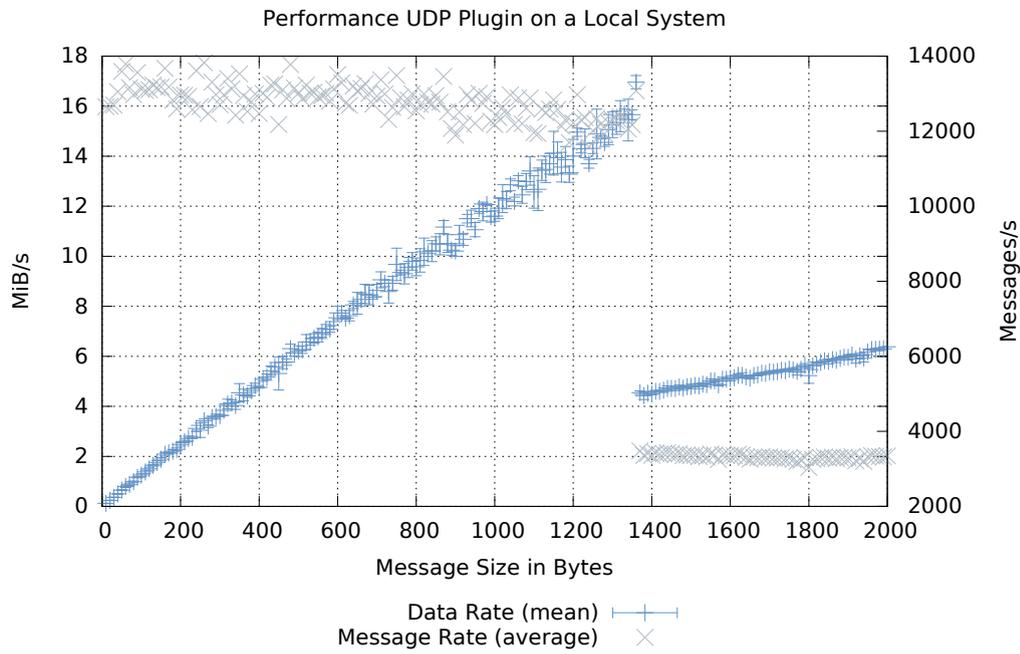


Fig. 4.13: Local Performance UDP Plugin on a DELL Precision T3500: Impact of Fragmentation

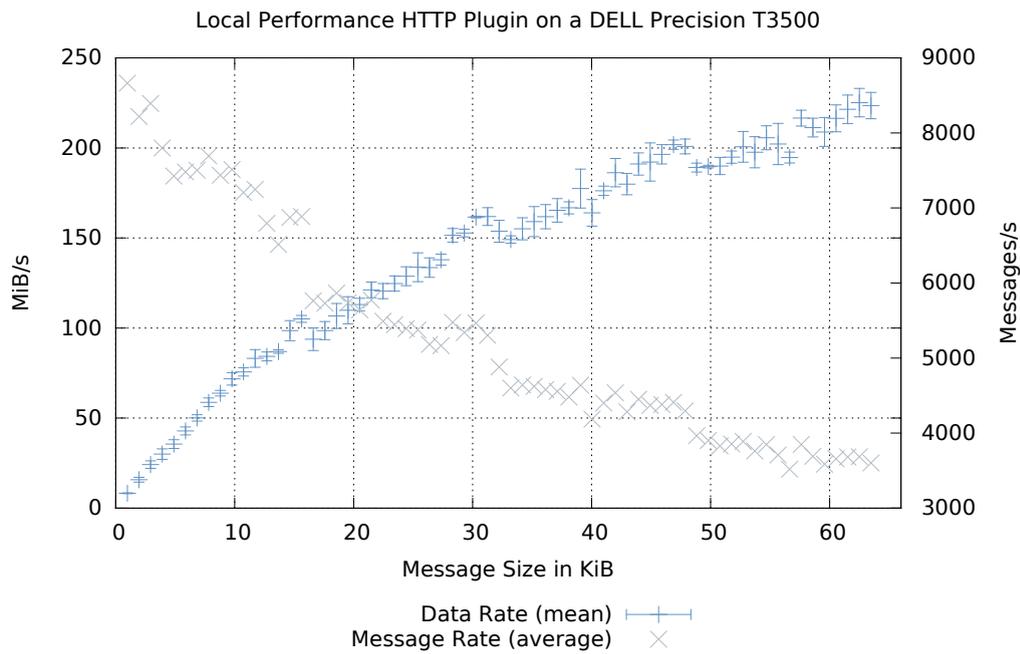


Fig. 4.14: Local Performance HTTP Plugin on a DELL Precision T3500

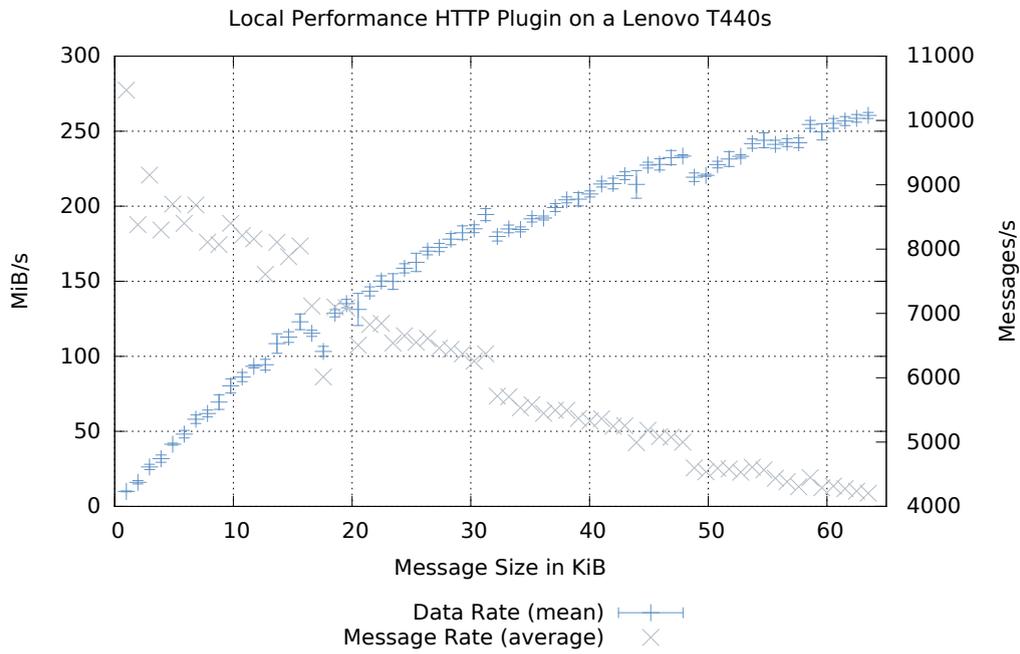


Fig. 4.15: Local Performance HTTP Plugin on a Lenovo T440s

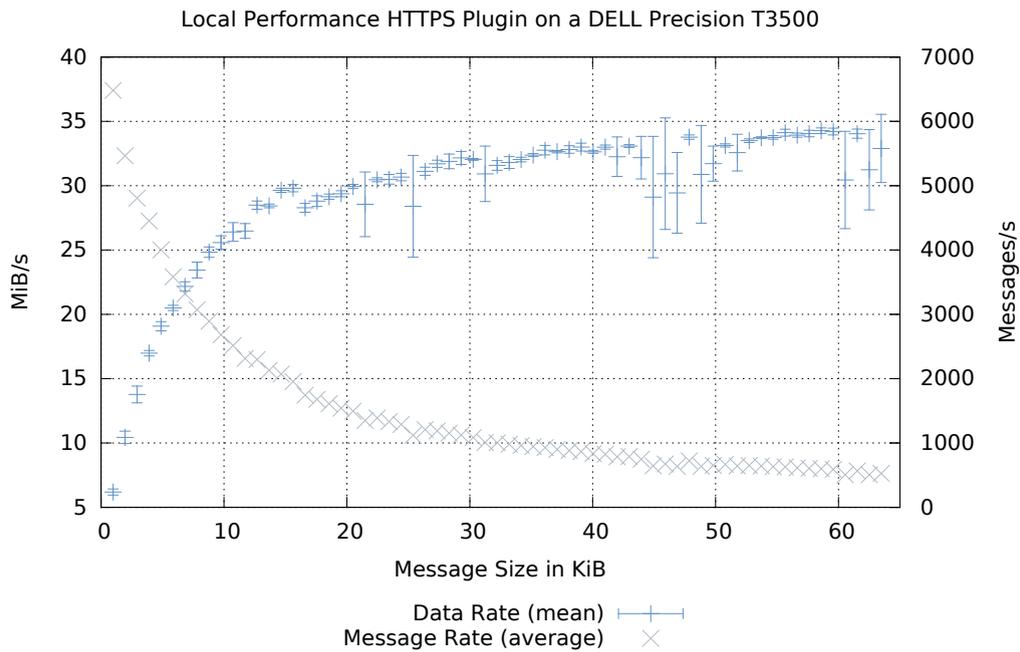
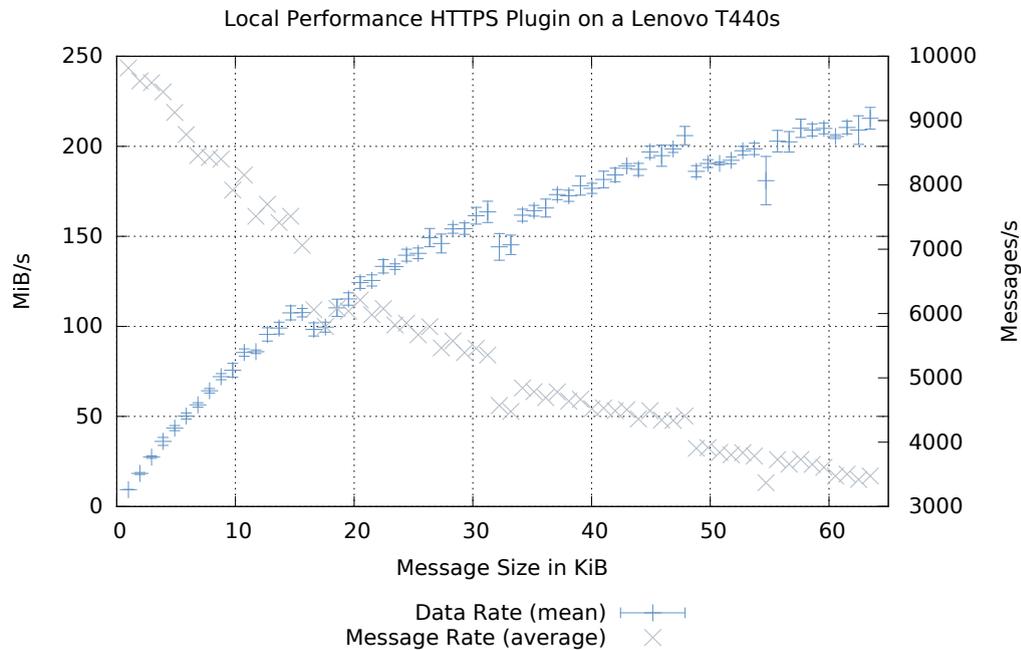


Fig. 4.16: Local Performance HTTPS Plugin on a DELL Precision T3500 not supporting AES-NI



**Fig. 4.17:** Local Performance HTTPS Plugin on Lenovo T440s supporting AES-NI

performance is similar to the performance achieved with the HTTP plugin. On this system CPU cores did not immediately go to 100% load. We did not tune the cryptographic cipher suites and used the default cipher (AES-128-GCM) as this is the default used and selected by the client and server libraries. Here it is important to note that neither the Xeon CPU nor the T8300 CPU used support Intel's AES-NI technology adding new CPU instructions for full AES hardware acceleration. Therefore, newer CPUs supporting this technology can achieve higher crypto performance even when running at a lower clock rate. As we can see with from Table 4.1 is cryptographic performance on the Xeon for the AES-128-GCM cipher with 46.42 MB/s close to the data rate achieved with the HTTPS transport plugin whereas on the Lenovo system with AES hardware acceleration the crypto performance does not limit the performance of the HTTPS plugin.

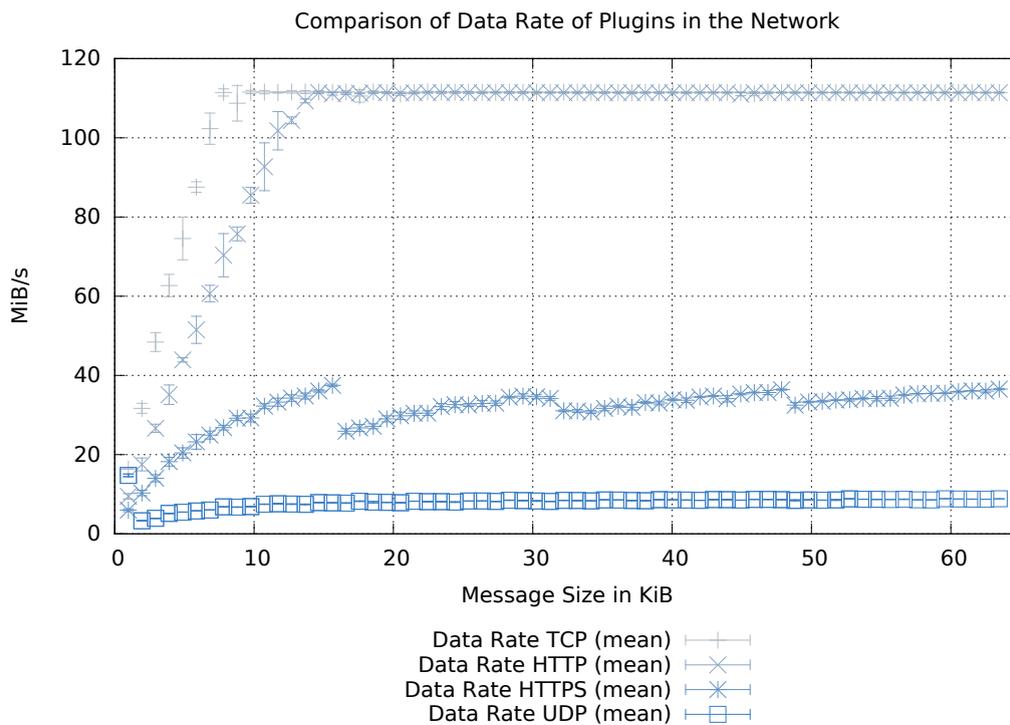
When comparing the performance of the HTTP plugin achieved on the DELL and the Lenovo system, we can see that even modern CPUs optimized for power consumption can achieve higher performance than older systems running with a higher clock rate due to progress in CPU design. For HTTP performance, the Lenovo system could even outplay the DELL workstation. While running the experiments on the Lenovo T440s system, we monitored the clock rate on the system and observed that the system never used the turbo boost frequency but operated with a maximum clock rate of 1.6 GHz.

The performance of the UDP plugin is significantly lower than for the other plugins. This is caused by the expensive fragmentation and fragment acknowledgment mechanisms and the current limitation that we allow only one UDP message in flight at the same time. As soon as messages have to be fragmented, performance of the UDP plugin significantly decreases.

#### 4.4.5 Results on Network Performance

A comparison for the performance of different plugins achieved over the network is depicted in Figure 4.18. The data and message rates for the different TRANSPORT plugins achieved over gigabit Ethernet are depicted in:

- Figure 4.19 for the TCP plugin
- Figure 4.20 for the UDP plugin
- Figure 4.21 for the HTTP plugin
- Figure 4.22 for the HTTPS plugin



**Fig. 4.18:** Performance Comparison Between Plugins over Gigabit Ethernet

With our experiment using a real network link between peers we see that both the TCP and the HTTP can saturate a gigabit Ethernet link. Here both plugins perform similarly good achieving a data rate of about 112 MiB/s or 913 MBit/s only for application data not including TRANSPORT and plugin specific overhead. For the HTTPS and the UDP plugin we achieve the same performance as on the local system. For HTTPS performance it is important to note that the T61 notebook did not support AES hardware acceleration. Here the same limitations to performance as on the local system apply. Networking links and hardware do not impose limitations for the HTTPS plugin as the performance is CPU bound. For the WLAN ad hoc plugin we can only present limited experimental data. We could transmit data with a data rate of about 1.5 KiB/s between peers. Here it is important to note the experiments were conducted in an environment where many WLAN networks are operated and many users communicate over WLAN. Here a more sophisticated evaluation is required in an environment with less WLAN equipment and users. In addition, the use of the 5 GHz ISM band has to be considered in future work.

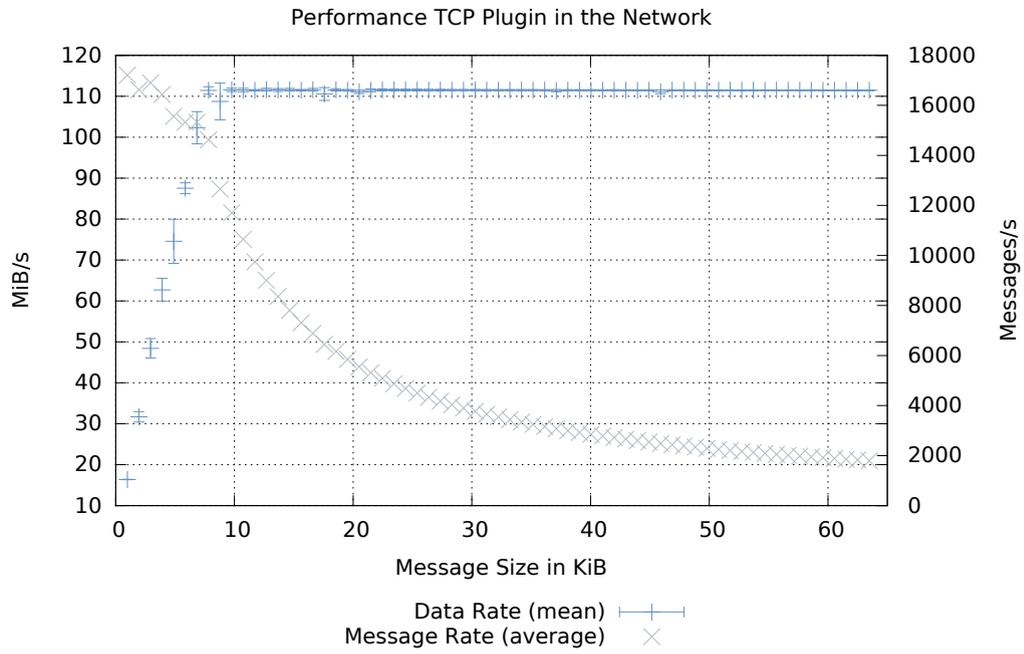


Fig. 4.19: Performance TCP Plugin over Gigabit Ethernet

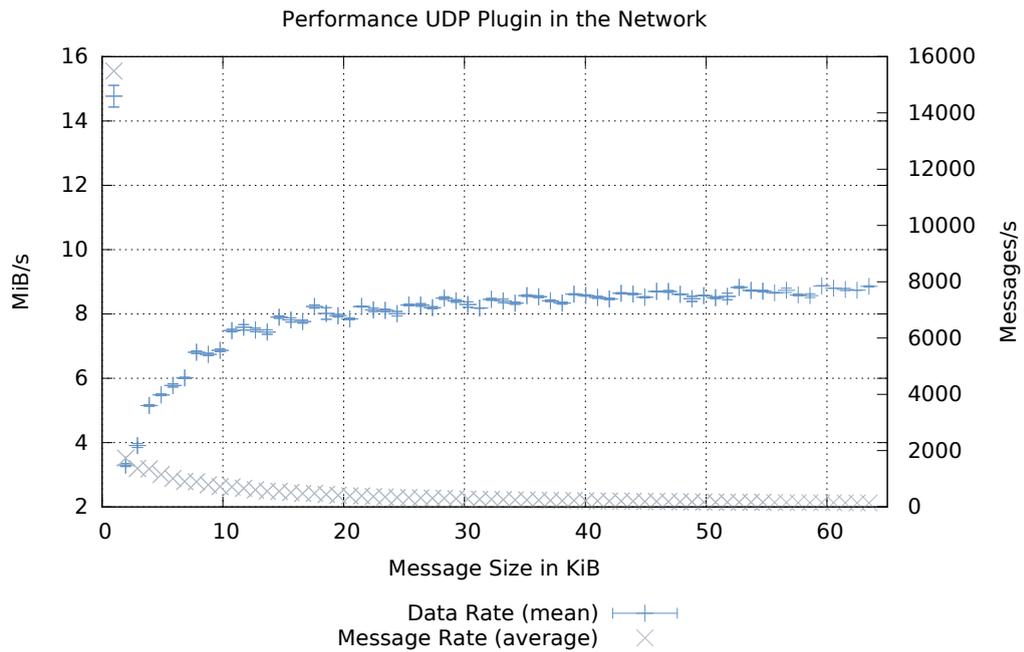


Fig. 4.20: Performance UDP Plugin over Gigabit Ethernet

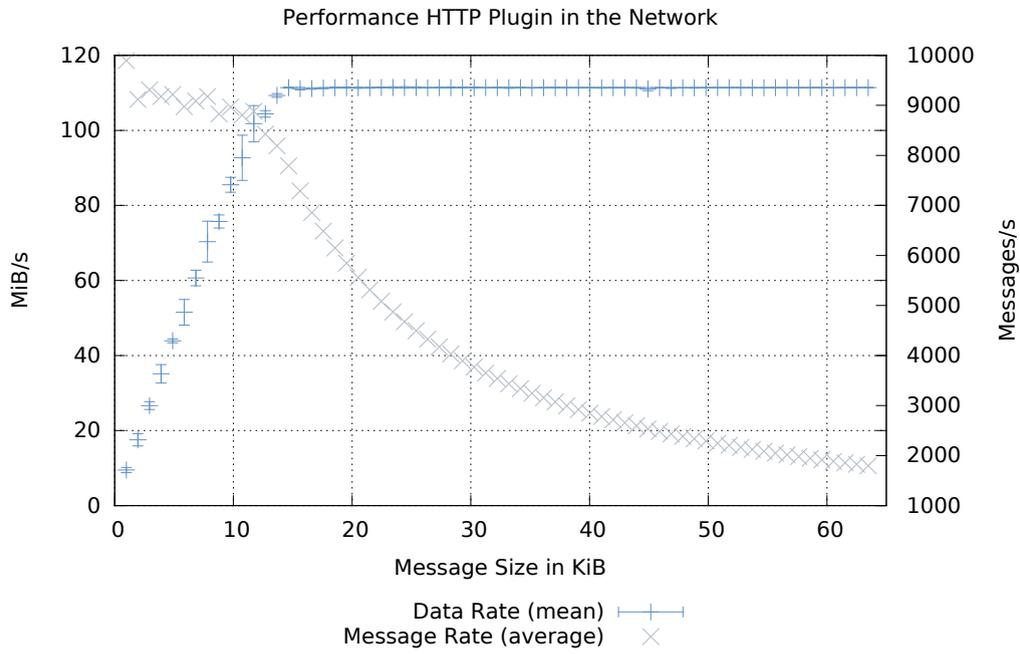


Fig. 4.21: Performance HTTP Plugin over Gigabit Ethernet

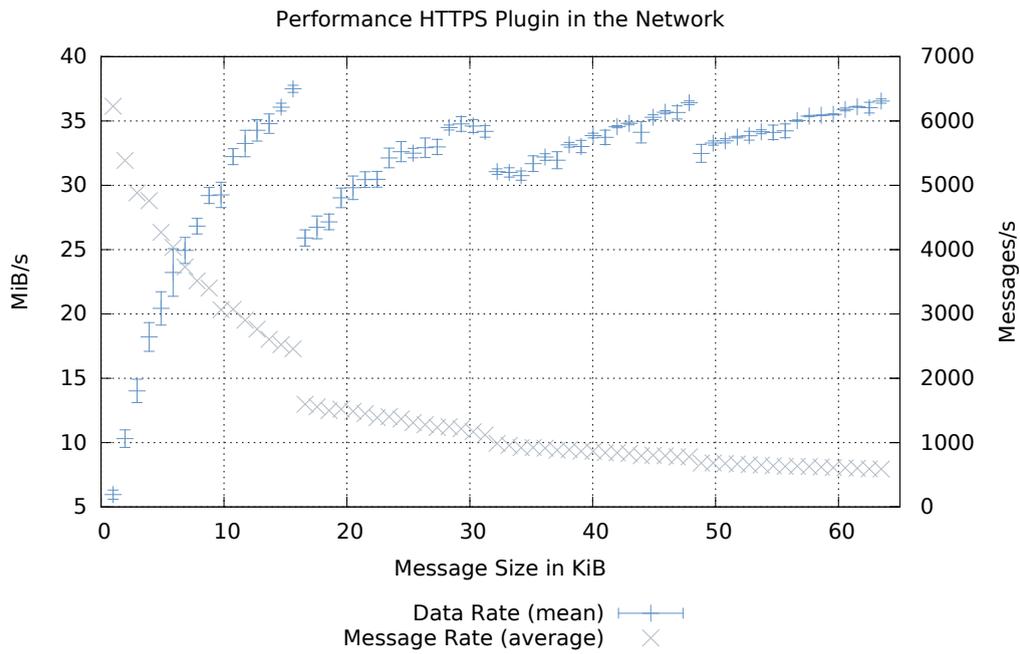


Fig. 4.22: Performance HTTPS Plugin over Gigabit Ethernet

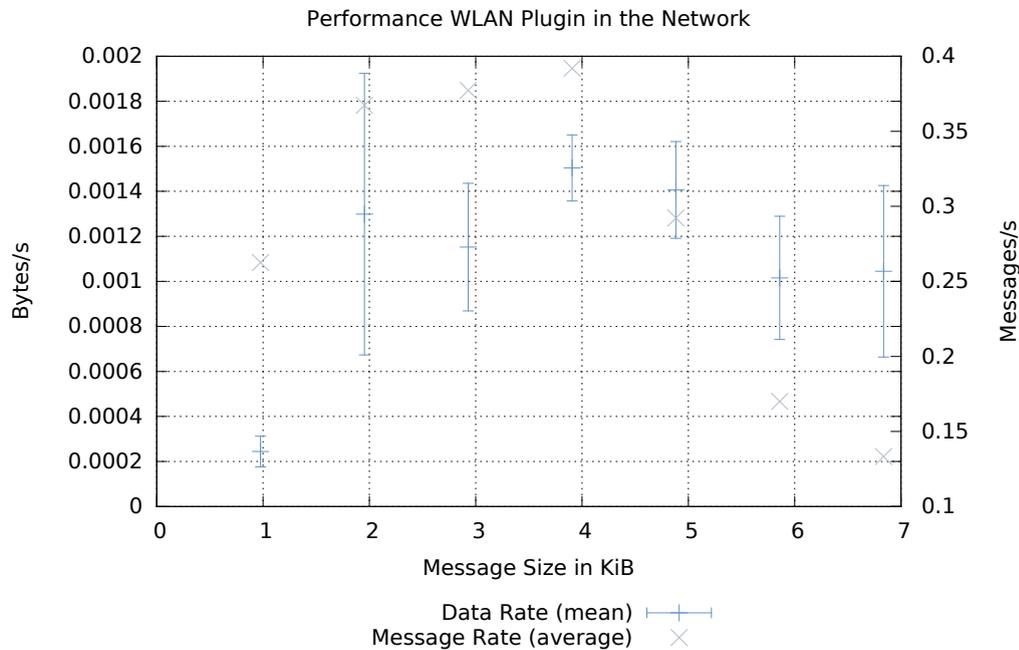


Fig. 4.23: Performance WLAN Plugin

## 4.5 Related Work and Comparison

In this section we analyze the transport infrastructures of other distributed applications and compare their approaches with the design and implementation of GUNet's transport infrastructure. In this context we put a special focus on peer-to-peer applications with a focus on security and resilient communication.

### 4.5.1 Tor's Pluggable Transport Architecture

The Tor project<sup>21</sup> is a project with the focus on providing anonymous communication based on the idea of onion routing. By providing anonymous communication, the Tor software and its communication are in the focus of various parties interested in preventing free and anonymous communication on the Internet and therefore trying to suppress and filter communication with Tor. Various attempts are made to limit and degrade Tor communication. The Tor project therefore provides various counter measures to antagonize these attempts [LMP<sup>+</sup>12, The, SJP<sup>+</sup>11]. To circumvent censorship on IP level, Tor uses so called *bridge* relay nodes. Bridge relays are Tor nodes not listed in the Tor directory servers. One proposed approach to circumvent censorship on DPI level is the use of a pluggable transport architecture, comparable to GUNet's transport plugin architecture, described in [The]. Using this pluggable transport architecture, traffic between a Tor client and a bridge can be transformed in a way that it is less recognizable as Tor traffic. Various different transports are listed, all working very differently. Prominent examples for pluggable transports are Scramblesuit, Stegotaurus and FTE [DCRS13] for obfuscation, SkypeMorph [MMLDG12] for traffic morphing, and Flashproxy [FHE<sup>+</sup>12] and Dust for resilient communication. An extensive list of pluggable transports available can be found

<sup>21</sup> <https://www.torproject.org/>

on the Tor website<sup>22</sup>.

Tor only supports one pluggable transport to be used at a time and does not support to switch between pluggable transport or combine multiple pluggable transports. Users must be aware that their ISP blocks or censors their Internet connection and configure the pluggable transport they want to use. Here users have to understand how the ISP censor degrades Tor traffic and know which pluggable transport is the best to antagonize this kind of degradation attempt. In addition require some pluggable transports additional configuration effort or the use of special Tor bridges compliant with these pluggable transports [The14]. If multiple pluggable transports are available, the Tor software also simply uses the first pluggable transport configured.

#### 4.5.2 SPOVnet's ARIBA Resilient Transport Underlay

The Spontaneous Virtual Network (SpoVNet) project<sup>23</sup>[BHMW08b] is a research project with the goal to allow decentralized applications to communicate over a heterogeneous communication infrastructure using self-organizing virtual overlay networks of decentralized systems. SpoVNet is a framework with the aim to enable developers to implement a diverse set of applications on top of a common foundation. SpovNet tries to overcome heterogeneity in networks and network protocols as well as limitations (like NAT and firewalls) with a self-organizing, decentralized transport infrastructure, virtual addressing of nodes and a DHT-based control overlay.

SpovNet uses the Ariba underlay abstraction<sup>24</sup>, described in [BHMW08a], as a transport infrastructure. Ariba provides communication over heterogeneous communication links using an homogeneous interface for the different supported communication mechanisms and copes with mobility, heterogeneity, and middleboxes restricting end-to-end connectivity. To provide this homogeneous interface, Ariba provides an abstraction layer with virtual links for the various communication mechanisms. As underlay communication protocols Ariba supports multiple transport protocols including TCP, UDP, SCTP, and IPv4, IPv6 and Bluetooth RFCOMM as communication mechanisms. Ariba supports NAT relaying to connect hosts in restricted environments and protocol relays to overcome heterogeneity in networks.

Ariba has a similar objective as the GNUnet's TRANSPORT infrastructure but focuses on overcoming heterogeneity between networks and does not focus on making communication resilient or secure. Ariba's main focus is to provide different communication mechanisms with the focus to provide connectivity between participants. So Ariba uses the same approaches and both designs share the objective to increase connectivity, but Ariba only focuses on overcoming heterogeneity of networks not on using these approaches to improve communication with respect to resilience and censorship-resistance.

#### 4.5.3 I2P's Transport Architecture

The Invisible Internet Project (I2P)<sup>25</sup>, described in [ZH11], is a network to provide anonymous communication over the Internet. In contrast to Tor, I2P does not primarily try to provide anonymous access to services on the Web, but instead creates an overlay network where participants can anonymously exchange messages using a variety of different services (like email, blogs or file sharing) that are operated within the I2P overlay.

<sup>22</sup> <https://www.torproject.org/docs/pluggable-transport.html.en>

<sup>23</sup> <https://www.spovnet.net>

<sup>24</sup> <http://www.ariba-underlay.org>

<sup>25</sup> <https://geti2p.net>

I2P's transport service provides message-based communication between I2P routers using multiple transports. At the moment two transports are supported: a TCP-oriented transport called NTCIP, and secure semi-reliable UDP (SSU). I2P also supports NAT and firewall traversal techniques to improve connectivity for devices in restricted networks.

I2P provides the possibility to select the transport per message using a *bid* based system, finding the decision which mechanism to use based on a set of inputs. I2P participants are similar to the participant assumed in this design located in restricted networks and I2P tries to increase connectivity using similar NAT traversal approaches. But the I2P design lacks extensibility to support additional mechanisms and does not use the full potential of supporting multiple transport mechanisms.

#### 4.5.4 BitTorrent Protocol and Obfuscation

BitTorrent<sup>26</sup> is a peer-to-peer protocol with the main focus on sharing data over the Internet and is best known for its use with file sharing but it is also used for other purposes like video streaming, and major companies rely on this technology to provide and enhance their services. BitTorrent itself is only the protocol specification described in a public domain document [Coh08]. This protocol is implemented by a large number of BitTorrent clients, adding additional and client-specific functionalities which can only be used with compliant client software also implementing these features. For the communication between clients, BitTorrent relies on TCP and the UDP-based uTorrent Transport Protocol (uTP) protocol specified in [Nor09].

BitTorrent is said to cause a high percentage of traffic on the Internet [Tor13] and therefore many ISPs apply filtering or traffic shaping techniques on BitTorrent traffic to reduce the network load caused by BitTorrent [XYK<sup>+</sup>08]. Several BitTorrent clients try to make communication resilient against this service degradation attempts applying traffic obfuscation techniques and make filtering of BitTorrent traffic harder and provide confidentiality and integrity for client communication.

Techniques used are protocol encryption (Protocol Encryption (PE)) using Message Stream Encryption (MSE) or Protocol Header Encryption (PHE). PHE was only used in early implementation approaches and but was abandoned and replaced by MSE. With these approaches, the protocol header (with PHE) or the whole data stream (MSE) exchanged between the clients is encrypted using a symmetric stream cipher like RC4. This approach is described in [LuTP06].

Using this approach, BitTorrent clients try to hide and obfuscate their traffic. Due to the large number of different clients, deploying such an approach is hard to achieve. In addition, different investigations have proven that protocol obfuscation and Bittorrent MSE is still detectable and therefore easy to classify and filter [HJ10].

## 4.6 Conclusion and Findings

The communication infrastructure presented in this chapter was tailored to the requirements of the GUNet peer-to-peer framework and its design principles and objectives. GUNet aims to provide secure and free communication for a decentralized architecture and a communication infrastructure has to support these objectives. As the main objectives to achieve we defined to provide connectivity in case of failing or missing communication infrastructure, improve connectivity between systems and counteracting limitations

---

<sup>26</sup> <http://www.bittorrent.org>

to end-to-end connectivity and provide secure communication resilient against attempts to degrade or block GUNet.

To establish the peer-to-peer overlay, GUNet typically uses existing communication infrastructure to connect peers. But in case no communication infrastructure is available or existing communication infrastructure is failing or degraded, GUNet can connect peers using ad hoc networking techniques based on Bluetooth or WLAN. These mechanisms allow peers to discover each other directly and to exchange data without requiring a deployed communication infrastructure. The DV component enables GUNet peers to communicate with peers when not directly connected with each other: with DV traffic between peers can be forwarded by intermediate peers via the peer-to-peer overlay. This approach is transparent for the peers and increases the connectivity between peers in the peer-to-peer overlay and the resilience of the network.

To counteract limitations to end-to-end connectivity and to increase connectivity for peers in restricted environments or affected by limitations imposed by middleboxes, GUNet supports NAT traversal techniques like UPnP and static port forwarding but also supports NAT hole punching techniques based on ICMP and UDP. In addition with the HTTP and HTTPS plugins GUNet supports to use proxy servers for peers limited by the enforced use of a proxy server. Peers not able to accept incoming connections can benefit from DV, enabling these peers to communicate with a larger set of peers in the peer-to-peer overlay.

To make GUNet resilient against censorship and service degradation, GUNet supports multiple transport mechanism to counteract such attempts. GUNet supports to communicate with other peers using TCP, UDP and HTTP and HTTPS. With the HTTP(S) plugin is it possible to tunnel GUNet traffic in a HTTP(S) tunnel and benefit from the use of a reverse proxy *hiding* a GUNet peer in a popular website. GUNet continuously monitors performance properties of communication mechanisms and improves performance and resilience by using the mechanism providing the best performance properties. GUNet's approach for transport selection will be discussed in detail in the next chapter.

Secure communication is ensured by treating every information received as untrusted and using information (for example about remote peers) only after validating those information and authenticating the communication partner. GUNet employs a sender decides approach where peers decide about allocated resources, accepted data rates and communication mechanisms locally without exchanging information or collaborating with remote peers to find these decisions. Communication between is secured with CORE service providing secure link encrypted communication between peers.

With GUNet's transport infrastructure providing communication based on these pillars, we can provide a profound foundation to establish communication between participants in a decentralized peer-to-peer system. It can provide high-performance communication as we saw in the evaluation where we showed that TRANSPORT can saturate a gigabit Ethernet link using off-the-shelf hardware. But it can also make communication more resilient using transport protocols providing lower data rates but other desirable properties. Future work can focus on providing additional transport mechanisms to make communication more robust against degradation and improve communication in restricted environments. An additional focus for future work should be the traversal of middleboxes and improve detection of degradation attempts to improve the resilience against service degradation.



## 5. ADDRESS SELECTION AND RESOURCE ALLOCATION IN DECENTRALIZED PEER-TO-PEER NETWORKS

A censorship-resistant and resilient communication infrastructure, as the one described in the previous chapter, provides huge benefits for decentralized peer-to-peer applications. With the communication infrastructure proposed in Section 4, overall connectivity between participants is improved by providing the possibility to create ad hoc networks, enabling users to communicate and exchange information even in environments without deployed communication infrastructure, connectivity for users in restricted environments is improved employing port mapping and NAT traversal techniques and degradation attempts are counteracted by supporting multiple transport mechanisms and providing the possibility to switch in case of service degradation or filtering.

So the proposed communication infrastructure for the GUNet peer-to-peer framework provides powerful features to overcome the limitations with respect to end-to-end connectivity analyzed in Section 2 and overlay connectivity analyzed in Section 3. But having such a resilient infrastructure does not automatically solve all problems, but instead creates new challenges: when a transport infrastructure provides the ability to switch between transport mechanisms and to build ad hoc networks, new questions arise: the transport infrastructure provides a set of various functionalities, but how can the performance of these mechanisms be evaluated? Which mechanisms are available at the moment? Which mechanism should be used in a specific situation? Which mechanism is the best to use in the current situation? How do we define *good* and how do we define *better*? And for which objective do we try to be good? When should we switch between the different mechanisms? How many resources should be allocated to a mechanism?

As we can see from these short overview over some possible questions arising in this context, it is not sufficient to just provide the functionality, but this functionality also has to be managed. In this chapter we will present the design and implementation of an automatic transport selection and resource allocation mechanism, designed to match the requirements and objectives of decentralized peer-to-peer systems. As a preparatory work for the design, we present a detailed analysis of the entities existing in the problem setting this mechanism is located in and based on these entities, we define the problem of address selection and resource allocation in decentralized networks. We define the objectives the mechanism has to achieve when solving this problem as well as the scope and limitations of the presented approach. Based on this problem definition, we present the design and the implementations of such a mechanism. The proposed mechanism is designed to match the requirements of peer-to-peer networks and pays respect to the specific properties of underlying transport mechanisms, possibly conflictive requirements of different applications working on top of a peer-to-peer framework, and resource restrictions present on a peer. We present three different approaches how to find a solution for the defined problem using a greedy heuristic, mathematical programming and machine learning techniques. Each of these approaches is considered due to the specific advantages it provides and we compare the benefits, drawbacks as well as performance and quality provided by each approach.

Parts of this chapter were previously published in [WOG14] and researched in col-

laborative work with other researchers and students. The design of the Reinforcement Learning (RL) solver is based on Fabian Oehlmann's work on the suitability of machine learning for bandwidth management in decentralized networks [Oeh14] and the analysis covering suitability of heuristics is based on Robert Schirmer's work on operations research approaches for allocation problems in decentralized networks [Sch13].

### 5.1 Background and Analysis of the Problem Setting

To motivate the problem of automatic transport selection and its relation to resource allocation, we start with an in detail analysis of the participating elements in a peer-to-peer system and how these elements interact. We do this with respect to the communication infrastructure and properties defined in the previous section and with a special focus on the requirements of peer-to-peer frameworks like the GUNet peer-to-peer framework described in Section 2.8.3. Based on the findings of this section, we elaborate the problem to solve, the objectives and challenges and define the requirements for a transport selection and resource allocation mechanism.

#### 5.1.1 Peers

As described in Section 2.8, a peer-to-peer network consists of a variable number of participants, the so called *peers*. The number of peers is not fix nor a priori known in most peer-to-peer networks. Peer-to-peer networks are designed to allow peers to join and leave at will and without prior coordination. Peers in a peer-to-peer network are expected to join and leave frequently, a behavior called *churn*. Fully decentralized peer-to-peer networks do not rely on any trusted, centralized coordination instances and therefore the process of joining and leaving the network is also not managed by a centralized instance and peers have to coordinate this process in a decentralized manner.

Each peer in a peer-to-peer network maintains a list of possible connection partners and tries to maintain connections to these peers. The set of peers it is connected with is dynamic. Due to churn, peers can join and leave the network without prior coordination. So each peer has to decide with which peers it can communicate with it wants to maintain a connection. This decision is not static and can be reconsidered over time since the set of available peers changes due to churn. Maintaining connections to a larger number of peers helps to increase connectivity and resilience in the peer-to-peer overlay and therefore stability in the peer-to-peer overlay. On the other hand is maintaining connections expensive for a peer since connections require resources like sockets and memory for buffers. Connections have to be maintained requiring computation time and network bandwidth to do so. Therefore, every peer has to find a trade-off between increasing the resilience in the overlay by maintaining a larger number of connections and saving resources by restricting the number of connections to a certain threshold. Each peer has to decide to which peer in his set of known peers it wants to maintain a connection to. This decisions can be based on peer-based properties: a peer can use information like prior uptime, amount of data exchanged and transmission quality provided with this peer to make this decision.

#### 5.1.2 Transport Mechanisms

One of the features of the communication infrastructure described in the previous chapter is to increase overall connectivity between peers and to counteract service degradation by providing support for multiple transport mechanisms. In the case of lacking or failing

infrastructure, the communication architecture can create ad hoc networks to communicate with other peers without managed communication infrastructure. To communicate with a peer is not directly connected to, the peer can use the DV plugin and communicate by relaying the traffic over other peers in the peer-to-peer overlay. In case of service degradation or censorship, the transport infrastructure can switch to a different transport mechanism to counteract the censorship. The transport mechanisms work on different layers of the ISO/OSI protocol stack. So on the one hand, GUNet supports multiple transport layer protocols like TCP and UDP and application layer protocols like HTTPS and on the other hand mechanisms like WLAN working directly on physical layer. Therefore, we do not use the term *transport protocols*, but instead the term *transport mechanisms* in the remainder of this discussion. Due to the intention these mechanisms originally came from, they have different properties and each single one of them is especially suitable for a different communication environment. Some of them are reliable, like TCP, others provide only unreliable communication like WLAN and UDP, plugins may be connection-oriented or connectionless or provide point-to-point or point-to-multipoint communication.

Each peer in a peer-to-peer network not only has to decide with which peers it wants to communicate, it also has to figure out which transport mechanism is the *best* to use in the current environment. Each mechanism has a specific use case and has different properties. When two peers supporting multiple transport mechanisms communicate with each other, the *best* available transport mechanism should be used.

### 5.1.3 Transport Mechanisms with Multiple Addresses

In addition to decide which transport mechanism to use, the address selection mechanism may also have to decide which address provided by a single transport mechanism to use. Depending on the respective transport mechanism, a transport mechanism can provide more than a single address, as described in Section 4.3.2. So for example the TCP and UDP plugins can provide both IPv4 and IPv6 addresses when the peer supports both IPv4 and IPv6 and a system may have multiple network interfaces, each equipped with one or more addresses. With the transport infrastructure proposed in the previous chapter, the plugin notifies the TRANSPORT service about all available addresses and the TRANSPORT service gives these addresses to the neighbor discovery and bootstrapping mechanisms to publish them in the peer-to-peer network. Depending on the transport plugins of the remote peer and the properties of the remote system, the remote peer then selects the appropriate address from the set of available addresses to connect to a peer.

So the problem of address selection is extended to select the peers to communicate with, select the most suitable transport mechanism and based on these decisions, select the most suitable transport address if a transport mechanism provides more than one address. As for the set of addresses provided by a single transport mechanism, the addresses may have different properties depending on the transport mechanism, so the address selection mechanism has to figure out the *best* address available from a set of addresses to communicate with a peer.

### 5.1.4 Network Scopes

With TCP's IPv4 and IPv6 addresses we saw that a single transport mechanism can provide more than one address, for example to differentiate between different protocol versions. But the number of available addresses can be even higher for a single transport mechanism. If a system contains multiple network interfaces or if the protocol used by the plugin provides this notion, a transport mechanism can provide different addresses for network

links located in different networks or *network scopes*. Most systems provide a virtual local pseudo network interface for local communication, the so called *loopback* interface. This interface is used by applications for network communication within the context of the same operating system. Depending on the protocol, additional network scopes are possible. With IPv6 additional network scopes are defined in [HD06]. IPv6 defines in addition to global and private addresses known from IPv4 six additional network scopes: *interface-local*, *link-local*, *admin-local*, *site-local*, *organization-local* and *global*. The idea behind this approach is to make it easier to restrict certain traffic to an administrative domain. This approach simplifies controlling and routing since traffic can be filtered or not forwarded at the boundary of each administrative domain. So each IPv6 enabled network interface can have an address assigned in each of these network scopes. With IPv4, we can have a global IPv4 address and an address for the local loopback interface and, when affected by NAT, an additional public IP addresses. In addition, a system can have multiple network interfaces. A standard notebook may already have three active network interfaces: the loopback interface, an Ethernet network interface and a wireless WLAN interface. Each of these interfaces may support IPv4 and IPv6 with multiple addresses in different network scopes.

For a transport selection and resource allocation mechanism for decentralized peer-to-peer networks, it is important to know *where* a peer is located. A peer is communicating with various other peers which may be *close* and *cheap* to talk to from a networking perspective or they may be *far* and it is *expensive* to communicate with these peer. A remote peer may be reachable over an expensive and slow WAN dial-up connection, a lossy wireless connection or over a fast wired Ethernet connection in the local LAN. Communication with peers in different network scopes can have different properties. Within the LAN, we can have very fast communication with a high throughput since LANs provide communication with a low latency and high bandwidth, whereas WAN connections commonly have a smaller line speed and communicating with peers connected over multiple intermediate hops on the Internet will experience a higher latency. Communication in a WLAN may have a higher bandwidth compared to a WAN connection but also a higher latency and loss rate compared to the LAN.

Therefore, a transport selection mechanism has to be aware of different kinds of addresses and how *well* each single one of them is suitable for communication with a peer. The proposed address selections mechanism must have a notion of *network scopes* to distinguish between the different addresses and the networks these addresses are located in. Based on this notion, it can improve its decision which address is used to communicate with a peer.

### 5.1.5 Bandwidth Restrictions for Network Scopes

Based on the notion of network scopes introduced in the previous section, we have to introduce an additional aspect related to these network scopes with respect to the resources available in these networks. In a LAN, the bandwidth available may be virtually unlimited whereas in other network scopes bandwidth can be limited or restricted by policy. Some network scopes can only provide a certain limited line speed, for example when using a dial up connection. So when communicating with addresses in these network scopes, the consumption of the scarce resource has to obey certain limits to not over-utilize the link available. This is even more important since the peer-to-peer application may not be the only application on a system using a network link. Therefore, bandwidth available must not be fully exploited at full line speed to prevent cannibalization of other applications on

the same system.

A second aspect to be aware of is that for certain network scopes, like LAN, the amount of bandwidth consumed may not matter, while this is not true for other network scopes: when using a mobile 3G or 4G Internet connection with a data plan, the bandwidth consumed will matter to the user. The same issue applies to users using a broadband access with data capping, as described with economic motivations in Section 1 and an application fully exploiting the bandwidth available.

A user may want to restrict the application with respect to the amount of bandwidth it consumes. For every network scope, the user may want to provide a *quota*, restricting the amount of bandwidth the application consumes in a certain time period. This time interval can vary: a user may want to configure a certain amount of bandwidth to use per month for his 3G connection (to not exceed his data plan) and on the other hand restrict bandwidth consumption per second (to limit the transmission speed and not penalize other applications). This restriction may be different per network scope: a user can define a high quota for his LAN scope but restrict the bandwidth consumption in the WAN scope to a ratio or total amount of the nominal par of bandwidth available. In addition, a user even may want to configure separate quotas for incoming and outgoing traffic, since network connections can be asymmetric.

### 5.1.6 Resource Allocation and Address Selection

An additional aspect to selecting a suitable address is the aspect of sharing and allocating scarce resources to peers. Some resources on a system or in the network can be limited. When deciding with which peers to communicate and which address is suitable, the address selection mechanism also has to distribute the available resources to peers. The network bandwidth available can be one of these scarce resources, other resources may be local resources like network sockets, memory (when a transport mechanism requires buffers) or computation time (when a transport mechanism is especially expensive with respect to its computational requirements). Therefore, resource allocation is an imminent aspect when having to select the *best* address to communicate with a peer. The process of address selection and resource allocation are tightly intertwined: An address must be evaluated with respect to its resource consumption in proportion to the resources availability. Selecting a set of suitable addresses to communicate to a set of peers depends on the available of resources to communicate with this peers. Then again the decision to allocate resources strongly depends on the set of addresses selected from the set addresses available to communicate with the peers: choosing a slightly worse address with respect to its properties but with more resources for communication can result in an improved overall performance. Therefore, finding a solution optimal with respect to address selection and resource allocation will strongly improve the overall quality of the solution.

### 5.1.7 Transport Properties

Different transport mechanisms and also the addresses provided by a mechanism can have different properties. In Section 5.1.2, we introduced the different transport mechanisms. As described there, a mechanism is a generalization since these mechanisms have completely different use cases or purposes, use different transport protocols and may even work on different layers of the protocol stack. Based on these different concepts, it is only natural that these mechanism behave differently. In Section 5.1.2 we described that

even for the same transport mechanism addresses can behave differently and have properties depending on the realization of the mechanism. A transport selection and resource allocation mechanism must pay respect to these differences: in previous sections we used the notion of the *best* or *suitable* addresses. We used this abstract generalization since at a certain point the proposed address selection and resource allocation mechanism has to evaluate and define what *good* is and when a mechanism or an address is *better* than a second one. Therefore, we have to be able to evaluate and compare transport mechanisms to figure out the *best* solution for the problem described in the previous sections. We need a way to evaluate and compare between different addresses of the same or different transport mechanisms with respect to various different aspects of their behavior.

To be able to collect information about the behavior of transport mechanisms, to be able to evaluate mechanisms and addresses and to compare them with each other, we introduce in the following sections the idea of so called *transport properties*. These transport properties capture the properties and behavior of a transport mechanism or address with numerical values with respect to different aspects of a transport mechanism. By describing these properties with numerical values, we can compare between different addresses and based on this comparison evaluate the quality of different transports to find the *best* solution for the address selection and resource allocation problem described in this chapter.

Transport properties describe various aspects how a transport mechanism can be categorized. We propose a generic approach, which enables us to extend and add additional aspects later when required and by quantifying these properties we make them comparable and introduce an order in the set of transport addresses available.

Initially, we distinguish between different kinds of properties a transport mechanism can have. A property can be dynamic, so it can change over time, or it can be static. A transport property can be influenced or implied by the way the transport mechanism is working. This is what we call an *intrinsic transport property*. On the other hand, we define *extrinsic transport properties* as properties which are influenced by external events and the environment. These properties are not caused or influenced by the transport mechanism itself. A specific property of a transport service can now be static or dynamic and intrinsic or extrinsic.

### 5.1.7.1 Static and Dynamic Transport Properties

When analyzing properties of transport mechanisms, we can characterize them based on their temporal behavior. To classify properties of transport mechanisms based on temporal behavior, we can distinguish between properties constant over time and properties dynamic properties changing their values over time.

A first example for static a static property is the *overlay hop distance* for a TCP or UDP connection where communication partners are always directly connected with each other with an overlay hop distance of 1. For a connection established with the DV transport mechanism, described in Section 4.3.18, where traffic is routed over overlay peers, this assumption is not true: peers must not be directly connected on transport layer but the peers can be connected by a number of intermediate overlay hops. In addition, the number of intermediate peers can be dynamic: peers are exchanging routing information during operation and therefore new and possibly shorter routes may be learned and a peer may switch a shorter and therefore better route. For TCP or UDP connections it is sufficient to assume a static value fix over time and fix for all addresses for this property for example the number of IP hops. For the DV transport mechanism, the overlay hop distance is

known a priori and can change over time.

A second example is the energy consumption required to send a specific amount of data using a transport mechanism. This property is particularly interesting for mechanisms using wireless transmission, like Bluetooth or WLAN. For a transmission over a wired Ethernet connection using for example TCP, this property is hard to determine in a normal user environment. But for a WLAN connection, this property is beneficial to know if the remote peer is far away and a high amount of sending power is required to transmit data to this peer. Due to the mobility of both nodes or additional devices coming in range or leaving, this value can change over time, requiring a frequent re-evaluation of the current situation.

### 5.1.7.2 Intrinsic and Extrinsic Transport Properties

A second aspect to be considered when categorizing possible properties of transport mechanism is if the transport mechanism itself is responsible for the specific property due to its design, architecture or implementation or if the property is caused or implied by the environment the mechanism is operating in. If a property is caused or implied by the transport mechanism itself, we call it an *intrinsic property*. If the property is influenced or caused by external factors, we call the property an *extrinsic property*. To distinguish between properties being intrinsic and extrinsic is important when transport mechanisms have to be evaluated and compared with each other. A intrinsic property can be analyzed by inspecting the transport mechanism and its design and implementation off-line. In addition, it is possible to evaluate intrinsic properties with a local view of the transport mechanism and its state. To evaluate extrinsic properties, additional information or measurements about the communication partner, intermediate systems or the current state of the communication channel the mechanism is using are required.

An example for an intrinsic property is the computational overhead caused when using encryption on transport layer. If a transport mechanism uses a layer of encryption, like for example the HTTPS transport plugin described in Section 4.3.15, this causes higher computational *cost* and therefore transport performance can be lower. So transport mechanisms using additional encryption on transport should therefore only be used when their functionality is required since using these mechanisms shortens battery lifetime on mobile devices and transport performance is less efficient in comparison to other transport mechanisms. This computational overhead is an intrinsic property caused by the design and functionality of the transport mechanism and the impact can be evaluated by inspecting the design and implementation of the transport mechanism.

An extrinsic transport property, not caused or influenced by the transport mechanism, is for example the network delay for a network connection, so the time between sending a data message and the message arriving at the destination host. This network delay is influenced by a large number of aspects: the line speed of the network connection, other applications the connection is shared with, the number of the intermediate hosts routing the data to the destination and the current load of these nodes, possible failures and misconfiguration and many many more aspects have an impact on the time required to send a network message from one host and have it arrive at its destination. Not only is this property dynamic and therefore has to be reevaluated frequently, it is also not directly caused by the mechanism but the environment the transport mechanism is operating in. Quantifying an extrinsic property can therefore be a non-trivial task and can require expensive active measurements and this property is in addition different for each communication partner and every transport address available.

### 5.1.8 Application Requirements

A transport selection and resource allocation mechanism has the objective to find the set of *best* transport mechanisms to communicate with set of communication partners and it has to distribute available resources among the communication partners to achieve an *optimal* allocation. As in the previous sections, the question arises how to define and choose the *best* transport mechanism and how to assign the required resources to achieve an *optimal* resource allocation.

Since the communication infrastructure discussed in this work only provides the infrastructure for higher layer applications to communicate in the overlay network, the objectives for an optimal address selection and resource selection strategy depend on the applications using the communication infrastructure and their requirements with respect to communication. A voice or video communication application requires low latency communication with the communication partner to provide a convenient user experience. If the latency is too high, communication with the human partner is delayed and unresponsive, resulting in an unsatisfying user experience. On the other hand a telephony application requires only a small amount of bandwidth: codecs optimized for voice transmission and telephony, like G.729 codec [US96], can work with as little as 8 KiB/s. So telephony connections require a low latency connection and do not rely on high bandwidth between communication partners. In addition, due to the design of the codecs, they may even tolerate a certain amount of loss in the communication without impacting the application's quality of service and user experience.

A file transmission application on the other hand has communication requirements completely opposing to the requirements of a telephony application. A file transmission application wants to transfer the data as fast as possible to finish the operation as soon as possible. So a file sharing application requires a large amount of bandwidth but does not care much about the latency of the communication. In addition, such applications often tolerate no loss or corruption in the transmission since the data are required to be transferred completely and correctly.

As we can see from these two completely different examples, the requirements applications have with respect to communication properties can be very different. A transport selection and resource allocation mechanism to be used in an application framework has to pay attention and comply with the requirements applications have. A suitable mechanism must adapt its selection and allocation strategy to the applications' requirements. To realize this, interaction between the selection and allocation mechanism and the higher layer applications is required to enable applications to specify their requirements with respect to the communication properties.

#### 5.1.8.1 Application Preferences for Communication Partners

An application following the client/server paradigm only has to find one best address to communicate with the server and has to assign the resources accordingly to this connection. If such an application supports more than one transport mechanism, the application may have to figure out the most suitable address out of a set of addresses. But an application following the peer-to-peer paradigm does not communicate with only one but with many communication partners in parallel and therefore has to specify their *preferences* for peers in a larger set of peers. In a peer-to-peer environment, not every communication partner may be equally valuable to communicate with for the applications for various reasons: the peer can be the desired communication destination, a peer can be more valuable since it provides better connection properties, for example a lower delay or a larger band-

width, a peer can be more valuable since previous application operations were successfully performed or less valuable since previous operations failed. So in general, an application may have a higher preference for a peer it can successfully collaborate with and a lower preference for less valuable peers.

#### 5.1.8.2 Application Feedback

Besides specifying their requirements and preferences with respect to communication with other peers, applications can also give feedback about how satisfying the current solution is. The application can specify how satisfied it is with the current solution with respect to a communication partner and transport properties. While preferences and requirements are specified to allow the address selection mechanism to find a solution aligned with the requirements of the application, feedback is given after a solution was found and applied with the system to allow the system to adapt its solution strategy.

#### 5.1.8.3 Supporting and Balancing Multiple Applications

When designing a transport selection and resource assignment mechanism for decentralized peer-to-peer frameworks supporting more than one application to use a shared transport infrastructure, the challenge arises to strike a balance between the requirements and preferences of the different applications. If the peer-to-peer application is directly and tightly coupled with the peer-to-peer framework each peer-to-peer application employs their own *instance* of the framework. These instances are independent from each other and may run in parallel without interfering or interacting. A transport selection and resource assignment mechanism in a tightly coupled framework only has to pay respect to a single application and its peer preferences and communication requirements. With the GNUnet peer-to-peer framework all applications (on the same system or ran by the same user) use the same instance of the GNUnet peer-to-peer framework. So there is only one GNUnet *instance* running and all applications use and share this instance. With this design approach, a framework does not provide communication facilities just for one upper layer application but for a larger number number of applications. A transport selection and resource allocation mechanism in such a system therefore has to handle and fulfill the possibly conflictive requirements of a large and varying number of applications.

A second aspect to consider is that a framework like GNUnet is designed to provide the foundation for developers to implement their of decentralized networking application. As a consequence we do not a priori know what kind of application will use the framework and which requirements and preferences such an application will have. The mechanism designed in this chapter will therefore have to be generic and support differing kinds of applications with differing and opposing requirements and preferences. The applications and their requirements and preferences are not a priori known at the time when the framework is developed and can change over time when the applications are running. Since the applications are independent from each other and do not typically coordinate or communicate their requirements and preferences, the address selection and allocation mechanism has the goal to find a balance between the different applications.

#### 5.1.9 Summary

To summarize this evaluation and to give a description of the problem being solved in this chapter, we summarize here the problem of address selection and resource alloca-

tion in decentralized peer-to-peer networks with special focus on the use in peer-to-peer frameworks like the GUNet peer-to-peer framework.

In a peer-to-peer application, the transport infrastructure has to provide underlay communication with a certain set of peers specified by higher layer applications requesting to establish a connection with these peers. For each communication partner a set of addresses is available to establish a connection with this peer. Addresses have different properties and can be located in different networking scopes. To restrict the resource consumption of the peer-to-peer application, the user can specify resource restrictions including the resources consumed in the respective networking scopes. In the given peer-to-peer application, it is possible to have multiple applications using the transport infrastructure in parallel. These applications specify which communication properties they cherish to communicate with communication partners and give feedback if the current solution is satisfying their requirements.

A solution to this problem is the set of addresses having resources assigned to communicate with requested peers. From the given set of addresses, for each communication partner an address has to be select and provided with a share of the resources available to provide connectivity optimally satisfying the requirements and preferences of the applications. Since the set of peers, the addresses available and their properties may change over time, this solution has to be recalculated when the environment changes.

## **5.2 Design and Architecture**

In the previous sections, we analyzed the requirements for a transport selection and resource allocation mechanism particularly suited for the requirements of decentralized peer-to-peer networks and formulated the problem to solve. Based on this analysis we will now present our design and architecture of the proposed mechanism and the different solution approaches used to find a solution to the problem: a greedy heuristic, treating the problem as mathematical optimization problem and using a machine learning to learn a selection and allocation strategy. After describing our proposed design we will discuss in the following sections related work and describe the implementation of our approach.

The goal of the address selection and resource allocation mechanism is to provide a subset from the set of addresses the mechanisms evaluated to be the best in the current situation to achieve best possible performance for the applications according to the preferences specified by the applications and with respect to the current properties each address has and the currently given resource restrictions. To achieve this objective, the tasks are two-fold: on the one hand the best address for each peer has to be determined and on the other hand the available resources have to be distributed among these addresses. These tasks are tightly intertwined: choosing an address with good properties but being able to only assign few resources is a worse decision than selecting a slightly worse address but being able to provide a larger amount of resources. Therefore, both aspects have to be considered in an integrated approach. A solution found for the problem has to be reconsidered when the environment changes due to peers joining or leaving, applications changing their preferences or address properties changing.

The mechanism designed here has to be tightly integrated with its environment: on the one hand higher layer applications have to interact with mechanism to announce their preferences and on the other hand the mechanism has to collaborate with the transport underlay. The transport underlay provides information about addresses available including the properties of these addresses. When a new addresses or transport sessions are available or become unavailable, the transport infrastructure notifies the address selection about

this change. When the transport infrastructure is requested to connect to a remote peer, as described in Section 4.3.11.7, it requests an address to be suggested for this neighbor. The address selection mechanism in return provides the transport underlay infrastructure with information about the address to use to communicate with a neighbor and the amount of resources assigned to this address.

### 5.2.1 Input for the Transport Selection

As an input for the transport selection problem, we have the set of peers known to a peer and the addresses of the transport mechanisms to establish connections with the remote peers. This information is provided by the transport underlay. To compare the different addresses, the transport underlay provides information about the intrinsic and extrinsic properties and network information for each address as soon as this information is available or updated. The transport underlay also notifies the mechanism about changes and updates to an address. Since the transport underlay requests addresses for neighbors it wants to establish a connection with, the transport selection mechanism has to know the set of neighbors addresses are requested for.

Information about the resource restrictions and resource restrictions for the different network scopes are provided by the user specified in a configuration or predefined by the developer. The transport selection mechanism can use this information to find a solution obeying the current resource restrictions of the system.

Applications using the peer-to-peer system to communicate with other peers specify their preferences to the transport selection mechanism. Applications specify their preference for a specific peer and a certain preference properties particularly useful or important for them. The transport selection mechanism uses these information to find a solution matching the particular requirements of the applications. In addition, applications give feedback to the mechanism how satisfying the current solution is with respect to peers and properties. This allows the mechanism to adapt its solution strategy to meet the requirements better.

### 5.2.2 Output from Transport Selection

A solution to the transport selection problem (in the remainder of the problem called an *allocation*) is the set of addresses selected by the address selection mechanism to be used to communicate with the requested peers together with the amount of resources allocated to each of these addresses. This information is provided to the transport infrastructure using the addresses to establish connections to remote peers and at the same time to enforce the resource limits as described in Section 4.3.11.13. Addresses selected for communication are called *active* or *selected* in the remainder of this document.

### 5.2.3 Objectives for Transport Selection and Resource Allocation

When finding such an allocation, the mechanism has to determine the best address for each communication partner and distribute the resources available among these addresses. At the same time, the mechanism has to obey the resource restrictions, ensure usability of the communication and optimize the solution to satisfy the application requirements. When finding a solution, the transport selection and resource allocation mechanism has to follow the following objectives:

**Usability of Communication:**

To provide useful communication between participants, a minimum amount of resources is required for each connection to a communication partner as just setting up, maintaining and managing the connection creates a certain overhead. Thus, if an address is selected at all, at least a certain minimum amount of resources has to be assigned.

### **Diversity of Communication Partners:**

Communicating with a larger number of participants increases the resilience of the peer-to-peer network. Therefore, the resulting allocation has to distribute resources over a range of peers instead of assigning resources only to a tiny number of peers.

### **Relativity of Resources to Peers:**

Resources have to 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 has to get more resources assigned than a peer not valued by applications.

### **Utilization of Resources:**

Available resources should be fully allocated to allow participants to achieve maximum utilization when communicating. Depending on the application, applications are still free to not utilize the allocated resources.

**Austerity when Using Resources:** The resources provided by the system have to be used economically and communication performance maximized. Transports with high communication overhead or resource requirements should be avoided to minimize useless resource consumption and maximize application performance by preferring transports with low resource consumption.

**Stability of Connections:** Allocations have to exhibit some stability when using connections to minimize transport initialization overheads and provide predictable performance to applications. Establishing and switching between connections creates communication overhead, consumes resources and influences application communication, while using a connection for a longer period can improve performance, for example when using congestion or flow control.

### **5.2.4 Scope and Limitations**

The mechanism proposed in this work has a focus on finding set of addresses to communicate with a set of neighbors for a peer. We assume that the preferences with *which* peers to communicate are specified by the higher layer applications and are issued to the transport underlay responsible to establish connections. The address selection mechanism does not decide which neighbors to communicate with, but it can disconnect peers when no suitable allocation can be found according to the objectives defined in Section 5.2.2. In this work we do not specify how it is determined with which neighbors connections are established. This task is left to higher layer applications such as GUNet's TOPOLOGY daemon.

In this work, we assume that only a single address per peer has to be determined. This is based on the assumption that maintaining multiple connections at the same time consumes too many resources and the transport underlay should prefer to switch between different addresses if a single address performs not according to the requirements specified by the applications.

Our approach does not depend on any centralized instances coordinating the process of bandwidth allocation or address selection. Every participant in the networks finds this allocation on his own. To ensure privacy and prevent leaking of peer specific information as well as to prevent manipulation or DoS from remote peers in the network, the peer does not exchange allocation information with other peers. The solution found by our ATS approach is good from the local view of the peer, we do not try to achieve a globally optimal solution for all peers. Since we assume that the network can contain malicious participants in an unbounded fraction to the number of total peers, this is an restriction to strengthen the security of our approach.

### 5.3 Input Normalization and Correlation

Inputs to the address selection and resource allocation process originate from different sources like different applications specifying their preferences or giving feedback or from different transport plugins notifying the address selection mechanism about the performance properties of the addresses currently available. Different applications may use different value ranges to specify their inputs. One application can specify its preferences using a value range from  $[0..10]$ , with 0 indicating a low preference and 10 indicating a high preference, while a second application may use a range from  $[100..1000000]$ .

The same issue exists for performance property values provided by the TRANSPORT service and TRANSPORT plugins. For different performance values different value ranges can be used: a performance metric containing information about the overlay hop distance can use a range from  $[1..10]$  hops, while transmission delay could be in a range from  $[1..1000]$  ms.

To be able to use these inputs, these values have to be transformed to a common and known domain to be able to compare these values, perform computations and evaluate addresses. Therefore, the address selection mechanism provides an normalization component providing normalized preference and performance values to be used when solving the problem. The address selection mechanism normalization component takes the uncorrelated and unnormalized inputs from applications and the transport underlay and normalizes and correlates these values into a common value domain and provides these values to the solvers where they can be used to compare preferences and performance information and to perform computations using these values.

#### 5.3.1 Preference Normalization and Correlation

With preference *normalization*, we transform the absolute preference values to a common value range and with *correlation* make the values comparable between different peers and addresses. In our approach, every application can specify their preference for a specific preference type and a specific peer using an absolute floating number value. This floating number represents the application's preference. The ATS mechanism has to specify for every peer how important a specific preference type is for the applications as an output of the normalization and correlation process. For every peer and every preference type, a relative value represents with a floating number this preference in the range of  $[0 \dots 1]$ , with a lower value indicating a lower preference for this peer. If no preference is specified for a peer we use a default value of 0.

The basic idea for correlation and normalization is that every application issues which preference types for which peers are important for this application. We can normalize the

preference types  $p \in P$  for the application  $c \in C$  by calculating a relative preference  $f_{c,p}$  for each peer  $p$  by normalizing the absolute preference values  $a_{c,p}$  with:

$$f_{c,p} = \frac{a_{c,p}}{\sum_{p \in P} a_{c,p}}$$

with  $a_{c,p} = 0$  if no preference was specified for a peer. As a consequence we have normalized the absolute preference for an application to a range of  $f_{c,p} = [0 \dots 1]$  and  $\sum_{p \in P} f_{c,p} = 1$  if an absolute preference with  $a_{c,p} \neq 0$  exists. This range was chosen since preference values are often used with operations in solvers where no preferences should lead to a (multiplicative) result of 0.

After normalizing these values, we have to correlate the different relative values between the different applications to be able to provide a single relative preference value for each peer and each preference type. In our approach we correlate these values with the preferences specified by other applications by calculating a unified relative value  $f_p$  for a peer  $p$  as:

$$f_p = \frac{\sum_{c \in C} f_{c,p}}{\sum_{c \in C} \sum_{p \in P} f_{c,p}}$$

with  $f_p = 0$  if no relative preference was specified for a peer. As a consequence we have normalized and correlated the absolute preference values for an application to a range of  $r_p = [0 \dots 1]$  and  $\sum_{p \in P} f_p = 1$ .

In addition, we support the notion of preference *aging* to reflect that older preferences are less important than currently issued preference requests. We use a preference aging factor and update the current absolute preference value by multiplying the absolute value  $a_{c,p}$  with the aging factor  $f_{aging}$  in regular intervals:

$$a_{c,p} = a_{c,p} \cdot f_{aging}$$

After applying the preference aging to the absolute preference values, the NORMALIZATION component has to update the relative preference values to reflect this change.

### 5.3.2 Performance Property Normalization

The transport underlay and the different transport plugins provide the ATS service with information about the different addresses available to communicate with a remote peer. As described in Section 5.1.7 have different addresses and mechanisms different performance properties. To be able to use these information in the solver, we have to normalize the values for the different properties to a common value range: all values for a specific property have to be normalized to same range. Contrary to application preferences, a correlation between different property types is not required. For performance properties, we assume that smaller values indicate better performance properties. If a property has a different semantic (a smaller value indicating worse performance), the source (e.g. the TRANSPORT service) has to provide the inverse value to comply with this semantic.

To normalize property values we inspect all values of a specific property, determine the minimum and maximum values occurring, and based on these values, normalize the remaining *absolute* values  $a$  to a common range in  $[1..2]$ . This range is different to the range used for application preferences since property values are often used by the solvers in

(multiplicative) operations where a multiplication with the minimum relative value should not imply an end result of 0.

So based on the minimum value  $a_{min}$  and  $a_{max}$ , we calculate the *relative* property value  $r$  as:

$$r = \frac{a_{max} - 2a_{min} + a}{a_{max} - a_{min}}$$

When a new property value has to be normalized, we first have to check, if the value is smaller than the current value  $a_{min}$  or larger than the current value  $a_{max}$ . In these cases, we have to re-normalize all currently known relative values because otherwise the values are not any longer correlated correctly with the current value range. If only one absolute property value is known, we assume this value to be the maximum  $a_{max}$  and use a default of  $a_{min} = 0.0$ .

## 5.4 The Greedy Heuristic Solver

The first proposed approach to find a solution to the address selection and resource allocation problem is a fast greedy heuristic. This heuristic is based on the idea to distribute resources proportionally to communication partners according to the importance higher layer applications specify for the respective partner.

The heuristic solver is based on the idea of regarding to the different network scopes as *buckets* of resources. The heuristic selects the best address to be used with each communication partner; bandwidth in each bucket is distributed to the addresses proportionally to how important a peer is for applications. When the transport underlay requests an address for a peer, the heuristic selects the best address available for a peer from the set of addresses available. It selects the best address by comparing the properties provided by the transport underlay (*performance*, *austerity*) and choosing the address with the best performance properties. When an active address is existing for a peer, a different address is only activated when the currently active address was used for a minimum amount of time (*stability*). When an address is selected, the heuristic checks if in the network scope this address belongs to sufficient resources are available (*usability*). A new address is only selected if a minimum amount of bandwidth for all active addresses in this scope can be provided (*diversity*). If sufficient bandwidth for all addresses is available, the resources in the scope are distributed among the selected addresses in this network scope. Initially, every address in the scope gets the required minimum bandwidth assigned (*usability*). The remaining bandwidth is then distributed among all addresses according to the preferences the higher layer applications have specified with respect to bandwidth requirements for this peer (*relativity*, *utilization*). The heuristic can also explicitly tell the underlay to disconnect from a peer if not sufficient resources can be provided or no suitable address is available.

### 5.4.1 Design of the Solver

With the heuristic solver, address selection and resource allocation is realized in a greedy approach, so the heuristic first selects the best address to use and in the next step distributes resources. Whenever the transport underlay requests an address for a peer, the heuristic has to find the address to activate and assign resources to this address. Whenever addresses are requested or address requests are canceled and addresses and sessions are added or removed, the heuristic has to reconsider its current allocation and rerun the

address selection process. As a consequence of activating addresses or deactivating addresses, the heuristic has to redistribute resources in the respective network scopes. When addresses and addresses' performance properties are updated, the solver has to reconsider the address selection for the respective peer and when the active address changed redistribute resources in the network scopes of the current and previously active address. When (normalized) application preferences change, the heuristic has to redistribute resources in all network scopes to reflect the changes in the allocated resources.

#### 5.4.1.1 Address Selection

When the solver has to select an address for a peer, it analyzes all addresses available for the peer and compares the address properties of the addresses. While comparing addresses, the heuristic remembers the addresses it currently designates as the best address and replaces this address if it finds a better address. When done, address selection returns the address it evaluated to be the *best* address to communicate with the peer.

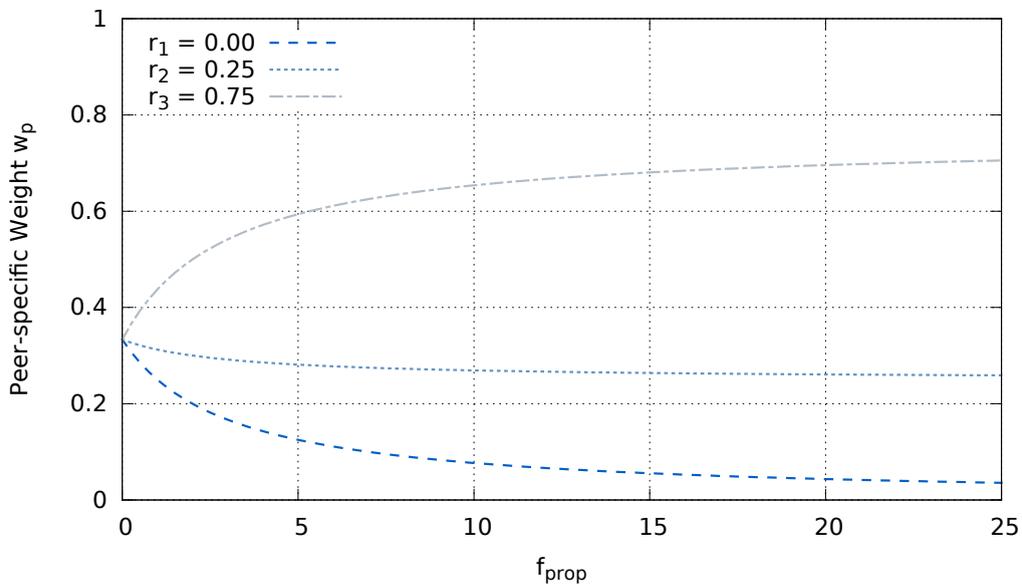
For every address, the solver checks if the address can be activated in the network scope it is located in. Hereby the heuristic ensures that with this address activated in the scope all addresses can get at least the minimum bandwidth  $b_{\min}$  assigned. If the current address is already active, the solver checks how long the address was active. Addresses are not switched if the connection was just established to ensure connection stability and prevent frequent address switches. Here the solver uses the connection stability factor  $f_s$ : if the connection was active for less than  $1 \cdot f_s$  seconds, the active address is kept as active. Address selection is aborted and the heuristic returns the active address. After checking if an address can be activated at all to ensure usability and checking an address' activation time to ensure connection stability, the solver starts to compare the properties of the current address with the address it has stored as the best address available. This comparison is based on the normalized address performance properties as described in Section 5.3.2. If the current address provides better performance, the solver replaces the best address stored with the current address, otherwise the best address is kept. After evaluating all addresses, the best address or no address (if no address can be provided) is returned. Based on this selection, resources allocation has to redistribute resources in network scopes where addresses were activated or deactivated (due to an address switch).

#### 5.4.1.2 Resource Allocation

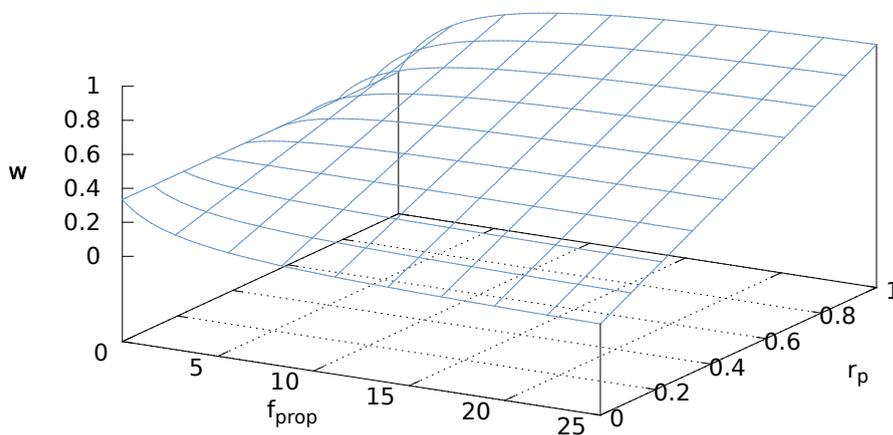
When the heuristic has to distribute the bandwidth within a network scope, it starts with first assigning a minimum amount of resources  $b_{\min}$  to all addresses selected as active in this network scope. The remaining amount of bandwidth available in this network scope is then distributed to the active addresses according to the preferences the applications have specified with respect to bandwidth for the peers. Here the heuristic uses the normalized relative preference values  $r_p$  as described in Section 5.3.1. To distribute the remaining bandwidth available in the current scope, the solver analyzes all active addresses in the network scope and evaluates the relative preferences  $r_p$  applications have specified with respect to bandwidth requirements for the peer the address belongs to. Since the normalized relative preference values represent the application preferences for all peers and not only the peers with active addresses in the current network scope, the solver has to re-align the relative preferences to reflect only the preferences for addresses in the current network scope. If no application specified a preferences for a certain peer, the default value  $r_{\text{default}} = 0.0$  is used. The heuristic calculates for every active address in the network scope a preference weight  $w_a$  used to assign bandwidth to addresses based on the relative

preferences for bandwidth and the total weight  $w_t$  reflecting the sum all of bandwidth preferences for peers with active addresses in the scope.

To distribute the remainder of the available bandwidth according to the preferences, the solver uses the weight  $w_a$  and an additional *proportionality factor*  $f_{prop}$  with  $f_{prop} \geq 0$  to respect the application preferences. The closer the proportionality factor  $f_{prop}$  is to 0, the fairer and with less respect to preferences is bandwidth distributed among the addresses. The bigger this value  $f_{prop}$  is, the more respect is paid to the preferences. As a default, we define in our implementation the value as  $f_{default} = 2.0$ . The impact of different values for  $f_{prop}$  is depicted in Figure 5.1.



(a) Impact of Proportionality Factor  $f_{prop}$  on Weights for Exemplary Relative Weights  $r_1, r_2, r_3$



(b) Impact of Proportionality Factor  $f_{prop}$  on Weights

**Fig. 5.1:** Impact of the Proportionality Factor  $f_{prop}$  on Weights

The bandwidth  $b_{a,in/out}$  assigned to every active address  $a$  from the set of active addresses  $A$  in a network scope  $n$ , providing an inbound amount of bandwidth  $b_{n,in}$  and outbound of  $b_{n,out}$ , is calculated as:

$$\begin{aligned}
 b_{\text{remainder},out} &= b_{n,out} - b_{\text{min}} \cdot |A| \\
 b_{\text{remainder},in} &= b_{n,in} - b_{\text{min}} \cdot |A| \\
 w_a &= 1.0 + f_{\text{prop}} \cdot r_p \\
 w_t &= |A| + f_{\text{prop}} \cdot \sum_{a \in A} r_p \\
 b_{a,out} &= b_{\text{min}} + \frac{w_a}{w_t} \cdot b_{\text{remainder},out} \\
 b_{a,in} &= b_{\text{min}} + \frac{w_a}{w_t} \cdot b_{\text{remainder},in}
 \end{aligned}$$

#### 5.4.2 Discussion

Heuristics are useful to find good solutions for problems where solving a problem is expensive and finding an optimal solution is hard. The heuristic presented here can compute the address selection and the resource allocation very fast even for a large number of peers, addresses and address scopes. This is beneficial due to the expected frequent changes in the problem caused by peers joining and leaving and updated properties. On the other hand does this approach lack the idea of providing a combined solution for the selection and allocation process. The heuristic first chooses an address and assigns resources in the next step without paying respect to the amount of resources available. Since there is no adaption with this approach, the heuristic does not benefit from the requirement to solve the problem repeatedly. The usability and applicability of different types of heuristics to the address selection and resource allocation problem is more in detail discussed in [Sch13].

### 5.5 The Mixed Integer Linear Programming Solver

To combine address selection and resource allocation with an integrated approach, we propose the use of mathematical programming and linear optimization to solve the address selection and resource allocation problem. We treat and formulate the problem as an optimization problem and use linear constraint programming to find an solution optimal by maximizing a set of objectives and obeying a set of constraints.

In this chapter, the transformation from the model based on mathematical sets into a linear optimization problem is introduced. Based on the model described in Section 5.2, we formulate a mathematical optimization problem to be solved to find the optimal allocation set in the given problem setting. For this transformation, we have to transform the restrictions and properties of the given problem into linear constraints and craft a linear objective function reflecting the objectives an optimal solution of the given problem has to achieve as determined in Section 5.2.3.

#### 5.5.1 Linear Programming

Linear programming is a standard technique established for several decades. It originates from the field of Operations Research (OR) and was originally used to optimize production and planning processes. The field of OR is well covered and several well-understood

algorithms to solve linear programming problems exist. A property of linear optimization is that solutions found are provably optimal if such solutions exist and if no solutions to the problem exist this is mathematically ensured.

Linear programming tries to optimize the solution for a given *objective function* subject to a set of *constraints*. Both the objective function and the constraints are formulated as linear equations. The goal of linear programming is to maximize the value of the objective function within the constraints given by the constraint equations. To minimize a problem, the problem is maximized with the objective function multiplied with  $-1$ . A linear programming problem can be seen geometrically, where the linear constraints define the *feasible* region. Geometrically this feasible region is a convex polyhedron. For a linear function, every local minimum is a global minimum since a linear function is a convex function. In addition, every maximum is a global maximum since a linear function is a concave function. For two reasons, an optimal solution may not exist. The problem can be *infeasible*, for example when constraints are inconsistent or conflicting. For example, the constraints  $x \leq 1$  and  $x \geq 10$  are conflicting and cannot be satisfied at the same time. As a second aspect, no optimal solution can be returned when the problem is unbounded, so if no maximum value for the objective function can be determined since no constraints exist limiting this value: geometrically, the polyhedron of the feasible region is unbounded in the direction of the gradient of the objective function with the gradient of the objective function being the vector of the coefficients in the objective function.

A linear programming problem can have distinct optimal solutions, all of them providing the same maximum or minimum value of the objective function. The optimal solution for a linear programming problem is always located on the boundaries according to the maximum principle for convex functions. Therefore, optimal solutions can only be found on the extreme point of the convex set of feasible solutions. If two extreme points are optimal solutions, all convex combinations of the solutions are optimal. [DD95]

The standard form of a linear programming problem consists of three parts:

- A linear *objective function*  $f(\vec{x})$  to be maximized or minimized
- A set of linear *problem constraints* with
- a set of non-negative variables  $\vec{x}$

A linear programming problem is often described in the matrix form and expressed as:

$$\max\{c^T \vec{x} \mid A\vec{x} \leq \vec{b} \wedge \vec{x} \geq 0\}$$

With  $\vec{x}$  being the variables to be determined,  $\vec{c}$  and  $\vec{b}$  are vectors and  $A$  a matrix of known coefficients.

If values in the set of non-negative variables  $\vec{x}$  are linear, the problem is a Linear Programming (LP) problem. If a subset of these variable are required to be integers (which is including binary values) the problem is called a Mixed Integer Linear Programming (MILP) problem and an Integer Linear Programming (ILP) problem when all variables of  $\vec{x}$  are required to be integer.

While an LP problem can be solved in polynomial time [Kar84], the restriction of variables makes the problem NP-hard. When the variables are restricted to binary values, the problem is one of Karp's 21 NP-complete problems [Kar72]. To solve LP problems, several well-established algorithms exist. Well-known methods to solve linear programming problems are *Basic-Exchange* algorithms, for example the Simplex algorithm proposed by Dantzig [Dan63], or *Interior Point* methods as Karmarkar's algorithm [Kar72].

MILP and ILP problems are solved by finding a solution to the corresponding LP problem (called the *relaxation of the ILP*) and then applying exact methods like *Branch and Bound* algorithms [LD60] or *Cutting-Plane* methods [NW88] to find the integer solutions.

### 5.5.2 Design of the Solver

With the MILP solver, we formulate the address selection and resource allocation as a linear optimization problem and apply mathematical optimization techniques to find an (mathematically proven) optimal solution. We therefore have to transform the problem described in 5.1.9 and formulated in 5.2 to a linear model usable in linear optimization. We have to formulate a linear objective function to maximize the objectives as described in 5.2.3 and to formulate a set of linear constraints to drive the objective function to the maximum and to restrict the solution to be feasible within the restrictions of the given problem domain.

As a result of solving the problem with an optimization process, we obtain for each address the decision if it is to be used to communicate with a remote peer (active) and the amount of resources assigned to this address. With our problem formulation, the vector  $\vec{x}$  contains information for each address  $a$  from the set of addresses available  $A$  about the inbound bandwidth  $b_{a,in}$  assigned, the amount of outbound bandwidth assigned  $b_{a,out}$  and an additional indicator  $n_a \in \{0, 1\}$  indicating if the address  $a$  was selected ( $n_a = 1$ ) or if it was not selected to be active ( $n_a = 0$ ). This indicator  $n_a$  being binary makes the problem an MILP problem and therefore NP-hard to solve.

To restrict the solution to be valid and feasible in the problem domain and to restrict the solution to obey the resource restrictions, we define a set of *feasibility constraints* used in the problem. These constraints ensure the feasibility of the MILP problem and that the solution is valid in the problem domain of address selection and resource allocation:

#### Communication Diversity

To provide resilient communication, we formulate this constraint to ensure a minimal number of connections  $n_{\min}$  with  $n_{\min} = \min(n_{\min}, |A|)$  to be established:

$$\sum_{a \in A} n_a \geq n_{\min}$$

#### Connection Usability

To ensure that established connections can provide a usable communication between peers, we define that each address selected to be active, has to get a minimum amount of resources outbound  $b_{out,\min}$  and inbound  $b_{in,\min}$  assigned:

$$\begin{aligned} \bigwedge_{a \in A} b_{a,out} &\geq n_a \cdot b_{out,\min} \\ \bigwedge_{a \in A} b_{a,in} &\geq n_a \cdot b_{in,\min} \end{aligned}$$

#### Address Limit

To enforce that only one address per peer is selected, we enforce for all addresses belonging to a peer  $a \in A_p$  and for all peers  $p \in P$ :

$$\bigwedge_{p \in P} \sum_{a \in A_p} n_a = 1$$

### Scope-Specific Resource Limitations

To ensure that in the different network scopes  $s \in S$  the resource limits for outbound bandwidth available  $b_{s,out}$  and inbound bandwidth available  $b_{s,in}$  are not exceeded and not more resources are consumed than available in the network scopes, we require:

$$h(s, a) := \begin{cases} 1 & a \text{ located in scope } s \\ 0 & \text{otherwise} \end{cases}$$

$$\bigwedge_{s \in S} \sum_{a \in A} h(s, a) \cdot b_{a,out} \leq b_{s,out}$$

$$\bigwedge_{s \in S} \sum_{a \in A} h(s, a) \cdot b_{a,in} \leq b_{s,in}$$

### Enforce a Finite Solution

To ensure feasibility of the problem and prevent an unbounded solution, we enforce the resources assigned to an address to reflect the resource limits of the scope  $s$   $b_{s,in}$  and  $b_{s,out}$  and in addition a (very large) virtual resource limit  $M$  to prevent numerical instabilities in the equation system with  $M_{s,in} = \min(b_{s,in}, M)$  and  $M_{s,out} = \min(b_{s,out}, M)$ :

$$\bigwedge_{a \in A} b_{a,out} \leq n_a \cdot M_{s,out}$$

$$\bigwedge_{a \in A} b_{a,in} \leq n_a \cdot M_{s,in}$$

To optimize the solution with respect to the objectives the address selection mechanism has to achieve according to Section 5.2.3, we define a set of *optimality constraints*, responsible to drive the objective function to the maximum to maximize the respective objectives:

### Resource Utilization

To improve communication between peers, more resources have to be assigned to the different connections and the resources available have to be optimally used. We therefore formulate an optimality constraint maximizing the resources used available on the system. The resulting utilization  $u$ , used in the objective function weighted by the coefficient  $U$ , is defined as:

$$\sum_{a \in A} b_a = u$$

$$\Leftrightarrow \sum_{a \in A} b_a - u = 0$$

### Resource Austerity

To provide good performance for the communication, resources have to be economically used and resources with low overhead have to be preferred over mechanisms with high overhead. So we add for all transport performance properties  $r \in R$  the following quality constraint and use the resulting quality  $q_p$  in the objective function. We therefore reduce the communication overhead by trying to use mechanisms with low overhead:

$$\bigwedge_{r \in R} \sum_{a \in A} r_a b_{a,out} = q_r$$

$$\Leftrightarrow \bigwedge_{r \in R} \sum_{a \in A} r_a b_{a,out} - q_r = 0$$

### Resource Relativity

The resources available on the system and in the different network scopes have to be distributed among the peers according to their importance for higher layer applications. Based on the preference values specified for their bandwidth requirements we distribute the available resources using the following *relativity constraint*, ensuring that more valuable peers get more bandwidth assigned than less valuable peers:

$$\begin{aligned} \bigwedge_{p \in P} \sum_{a \in A} f_p b_{a,out} &= r \\ \Leftrightarrow \bigwedge_{p \in P} \sum_{a \in A} f_p b_{a,out} - r &= 0 \end{aligned}$$

### Connection Diversity

To provide resilient communication for the peer-to-peer overlay, maintaining connections to a larger number of peers is an objective. We therefore define the objective to increase connection diversity  $d$  with:

$$\begin{aligned} \sum_{a \in A} n_a &= d \\ \Leftrightarrow \sum_{a \in A} n_a - d &= 0 \end{aligned}$$

#### 5.5.2.1 Objective Function

Based on the optimality constraints, we can formulate the resulting linear objective function to maximize for the optimization problem. The objectives we include are the objectives utilization  $u$ , relativity  $r$ , austerity  $a$  and diversity  $d$ . The different components included as objectives can be weighted using coefficients  $U, R, A, D$  in the objective function. The resulting objective function is defined as:

$$\max \quad u \cdot U + r \cdot R + a \cdot A + d \cdot D + \sum_{i \in P \times A} Q_i \cdot q_i$$

Based on this formulation, the solver can create a MILP problem from the address selection inputs and the system configuration and apply MILP solution techniques, as described in Section 5.5.1, to find an optimal solution. When the problem environment changes, the solver has to update the problem and resolve the problem. When new peers get requested or requests are canceled, new addresses become available or existing addresses become unavailable, the solver has to update the MILP problem and resolve the problem. Here we can differentiate between modifying and updating the problem: when information is added, the problem has to be updated: the *size of the problem* changes and rows and columns in the problem (referring to the standard matrix form of an MILP) have to be added or removed. When only transport performance properties are updated and therefore coefficients in the problem matrix  $A$  change, the solver can reuse the existing problem formulation and only update coefficients in the problem. This differentiation is important since the *Simplex* algorithm used to solve the integer-relaxation can reuse existing valid solutions from former solution runs when the problem size did not change. This *Simplex warm start* can lead to significant performance improvements as we show in the evaluation of this approach. The solution found by the MILP solver will be optimal within the objectives formulated or the solver will return that the problem is unfeasible, so no optimal solution exists. When the problem has multiple optimal solutions, it is (depending on the implementation) possible to iterate over the different optimal solutions and apply and additional mechanism to evaluate these solutions.

### 5.5.3 Discussion

With the MILP solver, we can provide a solution approach combining address selection and resource allocation in an integrated approach solving both aspects of the problem together. This will improve solution quality as shown in Section 5.9, since that solver can pay respect to both address properties and resources available to find an optimal solution and a trade-off between both aspects. With the MILP solver, we employ techniques from a well-covered field of research providing well-studied algorithms to solve such problems.

Based on the problem definition requiring a binary output to indicating if an address is active or inactive, the problem is a MILP problem NP-hard to solve. On the other hand can the solver performance benefit from the warm-start property of the Simplex solver. So the question arises if solving an expensive, NP-hard problem is a viable approach in the dynamic environment of a peer-to-peer system. Here the properties of the environment are a main factor impacting this question based on the number of and peers and the rate peers join and leave the network, addresses get available and unavailable and performance information and preferences are updated. This issue will be evaluated in Section 5.9.

## 5.6 The Reinforcement Learning Solver

As a third approach to solve the address selection and resource allocation problem, we propose a solver which benefits from the requirement to solve the problem repeatedly when the surrounding environment changes. The MILP solver, presented in the last section, also benefits from this fact but only by being able to reduce execution time reusing a solution found in a previous run. But the MILP solver is not able to improve the quality of the solution since it does not benchmark the solution and evaluate if the solution found matches the requirements of the applications and the transport underlay and adapt its parameters and objective to better match the requirements.

With the solver presented in this section, we envision a solution approach which can benefit from the fact of solving the problem repeatedly in an unknown, dynamic environment and being able to improve its allocation strategy by *learning* an allocation strategy fitting the environment and the requirements of the applications. In addition, we try to minimize *bandwidth excess*, caused by assigning peers more resources than applications can or do actually use when communicating with the peer. Here we try to adapt the allocation strategy by receiving feedback from the applications describing how satisfied the applications are with the current allocation and by paying respect to bandwidth excess, so the difference between bandwidth allocated to a peer and goodput achieved by the peer. Different to the greedy heuristic and the MILP solver, we allow the solver to *overutilize* the resources of the peer, so to allocate more resources than allowed to drive the link utilization to the amount of resources available. An additional benefit of *learning* a good strategy in an unknown and dynamic environment is that such an approach can overcome limitations imposed by defining the objectives for a solution in advance. Defining objectives a priori can lead to suboptimal solutions since these objectives may drive the solution in a direction not suitable for the current environment. So instead of defining constraints and objectives a priori, learning the properties of a solution online can improve the quality of service provided for the applications.

To realize this idea, the solver proposed in this section employs learning techniques from the field of machine learning. The RIL solver is based on the idea of *reinforcement learning* and has the goal to *learn* an optimal resource allocation strategy to distribute resources among the peers. The solver operates in an a-priori unknown environment and

tries to learn which *actions* lead to a good allocation. Learning in the context of address selection and resource allocation means to improve the strategy to allocate resources by evaluating how beneficial it is to perform a certain action in an unknown environment. Every action taken is evaluated based on their impact for the quality of the allocation and scored based on this impact to allow the solver to favor this action the next time he has to choose an action. This evaluation is accompanied with feedback given by the applications indicating how satisfied the applications are with the current allocation.

### 5.6.1 Machine Learning

The field of Machine Learning (ML) is a wide field of research, providing a rich set of approaches and techniques for various purposes and use cases. Machine learning focuses on enabling programs to solve problems without being explicitly programmed to do so. Here the fundamental approach is to enable a program to perform correctly on new, unseen tasks by generalizing over previous experience: with ML, programs learn how to treat new tasks based on their experience with previous tasks. Here we can find the difference between ML and data mining: with ML, a program tries to classify new inputs based on properties it previously learned, while data mining tries to find a-priori unknown properties in unknown data. ML techniques are used in a wide field of applications including spam classification, credit card fraud detection, computer vision, search engines etc.. The field of ML covers a wide field of techniques for different use cases. To structure the different approaches, ML techniques are usually classified in the following classes [Mar09]:

**Supervised Learning:** With supervised learning, the algorithm is *trained* with a set of training data and the expected output. Based on this training data, the algorithm can adapt to later automatically perform correctly on previously unseen data. The correct adaption of the algorithm is often ensured using a validation set with known input/output mappings, checking if data are mapped to the correct output. Supervised learning methods are often used in classification or clustering problems. Common approaches are artificial neuronal networks, support vector machines, Bayesian statistics, kernel estimators and decision trees.

**Unsupervised Learning:** With unsupervised learning, the algorithm is not adapted to the environment but it tries directly to detect similarities between inputs. Well-known approaches are k-nearest neighbor algorithms, Hidden Markov Models and self-organization of neural networks often used for clustering analysis.

**Reinforcement Learning:** is a compromise between supervised and unsupervised learning techniques. With reinforcement learning, the algorithm applies different solution strategies and can learn based on a feedback how beneficial the strategy performed was. So it is corrected if an output was incorrect but it is not told how to achieve the correct output. Prominent examples are dynamic programming, Monte Carlo methods and temporal difference learning.

**Evolutionary Learning:** adapts approaches from nature and biology by treating learning as an evolutionary process where organisms have to adapt to improve their survival rates. Here often a *fitness* function is used to score the quality of a given solution. Evolutionary learning is often used in search heuristics and stochastic optimization and well-known techniques are ant colony optimization and genetic algorithms.

All of these classes have the goal to learn from experiences in the past by generalizing over these experiences to predict how to find correct decision in the future. The interested reader can find an extensive introduction to ML in [Mar09] and [Alp04].

### 5.6.2 Design of the Solver

By using machine learning techniques to solve the address selection and resource allocation problem we want to learn an *optimal* allocation strategy by improving our strategy based on experience obtained from previous solutions and the feedback how satisfied applications are with our solution. This learning process takes places in an unknown environment we cannot a priori give a model for. So we do not know about the peers to communicate with, we do not know about addresses available and we do not have any information about the performance properties of the addresses, especially for extrinsic performance properties depending on the networking situation. In addition, we have no information about the applications and their requirements for communication for other peers. Based on this unknown environment and since this environment can dynamically change, it is not possible to formalize a model of the problem we can use to *train* a machine learning algorithm.

Due to this environment, supervised and unsupervised learning techniques are not suitable with the problem given here since they rely on valid training data (in the case of unsupervised learning) and cannot cope with dynamic environments and changing learning goals they have to adapt to. Evolutionary algorithms are not suitable since they only learn how to improve the solution for a single execution but do not learn how to predict the future by learning a generalized strategy. Evolutionary algorithms are often used as a *last resort* when no algorithm is known to find a solution to a problem. Since they often try to improve the solution with modifying and combining existing solutions (genetic algorithms) and heuristics try solve a problem as a combinatorial optimization problem, these algorithms are suitable for domains where solutions can be combined from other solutions. With the strict requirements for feasible solutions with Automatic Transport Selection and Resource Allocation (ATS), combining, modifying or a crossover of existing solution will often lead to invalid solutions as analyzed in [Oeh14] and [Sch13]. RIL approaches in contrary treat the problem as a control problem (which action to take) known from the field of control theory and do not require a model of the environment. RIL algorithms can learn a model of the environment and the allocation strategy on-line by performing actions and measuring the impact of the action.

We therefore relay on a reinforcement learning approach with the machine learning solver to find a suitable allocation strategy. Reinforcement learning has the objective to learn a strategy by testing different strategies which is similar to search over a *state space* of possible inputs and outputs in order to maximize a *reward*. RIL is generally described with an *agent* operating in an (unknown) environment trying to solve a problem by performing *actions*. It selects an action depending on the *state* it is currently in and receives a reward from the environment how good the current strategy is and updates its strategy based on this reward.

The RIL solver tries to learn an allocation for each peer the higher layer applications want to communicate with. Each peer included in the ATS problem is provided with a separate RIL agent learning a strategy for this particular peer. We define a set of actions each agent can perform in the environment: a peer can switch to a different address, the agent can increase the amount of bandwidth assigned to an address, it can decrease the amount of bandwidth assigned to an address and he can decide to do nothing. The

*state* an agent is currently in depends on the address he has selected and the amount of resources allocated to this address. Since the state space is based on the resources available, having a discrete state for every resource assignment is impossible due to the resulting dimensionality of the state space. Instead the RIL solver uses the concept of Radial Basis functions as described in [SB98]. Here *prototype features* are distributed over the state space and *activation* of this prototype state is calculated based on input address, inbound and outbound bandwidth provided for the state. The design of the solver relies on the basic concept of Temporal Difference (TD) learning algorithms allowing model-free learning: TD learning algorithms maintain an action-value function  $Q(s, a)$ , describing the value for each action  $a$  in the current state  $s$  and derive the policy by choosing the action with the highest value in every step. Our solver realizes two popular TD algorithms: SARSA and Q-learning. These algorithms update their action-value functions using an update rule based on the previous value estimation and the reward received from the environment for taking an action. SARSA updates its action-value function based on the reward it receives for the action taken in the previous step and the (discounted) return it can achieve in the current step taking the best action with the current strategy. Q-learning updates its action-value function based on the reward it receives for the action taken in the previous step and the (discounted) return it can achieve in the current step with taking the optimal future action, even if the action is not taken due to a exploratory step.

When a new agent is added to the solver, the agent starts to learn without any knowledge an allocation strategy by performing actions in regular intervals. When performing an action, the agent has to decide if it wants to learn by *exploring* the action-state space or to *exploit* the knowledge it acquired by following the current allocation strategy. Here the agent uses an *action-selection strategy* to find a trade off between exploration and exploitation. The solver supports two popular strategies, the  $\epsilon$ -greedy strategy performing an exploratory step with a probability of  $\epsilon$  and the softmax strategy, introducing probabilities for actions based on the current state to prevent to take actions known to lead to a bad reward. Softmax introduces a *temperature* value  $\tau$  leading to actions being selected and performing exploratory steps ignoring the best action available.

When exploiting the acquired strategy, the agent chooses the action with the highest value in the current state. When performing an exploratory step, the agent performs a random action. The agent updates his allocation strategy by evaluating how valuable the previous action was based on a reward it receives from the applications and the utilization of available resources. In our approach we allow agents to *overutilize* resources in a scope, so to assign more resources than available in the scope. If the agents over-utilize and more resources are consumed than available in a network scope they receive a penalty with the reward function. Based on the reward, the agents update the value of their actions in the state the action was taken using the TD update rules approach described above.

With multiple agents learning in a shared environment, the solver has to achieve good allocations for multiple peers at the same time. Therefore, coordination between the different agents is required to prevent the cannibalization of peers while others are provided with large amounts of resources. Here we apply the idea of social welfare algorithms to achieve a global fair allocation for all peers. The idea of welfare is combined with the reward function to give less reward to the agent when the solution is not balanced between agents. The solver supports both the Egalitarian Social Welfare model defining the utility of the current solution as the minimum utility from the utilities achieved for all peers and the Nash product, defining the utility of the solution as the product of utility achieved from all peers.

To allow faster learning when agents have not acquired knowledge yet, the solver provides adaptive stepping to perform actions. After an agent is initialized, the agent performs more actions to allow the agent to learn a strategy faster. This rate is decreased based on the amount of bandwidth utilized and available, with a minimum time interval between actions  $t_{\min}$  and a maximum time interval between actions  $t_{\max}$  and  $n$  an integer value:

$$t_{\text{adapted}} = (t_{\max} - t_{\min}) \cdot \left( \frac{b_{\text{utilized}}}{b_{\text{available}}} \right)^n + t_{\min}$$

To benefit from the allocation strategy learned over time and to allow the agent to learn faster in the beginning, the agent's action selection strategy is provided with a decaying exploration rate decreasing over time. With the decay factor, the exploration rate is decreased every time an exploratory step was taken.

An in-detail analysis of the suitability of different machine learning techniques and how different machine learning algorithms can be adapted to fit the address selection and resource allocation problem as well as description of the design and implementation details for the RIL solver can be found in [Oeh14].

### 5.6.3 Discussion

The RIL solver realizes the idea to learn a suitable learning strategy for the current environment. This is particularly useful for peer-to-peer systems, where no model for the environment can be given in advance. For a peer-to-peer system, this environment is unknown and in addition can dynamically change over time, a fact an adaptable learning approach can benefit from. When no model can be given in advance it is hard to define objectives a solver has to achieve in such an environment. This is a second aspect the RIL solver can attempt to learn dynamically. Due to its ability to overallocate, the performance of the peer-to-peer system can benefit by providing better performance for every single peer.

Learning in a dynamic environment is a complex task for a RIL agent. With RIL, the agent performs tasks to improve its strategy and therefore learning an allocation strategy requires time. This rises the question, if a RIL agent can adapt to its environment fast enough. In addition, the question has to be answered if the agent can adapt to a changing environment and requirements. This requires the agent first to realize this change and then to adapt by learning a current strategy. The speed of this adaptation is a critical issue for this type of solver.

## 5.7 Related Work and Comparison

In this section, we analyze and discuss how other decentralized and peer-to-peer applications supporting multiple transport mechanism or applications solve the problem of address selection and transport selection covered in this section.

### 5.7.1 Quality of Service in IP Networks

Since the Internet is a network consisting of a large number of independent, interconnected networks operated by different providers, no assertions about the provided quality of service can be made and the behavior of the network can only be modeled stochastically. On the Internet different approaches exist to ensure quality of service for applications. In general

we can distinguish between approaches trying to request and reserve resources through the network, so called Integrated Services (IntServ) approaches, and approaches prioritizing traffic classes by marking the traffic according to specific traffic classes and treating these traffic classes special, so called Differentiated Services (DiffServ) approaches already described in Section 2.6.1.

Integrated Services (IntServ) approaches try to reserve the resources required for a communication in advance through the network. To do so they use a parametrized approach, where applications can specify their requirements with respect to traffic properties. IntServ uses resource reservation protocols like RSVP [BZB<sup>+</sup>97] to reserve the required resources on all intermediate hops on the way to the traffic destination. On every intermediate hop a *soft state* has to be established to manage the reservation on the intermediary hop. When this state is not refreshed, it is automatically removed. IntServ requires all intermediary systems to collaborate, accept such reservations and treat the traffic according to the requirements. IntServ approaches are seen critical with respect to scalability in particular with respect to routers on the core of the Internet as these systems' main task is packet-switching at highest possible rates. These routers would have to accept and maintain a large number of reservations. Therefore, IntServ approaches are seen critical with respect to scalability and became less important with the growth of the Internet.

DiffServ approaches classify network traffic according to predefined traffic classes and treat these traffic classes differently using different queuing strategies to achieve the expected performance expectations. As described in Section 2.6.1 uses DiffServ the IP header to store traffic markings in IP traffic. With DiffServ approaches, all classification and prioritization is performed in so called *DiffServ Domains* representing a group of routers implementing the same DiffServ policies. Routers on a network may treat traffic according to the classification but are not required to do so and can ignore or overwrite the DiffServ markings. Network operators are often interested in controlling and managing the network traffic in their networks. Therefore, traffic classifications are often dismissed when traffic enters at ingress routers of a DiffServ domain. Therefore, DiffServ approaches can be used to prioritize certain traffic classes within a DiffServ domain but they do not provide end-to-end QoS between participants over the Internet.

### 5.7.2 The SpoVNet Project

The Spontaneous Virtual Network (SpoVNet) project<sup>1</sup> is a research project with the goal to allow decentralized applications to communicate over a heterogeneous communication infrastructure using self-organizing virtual overlay networks of decentralized systems. SpoVNet is a framework with the aim to enable developers to implement a diverse set of applications on top of a common foundation. SpovNet tries to overcome heterogeneity in networks and network protocols as well as limitations (like NAT and firewalls) with a self-organizing, decentralized transport infrastructure, virtual addressing of nodes and a DHT-based control overlay.

SpovNet uses the Ariba underlay abstraction<sup>2</sup>, described in [BHMW08a], as a transport infrastructure. Ariba provides communication over heterogeneous communication links using an homogeneous interface for the different supported communication mechanisms and copes with mobility, heterogeneity, and middleboxes restricting end-to-end connectivity. To provide this homogeneous interface, Ariba provides an abstraction layer with virtual links for the various communication mechanisms. As underlay communica-

---

<sup>1</sup> <https://www.spovnet.net>

<sup>2</sup> <http://www.ariba-underlay.org>

tion protocols Ariba supports multiple transport protocols including TCP, UDP, SCTP, and IPv4, IPv6 and Bluetooth RFCOMM as communication mechanisms. Ariba supports NAT relaying to connect hosts in restricted environments and protocol relays to overcome heterogeneity in networks.

The Ariba documentation lacks documentation how protocols and addresses are managed. According to the source code, Ariba employs a First-Come-First-Serve (FCFS) approach to select from the available addresses. Ariba stores all available communication addresses in a list. When having to communicate with a communication partner, it iterates this list and then selects the first address marked as available. In this decision no additional aspects besides the availability are paid to respect to. To the best of our knowledge, Ariba does not provide resources restrictions to limit resource consumption. Ariba supports higher layer applications and provides communication for these applications on a best-effort basis relying on FCFS address selection.

### 5.7.3 The Tor Project

The Tor project<sup>3</sup> provides anonymous communication for users on the Internet. Tor, already described with the transport infrastructure in Section 4.5.1, uses *onion routing* to make it hard for an observer to match source and destination of network traffic as it traverses the Internet. Tor uses so called *Bridge relays* [Din07], relays not listed in the public directories, to make it harder for service providers to restrict access to the Tor overlay.

To make Tor more resilient against filtering and censorship on the IP level, Tor provides *pluggable transports*. These *pluggable transports* obfuscate or morph the traffic between the client and the Tor node to make it less susceptible to censorship based on techniques like deep packet inspection. Prominent examples for pluggable transports are Scramblesuit, Stegotaurus and FTE [DCRS13] for obfuscation, SkypeMorph [MMLDG12] for traffic morphing, and Flashproxy [FHE<sup>+</sup>12].

If multiple pluggable transports are available, the Tor software also simply uses the first pluggable transport configured that works. With respect to clients, Tor realizes a best-effort service.

### 5.7.4 The Invisible Internet Project

The Invisible Internet Project (I2P)<sup>4</sup>, described in [ZH11], is a network to provide anonymous communication over the Internet. In contrast to Tor, I2P does not primarily try to provide anonymous access to services on the Web, but instead creates an overlay network where participants can anonymously exchange messages using a variety of different services (like email, blogs or file sharing) that are operated within the I2P overlay.

I2P's transport service provides message-based communication between I2P routers using multiple transports. At the moment two transports are supported: a TCP-oriented transport called NTCP, and secure semi-reliable UDP (SSU). I2P also supports NAT and firewall traversal techniques to improve connectivity for devices in restricted networks.

I2P provides a transport selection mechanism based on so called *bids*: For each message to be send, I2P evaluates which transport mechanism provides the *best* bid and chooses this mechanism to send this message. This decision is using a heuristic that considers multiple aspects like configured preferences, if an connection is already established,

---

<sup>3</sup> <https://torproject.org>

<sup>4</sup> <https://geti2p.net>

if a previous attempt to send failed as well as additional information about the state and the communication partner. To the best of our knowledge provides I2P a best-effort to client applications relying on its address selection mechanism. Bandwidth limitations are only used to enforce the maximum data rate for communication.

## **5.8 Implementation**

In this section, the implementation of the automatic transport selection and resource allocation mechanism is described based on the problem setting evaluated in Section 5.1 and the design elaborated in Section 5.2. We will present the implementation of the ATS service and present implementations of the solver mechanisms described in the previous sections. To support the different solution mechanisms proposed in this chapter, the component employs the functionality to load the solver as loadable plugins and to interact with them using a simple API. This section covers implementation details to allow the reader to understand how ATS is integrated in the GNUet peer-to-peer framework and the following evaluation of the proposed design.

### **5.8.1 The ATS Service**

The main component implementing the functionality for ATS in GNUet is the ATS service. In GNUet functionality is typically split in separate components implemented as services or daemons, as described in Section 2.8.3 and also described in the previous chapter covering GNUet's low level transport infrastructure in Chapter 4. Each GNUet service is implemented as an operating system process and is accessed by other GNUet components using an API and communication based on IPC.

The main task of the ATS service, as described in Chapter 5 covering the ATS design, is to provide address suggestions to GNUet's transport infrastructure and to allocate resources to these addresses based on the requirements of higher layer applications specify to communication with peers. The decision about the set of addresses and resource assignments has to be found based on the requirements of higher layer applications, current performance properties of the addresses available to communicate provided by the transport infrastructure and resource constraints set by the user. Based on these requirements, we see that the ATS service is interacting with at least three different components to acquire the information required to make its decision. These collaboration is depicted in Figure 5.2.

The ATS service needs to collaborate with the TRANSPORT service, responsible to maintain low level connectivity with other peers as described in Chapter 4, both to obtain information about the set of addresses it can choose from as well as to present its suggestions about the addresses to use and the resources to transport service.

On the other hand must high layer applications notify ATS about the requirements they have for communication with remote peers. Based on these requirements the ATS service can make an educated decision for address selection and resource allocation to meet the applications' requirements.

As a third aspect has ATS to obtain the user constraints on resources as described in Section 5.1.5. The user can restrict the amount of resources to be used by GNUet to a specific amount. These restrictions are stored within the configuration and have therefore to be obtained from the configuration to be used with the address selection process.

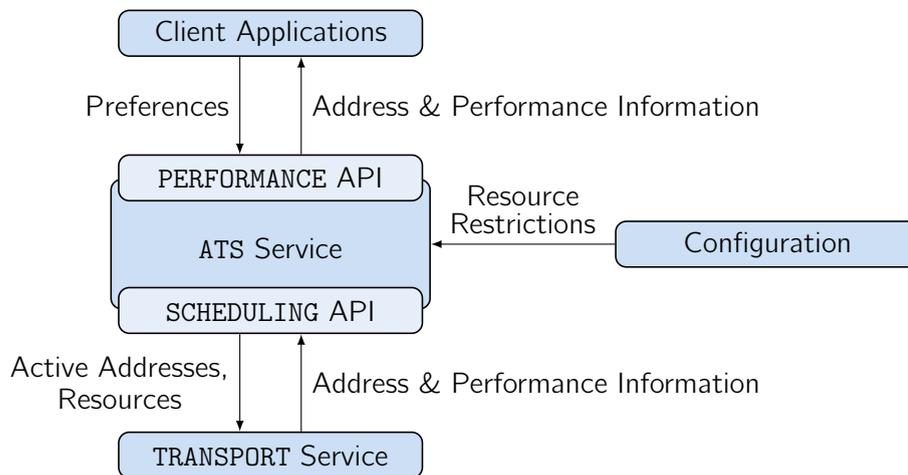


Fig. 5.2: ATS Interaction with Applications and Transport Infrastructure

### 5.8.2 ATS Information

To allow to exchange information between ATS and external components like TRANSPORT service, ATS specifies a format to exchange these information and set information entities to be used in the exchange.

The generic information format used defined with the ATS service and defines a data structure `GNUNET_ATS_Information` to exchange this information and a fix but extensible set of ATS information properties in the enumeration `GNUNET_ATS_Property`. Components interacting with ATS use this information struct (and arrays of this structure) to pass information to ATS. In each ATS information, elements specify the type of the property and the value of the property in an 4 byte unsigned value.

To exchange information about network scopes and refer to these scopes, ATS defines an extensible set of network scopes to be used in ATS and interacting components. These network scopes are used to define resource restrictions for network scopes and to define in which scope addresses are used. ATS currently defines the following network scopes:

**UNSPECIFIED:** used e.g. when no scope information is available for an address (yet)

**LOOPBACK:** used for loopback interfaces

**LAN:** address scope for addresses in the local network

**WAN:** address scope for addresses not in the local network

**WLAN:** network scope for addresses of the WLAN transport plugin

**BT:** network scope for addresses of the Bluetooth transport plugin

ATS information structs are also used to exchange information about network scopes for addresses by setting in the struct the appropriate type `GNUNET_ATS_NETWORK_TYPE` and the value in the struct to one of the network types described above.

Information about performance properties of transport mechanisms and addresses are exchanged between components by setting the ATS information type to the appropriate property type (e.g. `GNUNET_ATS_QUALITY_NET_DELAY` for network delay) and setting the value to the delay value. The set of performance properties is defined by ATS.

Applications can use the preference types predefined by ATS and use this preference types when specifying their preferences about communication with remote peers with the ATS service.

### 5.8.3 Interacting with ATS

To enable other components to interact with the ATS service to provide and obtain information from the service, the ATS service provides several API interfaces for interaction, depending on the respective purpose.

#### 5.8.3.1 The scheduling API

For interaction between the ATS service and the TRANSPORT infrastructure, ATS provides the SCHEDULING API. With the SCHEDULING API, the TRANSPORT service can initiate a connection to the ATS service to provide the ATS service with information about addresses available, notify the services about changes in the environment and issue address suggestion requests and cancel these requests. To interact, ATS's SCHEDULING API provides the following functionalities:

##### Connect to ATS service

**Add new addresses:** When TRANSPORT's validation tested new addresses or a peer has connected and new addresses are available to communicate with a remote peer, TRANSPORT notifies ATS about the availability of the new address. Together with the address, TRANSPORT can pass session information if a transport session is existing and ATS performance information.

**Update an address:** When properties of an address change, TRANSPORT updates the information ATS knows about an address. This can be a transport session, which is now established with this address, or updated transport performance information as described in Section 5.8.2.

**Specify that an address is currently used to communicate with a peer:** To achieve connection stability, it is useful for ATS to know if TRANSPORT is actually using a suggested address so it can stick to this address. The API provides the possibility to notify ATS, that an address is actively used or not used any longer.

**Delete addresses:** When an address is not valid any more, TRANSPORT notifies ATS and ATS deletes this address and to exclude it from further solution iterations.

**Request address suggestions:** When TRANSPORT is instructed to establish an overlay connections with a remote peer, it requests the address to use from ATS. ATS will respond with the best address available and notify TRANSPORT about changes to this decision as long as TRANSPORT does not cancel the request.

**Cancel address suggestions:** When TRANSPORT is not longer requiring address information for an overlay peer, it cancels the suggestion request and ATS will stop to suggest addresses for this peer.

##### Disconnect from ATS service

Clients issuing address suggestion requests to the ATS service are provided with information about active addresses and assigned resources for the peers they requested. Whenever the selected address or the amount of resources assigned to an address changes, ATS notifies the clients about these changes until they cancel the request. The TRANSPORT service uses this information to switch to the address now selected to be active, updates the resource restrictions or disconnect a connection with a peer if ATS cannot provide an address to communicate with a peer.

### 5.8.3.2 The performance API

To allow higher layer applications to interact with ATS service, the ATS service provides the PERFORMANCE API. With this API, applications can interact with the service to obtain information about the address currently used and to specify their preferences for peers and transport performance properties. It provides functionality to:

**Connect to ATS service:** With the connect call, the application can register a callback to be notified about changes for addresses and address properties.

**List addresses:** Returns a list of all available or only active address for a specific peer or all peers.

**Notify ATS about preferences:** With this function applications can specify their preferences for peers and transport properties.

#### Disconnect from ATS service

The PERFORMANCE API allows applications to specify their requirements for the communication with remote peers. With the PERFORMANCE API, applications can specify which transport performance property or properties are particularly important for them. An application can specify for each peer which transport performance property is important for it. To specify this importance, a float value is specified coming from the domain of the issuing application. This application-specific value is then normalized and correlated inside of the ATS service for further use as described in Section 5.3.1.

In addition can applications use this API to obtain information about the addresses currently selected and to obtain performance statistics about addresses and monitor the address selection process continuously. When connecting to the ATS service they can specify a callback function to be called whenever addresses or address properties change. In addition to monitor continuously, the application can request the list of currently active addresses for all peers or the address active for a specific peer.

### 5.8.4 Peer and Address Management

To manage information about peers, addresses, transport sessions and address properties provided by the transport underlay and application preferences provided from applications, the ATS service provides an ADDRESS component to manage this information. When applications or the transport infrastructure pass information to the ATS service, the ATS service gives this information to the ADDRESS component.

Information in the ADDRESS component is manipulated using the SCHEDULING API and the functions provided by the API. The ADDRESS component provides functions to add and delete addresses, add and remove transport sessions, update transport performance information, information if address are used by TRANSPORT and to add and remove address suggestion requests. The ADDRESS component passes updated information to the

PERFORMANCE clients and forwards this information to the solvers responsible to find a solution to the address selection and resource allocation problem described in Sections 5.8.8. The solvers notify the ADDRESS component about changes to selected addresses and resources assigned to addresses and the ADDRESS component forwards this information to the PERFORMANCE clients and the transport infrastructure. To prevent frequent notifications and to improve bandwidth stability, TRANSPORT is only notified, when the delta in a bandwidth assignment has a *reasonable* size. With respect to bandwidth this requires the delta to be larger than the minimum amount of bandwidth  $b_{min}$  required for useful communication.

The ADDRESS component is responsible to load the ATS solver plugin, as described in Section 5.8.8, and to forward information to the solvers. During initialization, the ADDRESS component loads the network quotas from the configuration and passes this information to the solver. For the interaction with the normalization component described in the next sections, the ADDRESS component provides a callback function called by the normalization component. When a normalized property or a preference is updated and the respective update called, the ADDRESS component notifies the solver about the update in this callback. The ADDRESS component provides a callback to allow the solver to notify it about updates to selected addresses and to updates in the resource assignment. When this callback is called, the ADDRESS component forwards this information to both the PERFORMANCE API clients and the transport infrastructure.

### 5.8.5 Management of Transport Performance Properties

To use transport performance properties when solving the address selection problem, address properties have to be normalized between the different properties as described in Section 5.3.2. The ATS service provides a NORMALIZATION component to perform this normalization and to make the normalized values available to components depending on this information. NORMALIZATION is integrated with the ADDRESS component: when the ADDRESS component receives performance properties from the transport infrastructure, it forwards this information to the NORMALIZATION component.

The normalization component performs an input normalization for all transport performance properties based on the formalization given in Section 5.3.2: for each property it analyzes the minimum and maximum value and normalizes performance properties of this type. As a result, the NORMALIZATION component can provide for all properties normalized relative transport performance properties  $r$  with  $r \in [1, 2]$ . If a performance property is updated and the computed relative value changes, the NORMALIZATION component notifies the ADDRESS component to forward this updated value to the solvers, which can recompute their allocations depending on this update.

The NORMALIZATION component stores the normalized values for all peers and addresses internally as it needs this information for its computation and to provide it to the solvers requiring these values for their computations when solving the address selection problem.

### 5.8.6 Management of Application Preferences

Since application preferences have to be normalized for an application and correlated between applications, application preferences are managed by the NORMALIZATION component, similar to the approach for transport performance properties described in the previous section. When applications notify ATS about their preferences for a peer and a transport property, this value is forwarded from the ADDRESS component the normalization

component. `NORMALIZATION` computes the relative preference value as described in Section 5.3.1. When this value is different from the previous value, the `ADDRESS` component is notified to notify depending components about this update. As for the performance properties, the `NORMALIZATION` component internally stores the computed relative values and other components can request these values.

### 5.8.7 The Solver API

For the communication between ATS service and plugin, the service provides a programming interface to allow the service to interact with the plugin, as well as to allow the solver to initiate interaction with the service. The main focus of this interaction is to allow the solver to notify the service about the results of the address selection and allocation process and the service to notify the solver plugin about changes in the current environment. The solver API therefore provides functionality for the following situations:

#### 5.8.7.1 Service to Solver Interaction

**Solver startup:** When the ATS services loads the solver plugin, it initializes the solver using the initialization function provided by the solver plugin. As an argument the service passes an environment handle containing information about the callback functions the solver can call with the service as well as additional information like configuration information and network quotas required to initialize the solver.

**Add a new address:** When information about a new address is provided by the underlay, the service notifies the solver to add this address to the current problem. As additional argument, information about the network scope and current transport properties are provided to the solver with this function.

**Update session:** When a new session is available for an address, the service can notify the solver to include this session together with the address in the problem.

**Update in use:** When an address is actively used by the transport underlay this fact is reported to the solver to allow the solver to handle established and used connections differently from addresses requiring a connection to be established. This function is also used to indicate that an address is not actively used anymore by the underlay.

**Update address properties:** The services notifies the solver whenever the performance properties of an address change and the solver has to reconsider its current selection and allocation.

**Network switch:** When an address switches from one network scope to a new network scope, the solver has to update its allocation. The solver can be notified using this function that an addresses switches from one network to another. This is in particular important when the network scope for an address cannot be determined when the address is added. In this case, the address is added to the `UNSPECIFIED` scope and when the scope is available, it is moved to the respective network scope.

**Delete an address:** When an address is not usable any more due to a disconnect etc., the service notifies the solver to remove this address from the current problem. If a session is specified, first the session is deleted, but the address itself is kept, to allow to establish a new outbound connection using this address.

**Start address suggestions:** When the transport underlay wants to communicate with a peer, it requests address suggestions for this peer from the ATS service. As long as this suggestion request is active, ATS will include this peer in the problem and suggest addresses for this peer to the transport underlay until this request is canceled.

**Stop address suggestions:** When address suggestions are no longer required for a peer since communication is not desired anymore, the service notifies the solver to not include this peer in the problem anymore and to stop suggesting addresses.

**Start a bulk operation:** Since the service may want to issue a larger number of updates at the same time and wants to prevent the solver from recalculating the solution for every single update, the service can indicate the begin of a *bulk operation* to the solver. This allows the solver to wait for the end of this operation before finding a solution for the updated problem.

**Stop a bulk operation:** This function indicates the end of a bulk operation and indicates to the solver that it now can start to find a solution for the updated problem.

**Application preferences:** To adapt address selection and resource allocation to correspond with application preferences, the applications specify their requirements for peers with respect to the different transport properties. With this function, the service can notify the solver about changing application preferences for a peer.

**Application feedback:** Besides specifying their requirements, the applications can also specify how satisfied they are with the current communication to a peer. Since applications are not required to give this information and in addition not in a fix time interval, applications can additionally specify for which time span in the past this feedback is designated.

**Solver shutdown:** During shutdown, the ATS service calls the shutdown method to unload the solver plugin. As a consequence, the solver stops all operations and frees all data.

### 5.8.7.2 Solver to Service Interaction

To allow the solver plugin to notify the service, several functions are provided by the ATS service to allow communication from the solver to the service. This includes functions to:

**Notify about allocation changes:** When the address selected for a peer or the resources assigned to this address change, the solver can use this function to notify the service about this change and the updated resource allocation.

**Indicate solver operations:** The solver can notify the service about the state of a problem solution cycle using this notification callback. The solver can indicate the begin of an solution cycle, the end of a solution cycle including more detailed information about the result of the cycle. It also provides solver-specific information, for example the begin or end of solving the LP relaxation of the MILP problem for the MILP solver described in Section 5.5.

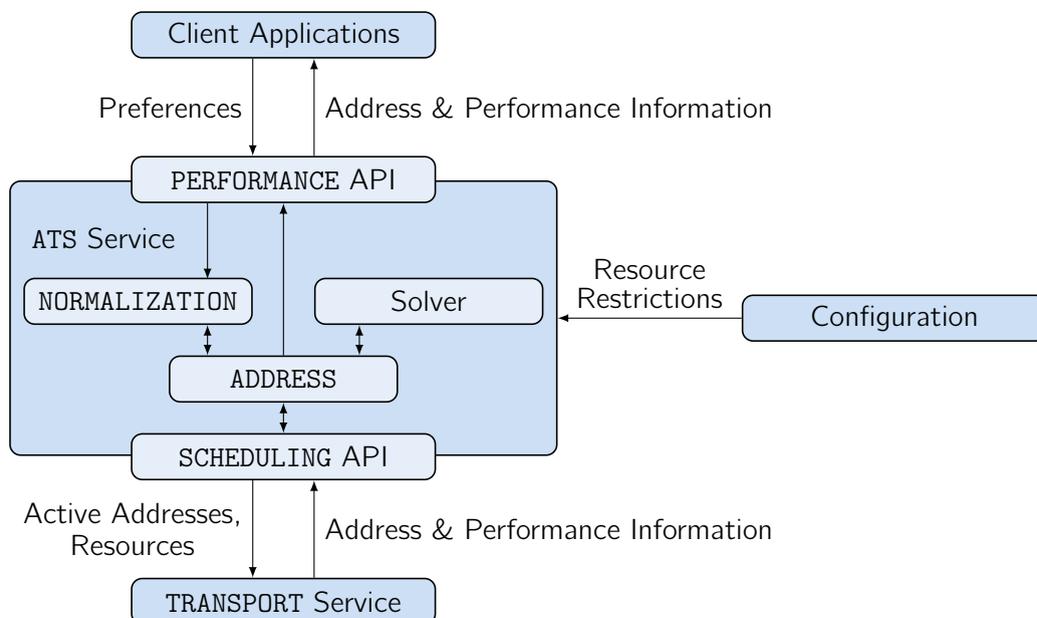
**Get properties for an address:** The solver can request the current transport properties for an address from the ATS service's NORMALIZATION component described in Section 5.8.6.

**Get preferences for a peer:** The solver can request the current application preferences for a peer from the ATS service's NORMALIZATION component described in Section 5.8.5.

### 5.8.8 ATS Solvers

To support the different solution approaches we propose and to allow to easily exchange the different solution approaches, compare them with each other and evaluate them under a realistic setting, we provide an extensible solver plugin architecture with the ATS service. This architecture supports to load the different solver implementations as plugins and provides a uniform communication interface for the interaction of ATS service and the solver plugin. In addition allows this architecture to instrument the solver plugins directly using an instrumentation driver to evaluate the different solvers directly.

The resulting architecture for the ATS service including its internal components and the interaction with applications and the transport infrastructure is depicted in Figure 5.3.



**Fig. 5.3:** ATS Service with Internal Structure and Interaction

### 5.8.9 The Greedy Heuristic Solver

The implementation of the greedy heuristic is (as for all solution approaches described in the following sections) realized in a solver plugin, which is loaded by the address component interacting with the address component using the solver API. During initialization, the ADDRESS component passes the current resource constraints to the solver and the solver creates a set of network scope structs representing the different network scopes available. For each network scope, the solver stores information about the resources available in this scope inbound and outbound, the number of available addresses and the number of *active* addresses in the scope. In addition, it loads the configuration settings from the configuration relevant for the solver. The list of configuration settings influencing the greedy heuristic solver is given in Table 5.1.

The solver is notified about changes in the environment by the ADDRESS component and whenever the environment changes the solver has to update and recalculate its allocation. When finding this allocation, the solver only includes information for peers actually requested and performs an allocation update only when information for requested peers change.

When a new address is added to the problem, the solver adds the address to the respective network scope the address is located in and updates the information stored with the network scope. If this address belongs to a peer with a pending address request, the solver updates its allocation to ensure this address is included in the allocation.

When a new request for address suggestions is received for a peer, the solver first finds the *best* address by comparing the performance properties for the peer's addresses. The algorithm ensures connection stability by not switching away from a connection just established. Here the solver uses a stability factor  $f_s$  obtained from the configuration and keeps the address if it was activated less than  $1 \cdot f_s$  seconds ago. The solver checks for every address if it is possible to activate this address. Here the solver checks if with this address activated all active addresses in the respective network scope can get at least the minimum bandwidth  $b_{min}$  assigned.

If it is possible to activate an address, the solver marks this address as active, updates the respective network scope and redistributes the resources in the network scope as described in Section 5.4.1.2 using the proportionality factor  $p$  obtained from the configuration. When no address is available or can be activated due to bandwidth restrictions, the solver returns and does not notify the ADDRESS component. When the resources assigned to the active addresses in this network scope change, the solver notifies the ADDRESS component using the respective callback for each address and notifies ADDRESS about the address and the resources assigned.

When an address or transport session is deleted, the solver removes it from the network scope it belongs to and redistributes the resources to the active addresses left in this scope. When the address or session was marked as active and address suggestions are requested for the peer this address belongs to, the solver tries to find an alternative address to use using the address selection algorithm. If an address is found, it activates the address and redistributes the resources in the respective scope. If no alternative address can be suggested, the solver notifies the ADDRESS component to disconnect the respective peer since no address is available by calling the ADDRESS component callback for the deleted address with no resources assigned (both inbound and outbound bandwidth are 0).

When an active address switches network scopes, the solver checks if it is possible to activate the address in the new scope and if possible, moves the address and redistributes resources in both the previous and the new scope. If it is not possible, the solver tries to find an alternative address to activate. If no alternative address is available the solver disconnects the peer. Addresses switching network scopes can for example occur when initially no scope information was provided and the address was added to the UNSPECIFIED scope. When the address information is updated with the correct network scope, the address has to be moved to the respective scope.

When transport performance properties are updated, the solver first checks if based on the updated properties the address is still the best address. If the address is still designated as best address, the solver redistributes resources in the address' network scope. If another addresses was designated best, the solver switches addresses, deactivates the old address, activates the new address and redistributes resources in both network scopes and notifies ADDRESS about the updates.

When application preferences change, the solver checks if based on the updated pref-

erence, the current active address for the peer is still the best decision and otherwise switches to the selected address. After this check the resources in the network scope(s) are re-assigned.

The prevent multiple resolution processes, e.g. when relative properties strongly intertwined with each other change, the solver supports to lock the solver from resolving the problem using bulk operations. The ADDRESS component can start a bulk operation and lock a semaphore and the solver will not re-solve the problem until the last bulk operation lock was removed. When no bulk lock is pending, the solver re-solves the problem and notifies the ADDRESS component about the updated solution.

When the solver is shutdown, it cleans up all data and data structures allocated by iterating over all network scopes. After the clean up the solver returns to the ADDRESS component and shutdown is completed.

**Tab. 5.1:** Configuration Values for the Heuristic Solver

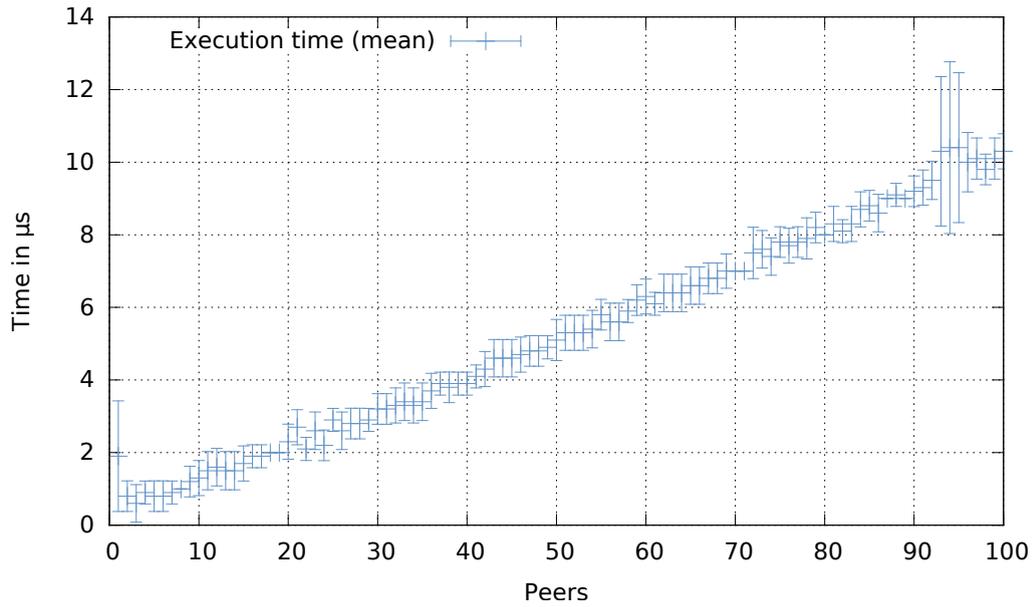
Name:	PROP_PROPORTIONALITY_FACTOR	Type:	float
Default:	2.0	Range:	$\geq 1.0$
The proportionality factor represents a trade-off between preference compliance and a fair bandwidth distribution when distributing resources within a network scope as described in Section 5.8.9. For values close to 1.0, the resources are distributed fairly, for larger values, the distribution follows closer to preferences and penalizes peers without preferences from applications			
Name:	PROP_STABILITY_FACTOR	Type:	float
Default:	1.25	Range:	[1.0 . . . 2.0]
To prefer addresses currently in use and prevent high frequent address switching, this factor is used to control how stable the address selection process should stick to an address.			

### 5.8.9.1 Performance Evaluation

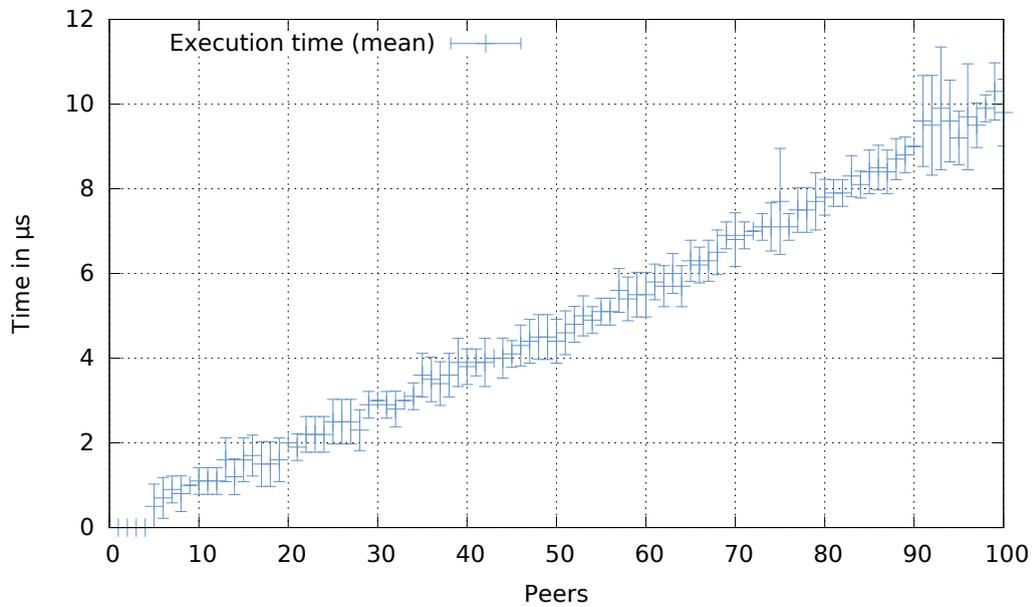
The scalability and performance of the proposed greedy heuristics was benchmarked using a dedicated benchmarking tool, sequentially increasing the problem size the solver has to solve and measuring execution times both for problems where the problem size changed as well as for updated problems, were only a certain percentage of the addresses in the problem was updated and the solution was recalculated. The results for the full execution with a changed problem size are depicted in Figure 5.4(a) and for the updated problem with 10% of all addresses in the problem updated in Figure 5.4(b). As we can see from both figures increases the execution time linearly with the number of peers and addresses in the problem for both the full execution and the updated execution run.

### 5.8.10 The Mixed Integer Linear Programming Solver

With the MILP solver we provide a solver for the address selection problem combining address selection and resource allocation in an integrated approach and providing a mathematically optimal solution. With this solver we formulate the problem as a MILP problem and use mathematical optimization algorithms to obtain the optimal solution and forward this solution to the ADDRESS component. The main focus of the MILP solver is to maintain a MILP formulation of the current environment setting, to update or recreate the



(a) Execution Time of the Heuristic Solver for a Full Solution



(b) Execution Time of the Heuristic Solver for an Incremental Solution

**Fig. 5.4:** Performance of the Heuristic Solver

problem when the environment changes and to find a solution for the MILP problem and notify ADDRESS about the change. To solve the MILP problem provided by the MILP solver, the solver uses the GNU Linear Programming Kit library<sup>5</sup>, a well-established free software library with a focus on solving large-scale LP and MILP problems.

The GNU Linear Programming Kit (GLPK) provides extensive functionality to solve both LP and MILP problems. To solve LP problems, GLPK supports both the Simplex algorithm as well as the interior points method described in 5.5.1. To solve MILP problems, GLPK provides a branch-and-cut solver as described in 5.5.1. This branch-and-cut solver (`intopt` solver) requires a valid solution of the LP-relaxation of the problem to find the MILP optimal solution. This solution can be obtained from a previous run of the Simplex LP solver or using the `intopt` pre-solver. A limitation of the `intopt` solver is, that it cannot use a solution provided by the interior point solver and therefore the interior point solver cannot be used in our approach. A second limitation of the `intopt` solver is, that if multiple optimal solutions exist, it is not possible to obtain all optimal solutions since the solver can only provide one optimal solution. To prevent numerical instabilities, the Simplex LP solver is equipped with an optimization functionality to automatically scale the problem. GLPK supports the LP warm start, as described in 5.5.1, so when the problem was only updated (coefficients changed) but the problem size did not change (no rows or columns added or removed), an existing solution from a previous run can be reused without having to solve the full LP. [Tea14]

In GLPK, optimization problems are represented using a GLPK specific problem structure. The problem is manipulated using GLPK functions, amongst others to set the optimization direction (minimize or maximize), add constraints (rows) and structural variables (columns) to the problem and define the coefficients of the objective function. The coefficients of the problem matrix  $A$  as described in 5.5, are stored in three arrays with two integer arrays defining the row and column this coefficient is located in the coefficient matrix and a third double array containing the coefficient itself. Information about GLPK specific implementation details can be found in [Tea14]. When the GLPK problem is configured with the objective function, columns and rows, these arrays representing the coefficient matrix are with GLPK. After the solution process, the result can be analyzed, so if an optimal solution was found or if the problem was unfeasible or the solution process was interrupted. If an optimal solution was found, the solution can be analyzed for the value of the objective function, value of structural variables (columns) and values of auxiliary variables (rows).

During initialization, the MILP solver obtains the resource restrictions for the different network scopes, loads the configuration parameters specified in Table 5.2 and initializes the GLPK environment. The solver only runs the optimization process on external events, so when the environment changes. When the MILP solver is notified about changes to the environment, the solver has to update the GLPK problem to reflect the change. Here the solver has to distinguish between updates changing the problem size and updates only updating the problem to be able to benefit from the speed-up of the Simplex warm start. Updates changing the problem size are new addresses and sessions added to the problem or address suggestion requests added or removed from the problem. In this case the solver has to recreate the MILP problem and include the update in the problem and then solve the problem using the GLPK solver. Changes only updating the current problem are updated transport performance properties and updated application preferences. In this case the solver updates coefficient arrays described in the previous paragraph and uses the `intopt` solver to find the optimal solution.

<sup>5</sup> <https://www.gnu.org/software/glpk/>

When creating the MILP problem, the solver first initializes the GLPK problem and sets the optimization direction to maximize. In the next step, the solver initializes the objective function with the coefficients  $D, U, R, A$  and  $Q_i$  to weight the different optimization objectives as described in Section 5.5. In the next step, the solver adds the structural variables (the columns) to the problem. It adds one continuous value column with bounds  $\geq 0$  for each objective in the objective function as defined in Section 5.5 to allow to use the optimality constraints to drive the objective functions to the maximum. In addition, the solver adds for each address to be included in the problem columns (structural variables) representing the bandwidth assigned and a binary column  $n$  indicating if the address is selected to be active or not. In the next step the solver iterates over all peers and addresses to be included in the problem and adds the feasibility and optimality constraints described with the design to the problem and sets the respective coefficients and their position in the constraint matrix  $A$  in the GLPK coefficient arrays. When done, the solver loads the new coefficient matrix into the problem. For each address and peer included in the current problem, the solver stores information about the constraints (rows) and structural variables (columns) related to the peer or address. This is required to later update the information related to a peer or address when the problem is updated and to obtain the resulting resource assignment and activation information for each address after solving the problem. After loading the matrix, the solver can automatically scale the problem using GLPK's problem scaling functionality to prevent numerical instabilities in the optimization problem.

When the problem has to be updated, the solver can directly update the problem with the new values based on the information about constraints and resources related with each address stored while creating the problem. When the solver is notified about an update for an address, it takes the updated value and sets the new coefficients in the problem matrix to represent the update. For the update, the solver retrieves the respective constraint row to update from the GLPK problem, locates the column to update, updates the respective coefficient and stores the updated row in the GLPK problem.

When the solver has to find a solution for the optimization problem due to a change in the problem size, the solver uses GLPK's LP Simplex solver to find a solution for the LP-relaxation of the problem which can be used by the `intopt` solver to find the solution for the MILP problem. The execution time of the Simplex solver can be controlled with respect to the number of maximum iterations and the maximum execution time to interrupt the optimization process. After solving the LP problem, the solver uses GLPK's branch-and-cut integer optimizer to find a integer optimal solution based on the LP relaxation found before. Since finding an optimal solution with the branch-and-cut solver is expensive and takes a long time but the optimizer is often able to find a solution very close to the optimal solution very fast, we added support for tolerating a solution slightly worse than the LP optimal solution and terminate the optimization process faster. Our solver can stop finding the optimal MILP solution when the branch-and-cut solver found a solution within a (configurable) *tolerance gap*  $g_{milp}$ . By default this gap is set in our solver to accept solution within a tolerance of 2.5% from the (LP) optimal solution. When a solution for the MILP problem is found, the solver analyzes the solution found and notifies the ADDRESS component about the results of the optimization process. The solver iterates over all addresses included in the current problem and extracts the information about the resources the `intopt` solver assigned to the address and if the address was selected to be active. This information can be easily extracted based on the information about the structural variables containing this information stored with each address when the problem was created. When the selected addresses or resources assigned to an address change,

the solver notifies the ADDRESS component about the updates. The MILP solver supports bulk operations to prevent the solver from frequently solving the problem when a large number of updates are performed. The solver implements these blocks using a semaphore and when all bulk operations are completed, the solver performs a single optimization run including all updates performed.

The relevant configuration parameters for the Mixed Integer Linear Programming solver are depicted in Table 5.2.

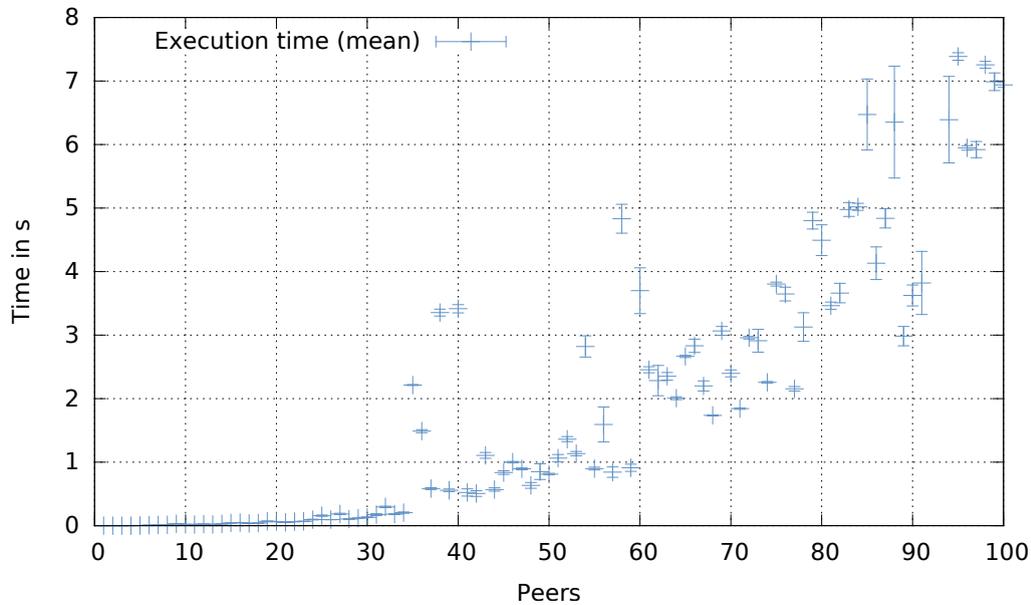
**Tab. 5.2:** Configuration Values for the MILP Solver

Name:	MLP_MAX_DURATION	Type:	time
Default:	3 s	Range:	$\geq 0$
Maximum duration for a Simplex solution process			
Name:	MLP_MAX_ITERATIONS	Type:	int
Default:	$\infty$	Range:	$\geq 0$
Maximum number of iterations for a Simplex solution process			
Name:	MLP_COEFFICIENT_D	Type:	float
Default:	1.0	Range:	$\geq 0$
Weight $D$ for diversity objective			
Name:	MLP_COEFFICIENT_U	Type:	float
Default:	1.0	Range:	$\geq 0$
Weight $U$ for utility objective			
Name:	MLP_COEFFICIENT_R	Type:	float
Default:	1.0	Range:	$\geq 0$
Weight $R$ for relativity objective			
Name:	MLP_MIN_BANDWIDTH	Type:	int
Default:	1024 bytes/s	Range:	$\geq 0$
Minimum bandwidth $b_{min}$ required			
Name:	MLP_MIN_CONNECTIONS	Type:	int
Default:	4	Range:	$\geq 0$
Minimum number of connections $n_{min}$ required to ensure diversity			
Name:	MLP_MAX_MIP_GAP	Type:	float
Default:	0.025	Range:	$[0 \dots 1]$
Tolerance gap $g_{mip}$ used with branch-and-cut solver			

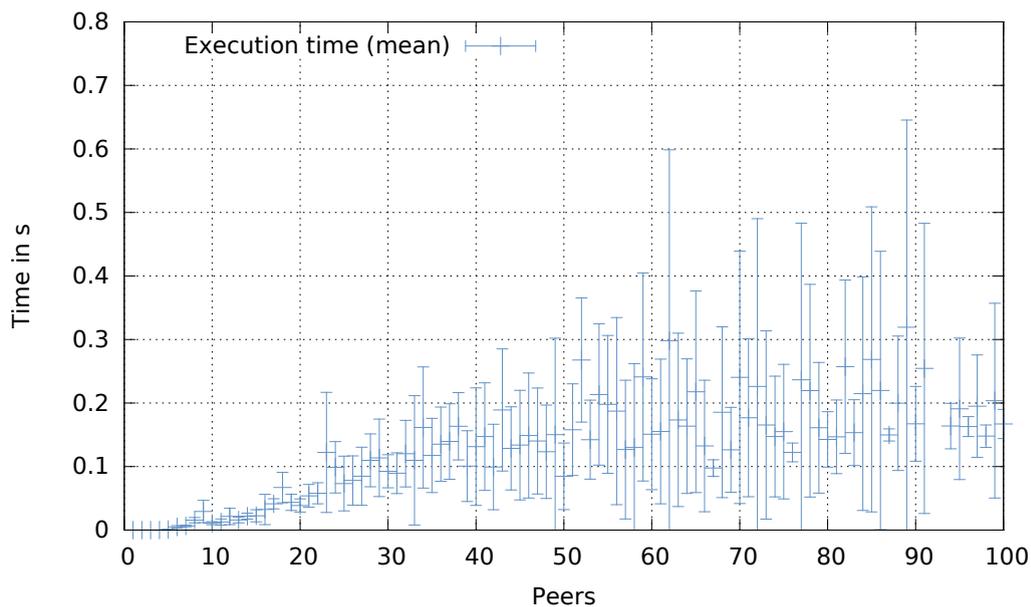
### 5.8.10.1 Performance Evaluation

We evaluated the performance of the MILP solver with respect to scalability and performance using the same tool as described with the heuristic solver. To benchmark the solver we incrementally added peers to the problem to solve and forced the solver to find a solution to the updated problem. For the MILP solver particularly interesting is the performance for a full and an updated problem and if we can benefit from the warm start property of the Simplex algorithm. The result of the benchmark is depicted in Figure 5.5. For this benchmark we incrementally added a peers with 10 addresses to the problem and the requested address suggestions for this peer. This change in problem size forces

the MILP solver to recreate the problem and fully resolve the problem. The resulting performance is depicted in Figure 5.5(a). After this problem was solved, we updated the performance properties of 10% of all addresses while locking the solver with a bulk operation. After updating all addresses, we released the bulk lock and measured the time the solver needed to solve the updated problem. The execution time of the solver to find a solution is depicted in Figure 5.5(b).



(a) Execution Time of the Mixed Integer Linear Programming Solver for a Full Solution



(b) Execution Time of the Mixed Integer Linear Programming Solver for an Incremental Solution

**Fig. 5.5:** Performance of the Mixed Integer Linear Programming Solver

### 5.8.11 The Reinforcement Learning Solver

The implementation of the reinforcement solver provides two important data structures. On the one hand the solver has to maintain a set of learning agents, a dedicated agent for each peer known (not only peers requested) by the transport infrastructure. Each agent stores information if the peer is currently requested for address suggestions and maintains a list of addresses available for the peer it is responsible for and the address the agent has chosen to be active. Besides this transport specific information, the agent has to store information for the Temporal Difference (TD) learning algorithms. The solver uses a the matrix  $W$  for action-value estimation. This matrix has a column for each action available to the agent and a row for each prototype state depending on the number of RBF prototype states as described in Section 5.6.2. A second matrix  $E$  is responsible to store information for eligibility traces and storing information about the last action taken and the last state-feature vector used.

As described with the design of the RIL solver, we define a set of actions the agent can perform. With the current implementation the agent can:

- increase the inbound bandwidth assigned by a certain amount
- increase the outbound bandwidth assigned by a certain amount
- double the inbound bandwidth assigned
- double the outbound bandwidth assigned
- decrease the inbound bandwidth assigned by a certain amount
- decrease the outbound bandwidth assigned by a certain amount
- halve the inbound bandwidth assigned
- halve the outbound bandwidth assigned
- do nothing

All of these operations can be combined with the action to switch to a certain address.

During startup the solver loads configuration settings from configuration and stores the inbound and outbound quotas passed for each network scope in a list. Different to the heuristic solver and the MILP solver, the RIL solver performs learning steps independent from changes to its environment in regular intervals. The frequency of these steps is determined by the adaptive stepping described in Section 5.6.2 and the (configurable)  $t_{\min}$  and  $t_{\max}$  parameters, defining the minimum and maximum period between two learning steps.

When the solver is notified about addresses getting available for a peer, the solver first checks if a learning agent for this peer is active. If no agent for this peer exists, the solver initializes a new agent with default values and prepares the required data structures. When an agent for this peer is ready, the solver adds the address to solver and increases the data structures  $E$  and  $W$  to contain this address and unblocks the agent to start learning and perform a learning step. When an address is deleted from a peer, the agent removes this address from the agent and the data structures required for TD learning and if the address was the selected as active, the solver suggests an alternative address. When addresses are added or change the network scope, the solver first checks if at least the

minimum bandwidth  $b_{min}$  is available in the new network scope and otherwise the address is marked as inactive and not used.

When the solver is scheduled to perform a learning step or due to changes in the environment, the solver starts with calculating the global discount for all agents and updates information for all network scopes about bandwidth assigned based on the assignment done agents and bandwidth utilized based on transport performance properties provided by the transport underlay. This update also includes the fairness of the current allocation with respect to the social welfare algorithm used. The solver supports both the Nash product and the egalitarian welfare algorithm described in the design. The resulting welfare value is stored within the network and used later by the agents to calculate the reward value they receive. After these updates, the solver performs a learning step for all agents currently active.

In a learning step, the agent first determines the state vector currently available and calculates the reward the agent retrieves based on penalty, agent utility and social welfare obtained for all agents with addresses in the same network. The penalty is applied when the last action increased the bandwidth and over-utilized available resources by giving a negative reward increasing quadratically. The utility is calculated based on how good the current address matches the application preferences with respect to transport performance properties and bandwidth assigned. The reward function also includes a bonus for doing nothing and getting no penalty for this action to improve the convergence behavior of the solver. Based on this reward and the state vector obtained from the RBF function, the agent selects the next action and updates the agents allocation policies. Here it depends if the agent uses the Q-Learning or the SARSA algorithm as described in the solver design. With Q-Learning, the solver updates the agent with the reward calculated, the state vector and the action with the highest return the agent can choose in the next step. This does not require this action to be performed, instead the agent can perform any action obtained from the action-selection strategy used. With SARSA, the agent updates the strategy with the calculated reward, the current state vector and the action obtained to perform in the next step. The action selected to be executed next is obtained using an action selection strategy as described in the solver design. This strategy provides the trade-off between exploitation and exploration and returns the action to perform. The solver implements a decreasing exploration strategy as defined in the design. When the environment for an agent is updated, the exploration rate is reset to the default value to allow the agent to learn about new addresses. The solver supports both the  $\epsilon$ -greedy and the softmax strategy and uses the algorithm defined in the configuration.

After performing the learning step, the solver updates the network scopes again to reflect the changes caused by the learning step and notifies the ADDRESS component about the changed resource assignments and addresses selected to be active. Finally, the solver schedules the next learning step according to the current state of the adaptive stepping mechanism controlling the learning speed as described in Section 5.6.2.

The relevant configuration parameters for the Reinforcement Learning solver are depicted in Table 5.3.

### 5.8.11.1 Performance Evaluation

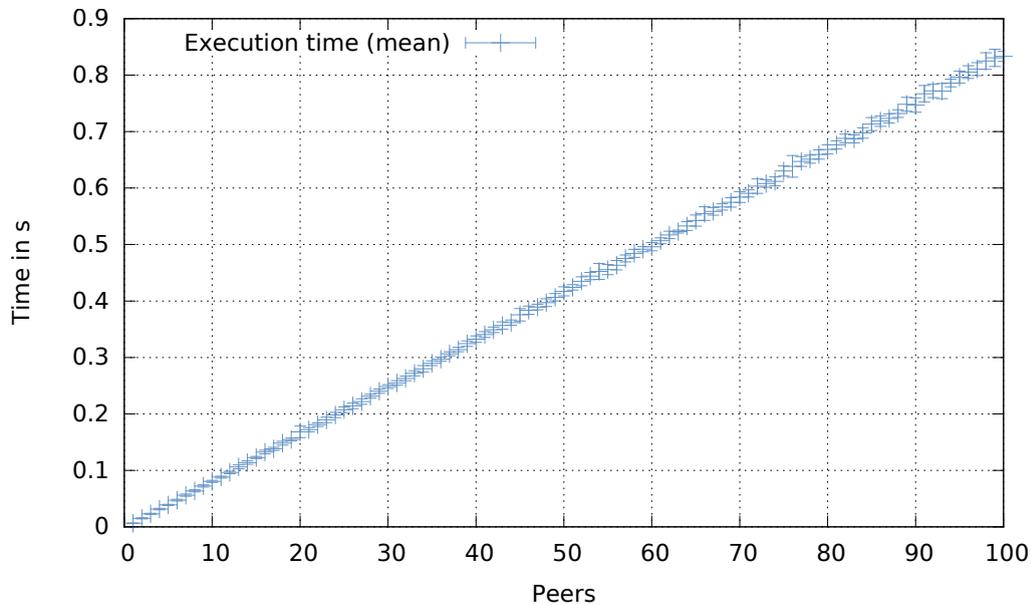
Similar to the greedy heuristic and the MILP solvers presented before, we evaluated the performance and scalability of the RIL solver with exactly the same evaluation tool and under the same settings: we step by step increase the number of peers in the problem setting and for each problem size we measured the execution time to solve the problem. In

**Tab. 5.3:** Configuration Values for the RIL Solver

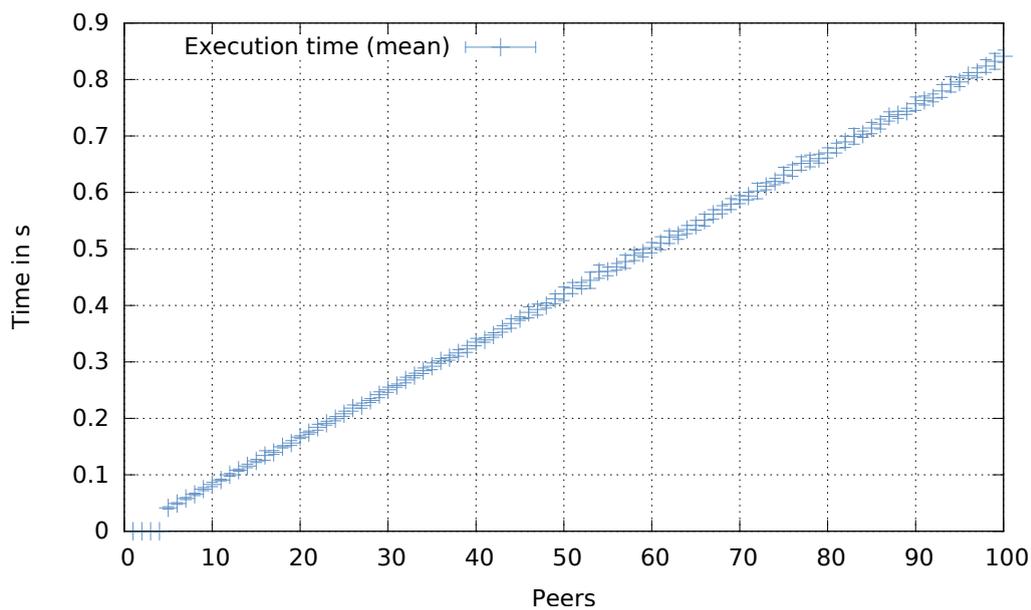
Name:	RIL_ALGORITHM	Type:	string
Default:	SARSA s	Range:	{SARSA, Q-LEARNING}
Temporal Difference (TD) distance algorithm used			
Name:	RIL_SELECT	Type:	string
Default:	SOFTMAX	Range:	{EGREEDY, SOFTMAX}
Action selection strategy to use, $\epsilon$ -greedy or softmax			
Name:	RIL_SOCIAL_WELFARE	Type:	string
Default:	NASH	Range:	{NASH, EGALITARIAN}
Social welfare mechanism to use: Nash product or egalitarian welfare			
Name:	RIL_RBF_DIVISOR	Type:	int
Default:	50	Range:	$\geq 0$
Number of prototype states for RBF state estimation			
Name:	RIL_STEP_TIME_MIN	Type:	time
Default:	200 ms	Range:	$\geq 0$
Minimum time between learning steps			
Name:	RIL_STEP_TIME_MAX	Type:	time
Default:	2 s	Range:	$\geq 0$
Maximum time between learning steps			
Name:	RIL_DISCOUNT_BETA	Type:	float
Default:	0.6	Range:	[0..1]
Learning discount $\beta$ in the TD-update for semi-MDPs			
Name:	RIL_DISCOUNT_GAMMA	Type:	floats
Default:	0.5	Range:	[0..1]
Learning discount factor $\gamma$ in the TD-update			
Name:	RIL_GRADIENT_STEP_SIZE	Type:	floats
Default:	0.01	Range:	[0..1]
Gradient step size $\alpha$ . Determines how much learning experience will update current strategy, used in Q-Learning and SARSA update rule			
Name:	RIL_DEFAULT_EXPLORE_RATIO	Type:	floats
Default:	1	Range:	[0..1]
Exploration ratio $r$ , with exploitation rate $e = 1 - r$ , used with $\epsilon$ -greedy action selection strategy			
Name:	RIL_DEFAULT_EXPLORE_DECAY	Type:	floats
Default:	0.95	Range:	[0..1]
Decay $r_d$ of exploration ratio used with $\epsilon$ -greedy action selection strategy			
Name:	RIL_DEFAULT_TEMPERATURE	Type:	floats
Default:	0.1	Range:	$\geq 0$
Default exploration ratio $r$ used with softmax action selection strategy			
Name:	RIL_DEFAULT_TEMPERATURE_DECAY	Type:	floats
Default:	1	Range:	(0..1]
Decay $r_d$ of exploration ratio used with softmax action selection strategy			

addition, we measured the time to solve the updated problems when 10% of all addresses were updated.

The result is depicted in Figure 5.6(a) for the full solution run and in Figure 5.6(b) for the updated problem. For the reinforcement learning solver, we expect no difference between the performance for a full solution cycle and a solution cycle for an updated problem since the solver has to perform the same operations in both cases. As we can see, the time to solve this problem is linear for both situations over the number of peers.



(a) Execution Time of the Reinforcement Learning Solver for a Full Solution



(b) Execution Time of the Reinforcement Learning Solver for an Incremental Solution

**Fig. 5.6:** Performance of the Reinforcement Learning Solver

## 5.9 Evaluation

To evaluate the performance and quality of the solution provided by the address selection mechanism and the three different solution approaches, we compared in addition to the performance and scalability of the solvers the quality of the solutions provided by the different approaches.

To evaluate the quality of the solutions provided by the solvers, we created a benchmarking tool we can use to run experiments comparable between the different solvers in a reproducible way. Our tool is based on the idea to define typical *scenarios* we want to analyze the solvers in. These scenarios represent typical processes in a decentralized networking environment. The timeline of a scenario is divided in *episodes* with a certain duration. With the benchmarking tool we can define scenarios with peers joining and leaving the network, addresses becoming available and unavailable, connections with peers being requested and canceled. In addition, we can provide the solver with transport performance properties for the different transport properties available and application preferences for peers and transport preferences. The benchmarking tool provides functionality to generate these inputs with so called *generators*. These generators can generate properties and preferences with a constant, linear and sinusoidal course of the function or with random properties with a configurable base value, maximum amplitude, frequency and duration. A typical scenario description is a set of episodes defining events in the network. These events are given to the solver to test. The solver uses this inputs and calculates a solution it returns to the benchmarking tool. The benchmarking tool logs the output passed by the solver in regular intervals for later analysis.

To benchmark the three solvers, we defined several scenarios and ran these scenarios with all of the three solvers. To measure the quality of the solutions provided by the solvers, we defined a quality function similar to the objective function of the MILP solver representing the objectives defined in Section 5.2.3. The quality function includes the utility of the current solution, relativity of the assignment with respect to preferences, the number of peers selected to be active and the quality with respect to transport performance properties and, different to the objective function of the MILP solver, a penalty if resources are over-utilized. In contrast to the objective function of the MILP solver, the goal function includes a penalty  $x(t)$  if resource (bandwidth) constraints are violated. This is required to penalize the reinforcement learning solver for over-allocation.

The quality function is in detail defined as follows:

Given a neighbor  $p \in P$  with inbound bandwidth  $b_{p,in}$  assigned and outbound bandwidth  $b_{p,out}$  assigned, where the address is selected from network scope  $n$  with a network scope inbound quota of  $l_{n,in}$  and outbound quota of  $l_{n,out}$ , and current properties values  $q_i$ , preferences  $Q_i$  for properties  $i \in P \times A$  for addresses  $A$ , including specifically a preference of  $q_{p,bw}$  for bandwidth, and a penalty  $x(t)$  in case of overallocation, the quality function is calculated using the following four factors:

- Relativity:  $r(t, p) = q_{p,bw} \frac{b_{p,in}(t) + b_{p,out}(t)}{l_{n,in} + l_{n,out}}$
- Diversity:  $d(t) = \frac{\text{peers}_{active}(t)}{\text{peers}_{requested}(t)}$
- Utility:  $u(t, p) = \frac{b_{p,in}(t) + b_{p,out}(t)}{l_{n,in} + l_{n,out}}$
- Quality:  $q(t, p) = \sum_{i \in P \times A} q_i(t) \cdot Q_i(t)$

The overall quality function for an experiment running from  $t_{start}$  to  $t_{end}$  is then

$$G = \int_{t_{start}}^{t_{end}} d(t) - x(t) + \sum_{p \in P} r(t, p) + u(t, p) + q(t, p) dt.$$

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. The three scenarios are modeled to represent the behavior of a file sharing application, a telephony application, and the case where both file-sharing and telephony execute together. We use two network scopes, scope  $n_0$  with a large amount of bandwidth available, and  $n_1$  providing only half the bandwidth of  $n_0$ . 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  $n_0$  to provide better latency properties than for the addresses in scope  $n_1$ , 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.

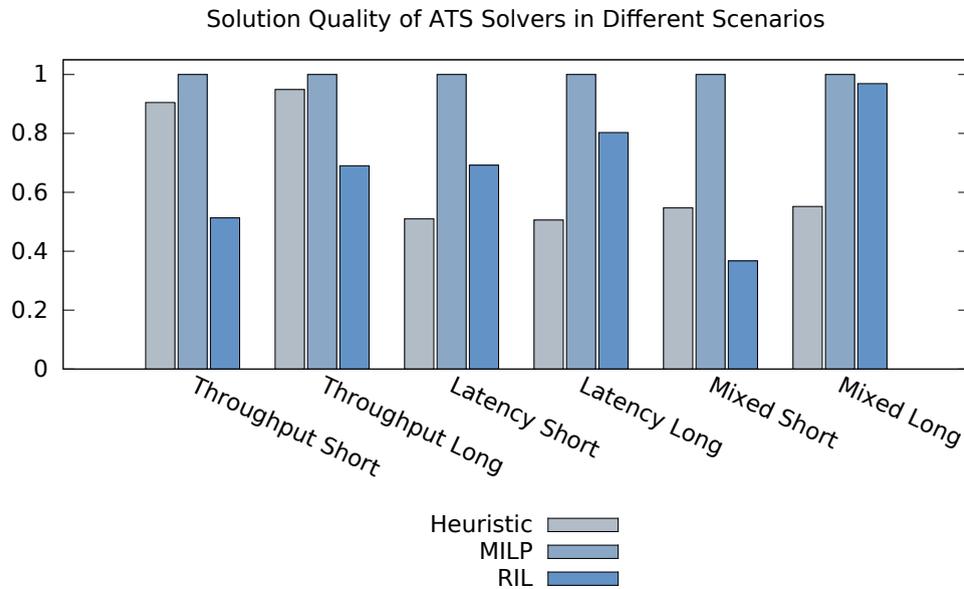
In the *latency* scenarios, we emulate an application requiring low latency values to communicate with both  $p_0$  and  $p_1$ . We generate delay values for the addresses of neighbor  $p_0$  and  $p_1$  located with  $s_0$  with random values between 20 and 25 ms and better delay values for addresses in scope  $n_1$ , providing less bandwidth, with delay values between 10 and 15 ms for  $p_0$  and 1 and 30 ms for  $p_1$ . We then begin to issue our preference for both peers with respect to latency, linear increasing values for  $p_0$  and sinusoidal for  $p_1$ .

In the *mixed* scenario, we simulate a peer-to-peer framework with applications issuing conflicting preferences. For the addresses located in  $n_0$  providing more bandwidth, we create delay properties for both addresses with values between 20 and 25 ms, whereas for the addresses in scope  $n_1$  providing less bandwidth, we create delay values between 10 and 15 ms for  $p_0$  and 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 maximize delay.

Table 5.4 and Figure 5.7 gives the results for the goal function (5.9) 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.

## 5.10 Discussion

Our heuristic computes 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 therefore it performs worse than MILP and RIL solver in most scenarios. Only in the throughput scenarios, where only the amount of resources assigned to the peers is important, the greedy heuristic performs almost as good as the MILP solver. Also, the heuristic does not benefit from the requirement to solve the problem repeatedly and therefore does not benefit from running in longer scenarios.



**Fig. 5.7:** Solution Quality of ATS Solvers in Different Scenarios

**Tab. 5.4:** 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

Treating the address selection and resource allocation process as an optimization problem and using optimization techniques has multiple advantages. The solution found by this approach is always an optimal solution and the different objectives can easily be weighted according to the application's needs by adapting the respective 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. This requirement to formulate constraints and objectives using linear equation make constraints impossible where several inputs and objectives directly depend on each other or are used in the same function and constraints not expressible as linear dependencies cannot be formulated. Requiring the output to contain binary variables to indicate if an address is selected makes solving the problem significantly more expensive since solving a mixed integer problem is NP-hard, while for linear programming polynomial time algorithms exist.

The runtime can be reduced by exploiting the fact that the Simplex algorithm can re-use an existing solution if only the coefficients in the problem changed (updated performance properties and preferences) but not the size of the problem itself (peers joining and leaving, addresses being added or removed). Furthermore, 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.

While MILP provides optimal solutions with respect to its objective function, an a-priori 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 to meet critical application requirements might be more desirable than an application that sticks to such 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. In addition is the solver performing sub-optimally if the requirements for an optimal solution change, so the objective of the allocation strategy the solver is supposed to learn in the current environment.

A special challenge for automatic transport selection and resource allocation mechanisms is traffic not designated for the local peer but forwarded or relayed to other peers in the peer-to-peer overlay. Components realizing such functionality are for example the DV component of GUNet’s transport infrastructure described in Section 4.3.18 or the traffic relaying functionality of GUNet’s CADET service described in [PG14]. Such traffic should not be accounted to the resource consumption of applications running on the local peer but has to be considered when enforcing resource restrictions on a peer. In addition, the question has to be answered which amount of a peer’s resources a user is willing to contribute to relay traffic to other peers. The current implementation of the TRANSPORT service and the ATS service support such functionality using information about the overlay hop distance for the DV component. With this information, ATS solvers can prefer direct connections with other peers over connections relaying traffic to the destination over intermediate hops. To be able to consider such functionalities with respect to resource consumption, applications have to collaborate with both the communication infrastructure as well as with the ATS service to indicate if traffic is designated for the local peer and has to be accounted to its resource consumption or if traffic is designated for an other peer in the overlay and has to be accounted to the resources available to relay traffic to other peers.

## **5.11 Conclusion and Findings**

With the ATS mechanism presented in this section, we realized an important building block for censorship-resistant peer-to-peer networks. It is designed both with a focus on improving resilience for the communication underlay by selecting a best performing address mechanism for the requested peers and also respects the requirements for higher layer applications. The proposed ATS approach tries to increase the quality and performance of communication in the peer-to-peer overlay network by favoring mechanisms providing good communication properties and considering application requirements when selecting these mechanisms and distributing resources among peers. In case of service degradation, the presented approach can detect and counteract such attempts and switch to a mechanism providing better communication properties than the degraded mechanism. The presented design does not contradict the idea of a neutral network since it tries to achieve best

possible communication and does not discriminate against other traffic and complies with the idea of a *dumb* network since every peer uses its local view of the network to find a solution to the problem. To solve the problem of address selection and resource allocation, we presented three different solution approaches each chosen for some specific property it provides. We evaluated the different solvers and saw with the MILP solver, that treating the address selection and resource allocation with an integrated approach is beneficial and improves quality of the solution. Using the machine learning techniques is a promising approach as we saw with the quality of the solutions provided by the RIL solver in scenarios running for a longer time giving the RIL solver time to adapt to the environment and learn a suitable allocation strategy.

The use of learning techniques in this domain is new and is therefore worth deeper investigations. Improved heuristics and fitness functions better adapted to application requirements should also be investigated. In this work we merely present an initial design showing the suitability of the approach. In future work, the different parameters for available algorithms should be analyzed with respect for their impact on the quality and a deeper understanding of multiple agent systems versus using a single agent system should be obtained, to see if using a single allocation strategy has benefits over the use of social welfare algorithms. Future work should also focus on how traffic relayed to a peers is treated and accounted in the transport selection and resource allocation mechanism.



## 6. GNS - A DECENTRALIZED, PRIVACY-PRESERVING AND CENSORSHIP-RESISTANT NAME SYSTEM

One of the most important services on today's Internet is the Domain Name System (DNS), used to map human-memorable names to IP addresses used by computers to address each other in the network. The human-memorable names provided by DNS are one of the important building blocks for the World Wide Web and its idea to link information and resources scattered over different servers. Due to its importance for today's Internet and its design and architecture, DNS is also a valuable target for malevolent parties trying to prevent and suppress communication between users and to make information on the Internet inaccessible. DNS architecture also allows powerful attackers to monitor which websites, services and information users access, allowing such an attacker to establish a system of mass surveillance. With DNS and its functionality to perform a reverse lookups from names to IP addresses and to acknowledge the non-existence of a domain name, an attacker can use DNS to analyze network infrastructure and enumerate IP networks.

Name systems are not only useful to map names to IP addresses but can also be used in different contexts. The approach to (securely) map names to identifiers is used in various other situations. Many services, like social networks, messaging or telephony providers, today relay on centralized services to name, identify and authenticate their users. When using centralized approaches, similar to DNS for name resolution, these services are easy to attack and censor due to their centralized architecture. And when identity services fail, all service building on top of these services also fail.

In this chapter we present the GNU Name System (GNS), a fully decentralized, privacy-preserving and censorship-resistant name system as an alternative to the DNS and other public key infrastructures and identity management systems. The GNS provides memorable and secure but not globally unique names. It protects the users' privacy by providing confidential publication and resolution of name/value mappings. Due to the design of GNS, it can be used as a replacement for security infrastructures like the X.509 public key infrastructure. In the following section, we will give an introduction to DNS and its limitations, explore the possible design space for name systems and present the adversary model used in this work. We then present the design and implementation of GNS.

Parts of this chapter were previously published in [WSG13] and [WSG14] and researched in collaboration with other researchers and students. An earlier design and implementation were previously described in Martin Schanzenbach's work [Sch12].

### 6.1 Introduction and Motivation

Preventing users from communicating and blocking access to information on the Internet can be achieved in various ways and for various motivations as we saw in Chapter 1. Besides preventing communication between participants on the network level, access to information and services can also be censored by making information unavailable and inaccessible for the users. An easy to employ and therefore well-known and widely used approach to make information inaccessible is to tamper with the name systems and ma-

nipulate mappings between human-meaningful names for services and the addresses used by machines to communicate in the network.

Name systems are a key service for the Internet and vital for most Internet applications. IP addresses used by computers to address each other in the network are not suitable for humans to remember. The current standard of IPv4 uses 32-bit addresses usually given in the form of four octets separated by dots. The upcoming IPv6 standard, already in use for example in China and with some ISPs in Europe and the United States, relies on 128-bit addresses written as eight groups of four hexadecimal digits each, separated by a colon: such addresses are typically beyond human capacity to memorize. The standard on today's Internet to map human-memorable names to addresses useful for machines to communicate is the Domain Name System (DNS). DNS, as briefly introduced in Section 2.4, is a decentralized but hierarchical name system with the main purpose to map human-meaningful names to IP addresses. When a user accesses a service, he provides a DNS name which is then translated to an IP address the computer can use to establish a connection to this service. Besides this core functionality, DNS increasingly provides more widely used functions such as service discovery (e.g. with MX records for mail servers or SRV records used with the Session Initiation Protocol (SIP) and the Extensible Messaging and Presence Protocol (XMPP)), or mappings from names to arbitrary values (e.g., to X.509 certificates that authenticate web servers). DNS is also often employed to increase resilience for services on the Internet, using DNS to provide redundancy. DNS can be used for load balancing and fail-over and DNS allows for seamless migration of services from one physical system to another by using a DNS name to address a service instead of an IP number to address a particular machine.

Based on the wide range of functionalities provided by DNS and their importance for services, applications and communication on the Internet, DNS is one of the most important services for today's Internet. A failure in DNS can have an enormous impact not only making websites inaccessible but also with serious impact on economy, technical processes and societies. We can observe the importance for free and unrestricted communication and how it can influence developments with events like the Green Revolution in Iran, the Arab Spring and recent political developments in Turkey and the Ukraine. The awareness for DNS and its importance for the functioning of the Internet has increased, as we can see with the European Parliament emphasizing the importance of maintaining free access to information on the Web in a resolution [Eur11] and Tim Berners-Lee, one of the inventors of the WWW, stating:

*The Domain Name Server is the Achilles heel of the Web. The important thing is that it's managed responsibly.*

With the increasing importance of the Internet and DNS as one of the important building blocks for the functioning of the Internet, interest for DNS arises not only from benign parties.

With the commercialization and internationalization of the Internet in the last two decades, the Internet and also the DNS are more and more at the mercy of commercial and political interests. Often these economic interest can succeed over the public interest just due to the economic power of the parties involved. Decisions with respect to the functioning of the Internet and its future developments are not driven by technological considerations anymore but by political and economic motivations. For DNS, a prime example is the introduction of new gTLDs (like ".jobs" or ".asia" and many more) by ICANN and IANA, under contract with the United States Department of Commerce (DOC), in 2012. With this process we can see the economic predominance on today's

Internet: to apply for such a gTLD, an initial fee of 185,000 US-\$ and an annual renewal priced with an additional 25,000 US-\$ per year plus additional transaction fees have to be paid by the applicant<sup>1</sup>, automatically limiting the possibility to apply for such a gTLD to financially strong organizations: so Google Inc. applied for 101 gTLDs, Amazon.com Inc. for 76 gTLDs and a company called donuts<sup>2</sup>, with the only purpose of being a registrar to sell these names, is applying for a total number of 307 gTLDs. Not only is this process dominated by commercial interests, these interests also conflict with public interest, as we can see with incidents like the companies Ferrero SpA registering the gTLD “.kinder” [Mey14] and Amazon.com Inc.’s (rejected) attempt to register the gTLD “.amazon” [Ber14], both being generic terms used in many other relations and with such a registration being successful, exclusively being given to a company. Besides having economic implications, this process can even have political disputes as a consequence as we saw with planned introduction of a gTLD “.wine”, a plan strongly objected by the French government to protect the interests of their wine-producing industry having political disputes between multiple governments and the ICANN as a consequence [Fiv14].

In addition to economic and political interests also malevolent and suppressive parties are interested in DNS and its importance for the functioning of the Internet. 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], as we have seen in the introduction in Chapter 1. One of the popular and easy to employ methods to make information on the Web unavailable is to tamper with information in DNS required to access these information and services. Since manipulating the DNS system is easy to perform and effective on the other hand, it is one of the most common methods for censors to make information inaccessible by making resolution of name/value mappings fail or modify them to redirect the user to a censor controlled impostor site. We saw this for example with Turkey’s blocking of the Twitter microblogging service and as a consequence also Google DNS servers in 2014 [Raw14, HA14]. The structure and design of DNS, based on names being *owned* by organizations and delegation of control to subordinated organizations, even simplifies the realization of these attacks, especially for adversaries like the adversary used in this work as described in Section 6.3.1. Since DNS domains are typically owned by organizations, an adversary can put pressure on these organizations by applying his legal or judicial powers to get control over these names or to remove them. Prominent examples (all taking place only within one week in mid-2014) demonstrating the power of such an adversary and its’ consequences are the (attempted) seizure of Iran’s top level domain “.ir” and its IP block by an US court to compensate Israel terror victims [Sha14] and Microsoft’s takeover of more than 20 DNS domains owned by the DDNS provider No-IP<sup>3</sup> based on a court granting a temporary restraining order against No-IP based on the accusation to distribute malware. So tampering with the DNS can cause collateral damage, too: As a consequence of Microsoft’s take-over, many legitimate domains hosted with No-IP failed to resolve due Microsoft’s DNS servers being overloaded and failed to respond. A recent study [Ano12] showed that Chinese censorship of the DNS has had worldwide effects on name resolution. Other collateral damage can be the filtering of benign websites as in [Kan14, Cow14] and [Hol12], already described with DNS manipulation in Section 2.4. Besides tampering with DNS to make services and information unavailable, it can also be used to prosecute users accessing information designated as *illegal* and therefore suppressed by an censor. Here information leaking

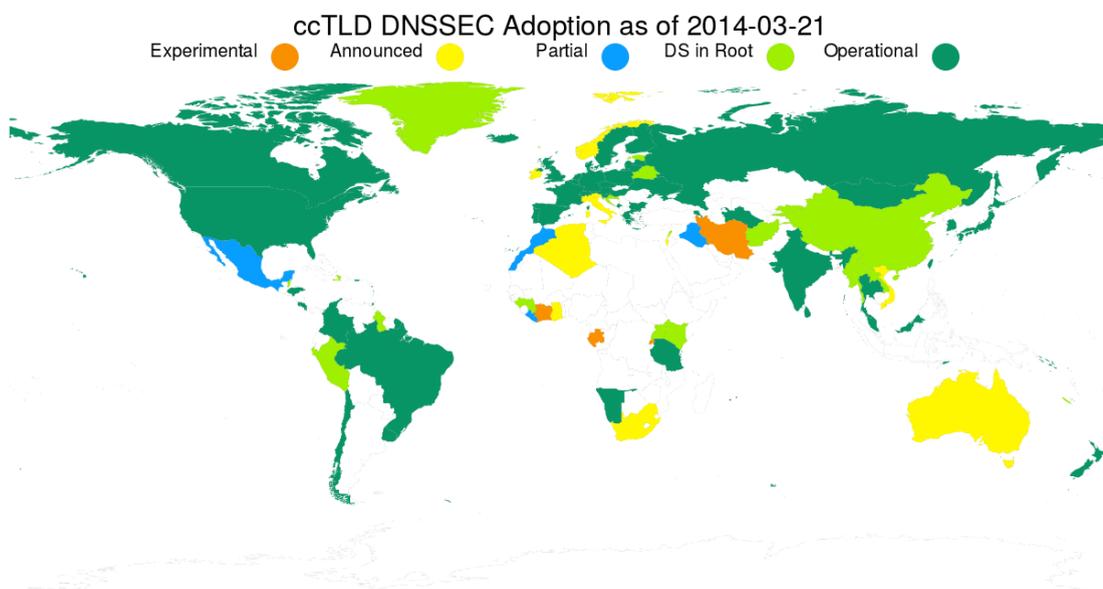
<sup>1</sup> <http://newgtlds.icann.org/en/applicants/customer-service/faqs/faqs-en>

<sup>2</sup> <http://www.donuts.co/about/>

<sup>3</sup> <http://www.noip.com/>

from the DNS resolution process can be used by an adversary to identify users accessing illegal information and based on this information prosecute the users using executive or legislative powers.

DNS was not designed with security as a goal and it does not provide any mechanisms to ensure authenticity, integrity or confidentiality of information published or resolved with DNS. This makes DNS 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 Domain Name System Security Extensions (DNSSEC). DNSSEC [AAL<sup>+</sup>05a] only adds data integrity and origin authentication to DNS, but does not address confidentiality of queries. Also, 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 accessing the queried servers. 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 security vulnerabilities [HS13]. In mid 2014, DNSSEC is still not deployed for all ccTLDs as we can see with the *DNSSEC Deployment Maps* provided by the Internet Society<sup>4</sup> also depicted in Figure 6.1. Even fewer DNS requests are validated with DNSSEC as depicted on APNIC's *World Map of DNSSEC Deployment*<sup>5</sup>. Here even in most high ranked countries only a maximum of 30% of all requests are validated with DNSSEC as shown in Figure 6.2. In combination with the use of stub resolvers in most operating systems (explained in Section 2.4) can DNS and DNSSEC not provide end-to-end authenticity for name resolution today.



**Fig. 6.1:** DNSSEC Adoption in ccTLDs in March 2014

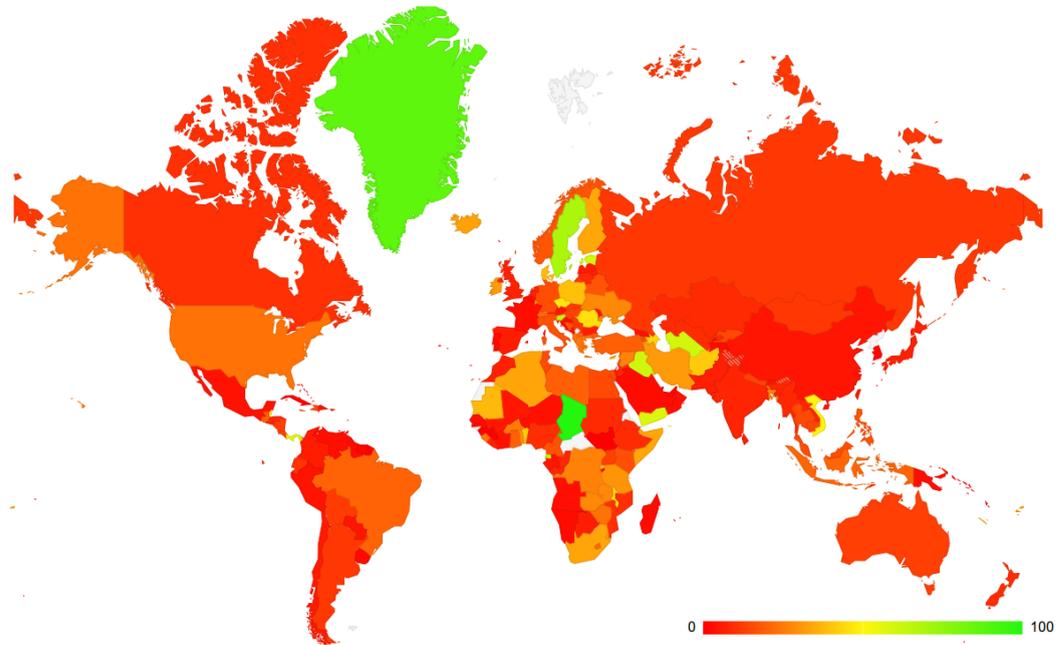
Source: Internet Society DNSSEC deployment statistics

Finally, DNSSEC is not designed to withstand legal attacks. Depending on their

<sup>4</sup> <http://www.internetsociety.org/deploy360/dnssec/maps/>

<sup>5</sup> <http://gronggrong.rand.apnic.net/cgi-bin/worldmap>

DNSSEC Validation Rate by country (%)

**Fig. 6.2:** Validation Rate of DNSSEC Requests in September 2014

Source: APNIC's World Map of DNSSEC Deployment  
Obtained in September 2014

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 [Kar12, DK12], in which the Danish police 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 DNS have owners, and ownership can be taken away by different means.

So for a resilient and censorship-resistant communication infrastructure it is not only important to provide and improve connectivity between participants to provide resilient communication, it is also necessary for such an infrastructure to provide a resilient and privacy-preserving way to make information and services addressable and accessible. Only the combination of resilient communication and resilient addressing of information and services can provide a holistic resilient infrastructure.

One of the key design aspects making current name resolution with DNS attackable and DNSSEC useless with our adversary, is the hierarchical design of DNS and the requirement of trusted, centralized registrars. Especially in fully decentralized peer-to-peer systems, not relying on any centralized or trusted instances, the use of a name system managed by organizations that may easily fall under the influence and the sphere of control of an adversary is not a viable option and we need to provide an alternative following the ideas and design principles of the underlying peer-to-peer system.

## 6.2 Background

### 6.2.1 The Domain Name System

The Domain Name System (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 organized as a distributed, hierarchical database. It provides mappings from names to values and stores name/value mappings in so-called *records* in a distributed database.

Names in DNS consist of *labels* delimited by dots. The root of the DNS 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*.

A DNS record consists of a name, type, value and expiration time. 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's authority 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 sub-domain. 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.

### 6.2.2 The Domain Name System Security Extensions

The Domain Name System Security Extensions (DNSSEC) as defined in [AAL+05a, AAL+05c, AAL+05b] add integrity protection and data origin authentication for DNS records. But DNSSEC does neither add confidentiality nor denial-of-service protection. It adds record types for public keys (DNSKEY) 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.

One of the issues related to DNSSEC is that most operating systems APIs do not provide the functionality to handle the DNSSEC security information. The function calls used do not provide the possibility to give the security information to the applications for verification. In addition do most operating systems only provide a stub DNS resolver. These stub resolvers do not perform the full DNS resolution, but give the task to resolve a name to a configured DNS server. By doing so they also rely on security verification performed by the server.

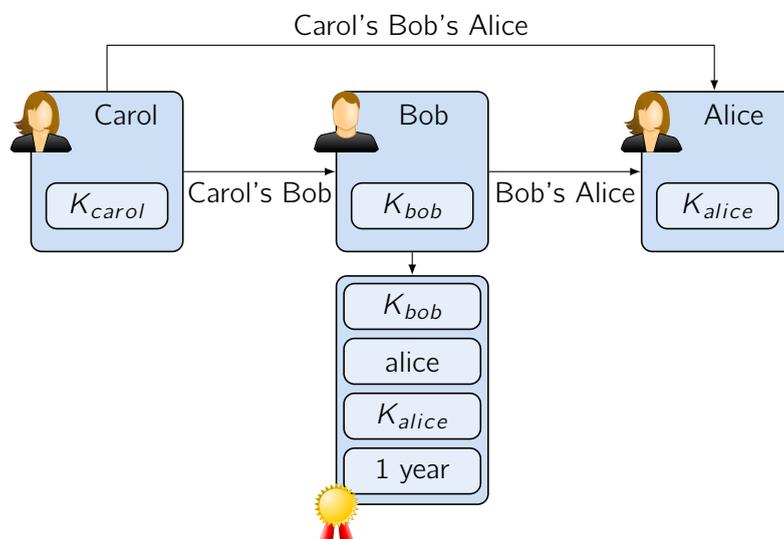
### 6.2.3 SDSI/SPKI

SDSI/SPKI is a merger of the Simple Distributed Security Infrastructure (SDSI) [RL96] and the Simple Public Key Infrastructure (SPKI) [EI96] and is defined in [EI99, EFL<sup>+</sup>99]. SDSI/SPKI was proposed to overcome over-complication and scalability problems of existing security infrastructures like the X.509 public key infrastructure. SDSI/SPKI 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. Each user in the SDSI system is a principal and owns a cryptographic key pair: every user is uniquely identified by the public key and can issue certificates using the private key. SDSI/SPKI does not require a centralized infrastructure since every participant is a certification authority issuing certificates. These principals serve as namespaces within which local names are defined.

A *name* in SDSI/SPKI is a public key and a local identifier, e.g.  $K\text{-Alice}$ . This name defines the identifier *Alice*, which is only valid in the namespace of key  $K$ . Thus,  $K_1\text{-Alice}$  and  $K_2\text{-Alice}$  are different names. SDSI/SPKI allows namespaces to be linked, which results in compound names:  $K_{\text{Carol}}\text{-Bob-Alice}$  is Carol's name for the entity which Bob refers to as  $K_{\text{Bob}}\text{-Alice}$ . Bob himself is identified by Carol as  $K_{\text{Carol}}\text{-Bob}$ . SDSI/SPKI allows assertions about names by issuing certificates. 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. 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.

GNS applies these key ideas from SDSI/SPKI in its name resolution mechanism in order to provide an alternative to DNS. The transitivity at the core of SDSI/SPKI is found in GNS as *delegation of authority* over a name. In both GNS and SDSI/SPKI, name resolution starts with a lookup in the local namespace defined by the local public key.



**Fig. 6.3:** SDSI/SPKI : Principals, Certificates and Linked Local Name Spaces

### **6.2.4 Distributed Storage in Peer-to-Peer Overlay Networks**

In peer-to-peer systems, it is common to use a Distributed Hash Table (DHT) [GRW05] to exchange data with other participants in the overlay. A DHT creates a decentralized key/value store to make mappings available to other users and to resolve mappings not available locally. GNS uses a DHT to make local namespace and delegation information available to other users and to resolve mappings from other users. To increase resilience of the system, several DHTs with a focus on resilience and censorship-resistance exist. Examples for such DHTs are  $R^5N$ , which is particularly useful for restricted route environments and supports replication of data on multiple nodes, described in [Eva11], and X-Vine [MCB11], using a social-based approach to limit the impact of Sybill attacks on peer-to-peer routing. When realizing an application using a DHT to store and retrieve mappings, the choice of DHT used strongly affects availability of GNS data and has a major impact on the overall performance of the system. A current approach of GUNet is to provide an additional DHT implementation based on X-Vine [Sin14]. With this approach, we will be able to evaluate the impact of DHT performance on GNS.

## **6.3 Functional Requirements**

To analyze the requirements a censorship-resistant name system has to fulfill, we start with defining the adversary model used in the remainder, and the attacks a system has to withstand. Based on this adversary model, we then develop functional requirements and the design for an alternative censorship-resistant name system.

### **6.3.1 Adversary Model**

The adversary model we use in the following has to be adjusted and adapted in the context of evaluating name systems. In the following discussion of name systems we assume an adversary modeled after a nation state trying to restrict access to information on the Internet. Our adversary tries to use his legal and executive powers to control name to value mappings and modify or remove these mappings to make information unavailable. The adversary tries to modify name value mappings prior not under his control to make resolution fail or redirect the resolution to an adversary-controlled impostor site. To achieve his goals, he tries to attack directly the parties responsible to manage name/value mappings in the name system.

Our adversary can assume any role in the system and even assume multiple identities. This assumption prevents the use of a trusted third party like certification authorities. In our model no upper limit in the number of adversaries in comparison to benign users is assumed. We can also assume that our adversary has more computational power than all benign users combined. We make this assumption since we expect a name system to be adapted by technical users first and therefore an attacker with supercomputers may initially have more computational power than early adopters.

On the other hand, we assume that the adversary cannot prevent basic communication between participants. In addition, he cannot prevent the use of cryptography and cannot break cryptography. We also have to assume that our adversary does not attack the end systems directly. Fortunately even for a nation state attacking all end systems of all users would be too costly.

### 6.3.2 Functional Requirements for an Alternative Name System

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 used by the network. In addition, name systems provide the possibility to address a service on the network and not a system it is running on, so providing an abstraction between services a user want to access and the physical infrastructure the services are provided by.

An alternative name system has to adapt to existing use cases and usage patterns to allow integration with the existing Internet and applications.

On today's Internet, 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 has to 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.

Linking information is important for applications where users or information are in relation with each other. Existing applications often require to access services directly without following a link. So besides the requirement to link information and provide the possibility to address information relative each other, a name system should provide the functionality to address any participant in the system even without a prior relation existing by supporting absolute names.

## 6.4 Design Space for Name Systems

To propose an alternative, privacy-preserving and censorship-resistant name system suitable for the use and following the demands, requirements and design ideas of peer-to-peer systems, we start with evaluating the possible design space of name systems. For this evaluation, we use a helpful conjecture formulated by Zooko Wilcox-O'Hearn about the design space of name systems called *Zooko's Triangle* [WO01].

Zooko's Triangle states, that names in a name system can have three different properties. According to Zooko, names can be:

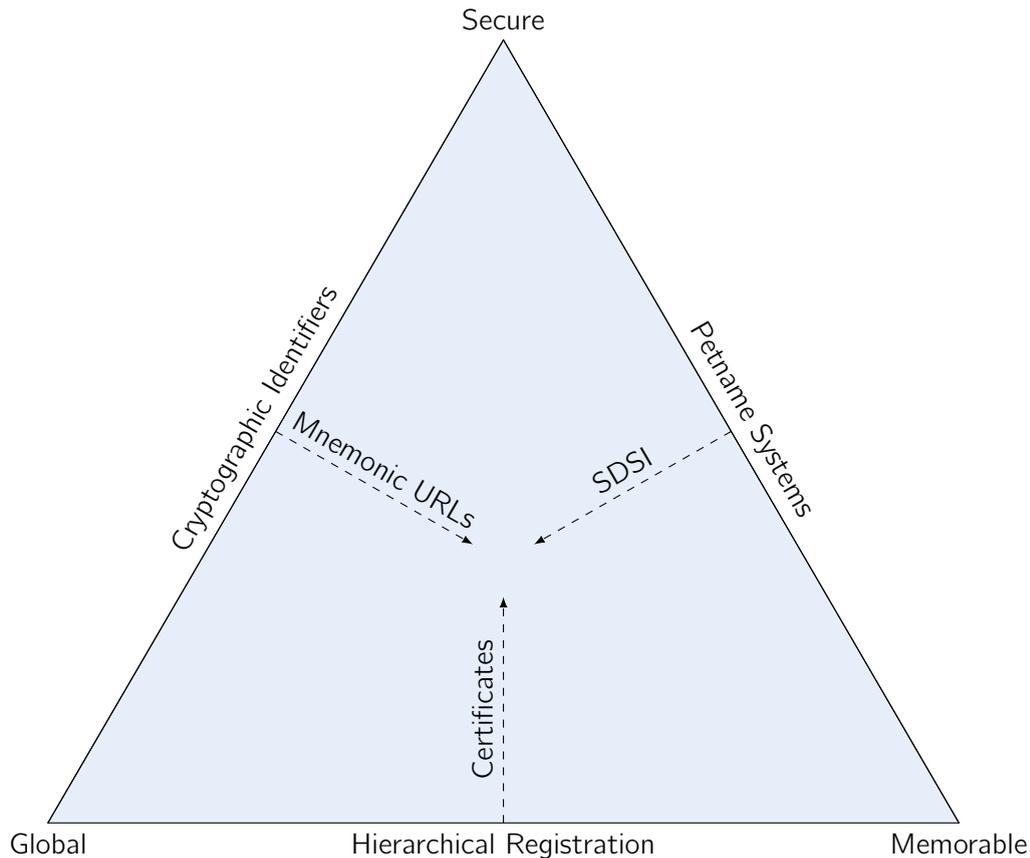
**Memorable:** Names are memorable to humans. They lack entropy and can therefore be enumerated.

**Global:** A global or globally unique name provides the same mapping to a value for all participants.

**Secure:** The name system can retrieve and add name to value mappings even under attack.

Zooko's triangle states that a name system can under our adversary model only achieve two of the three desirable properties. With our adversary, names in a name system cannot be memorable, global and secure at the same time. Name systems therefore have to de-emphasize one of these aspects. This assumption that one property has to be de-emphasized can be ensured with our adversary model and Zooko's triangle is a conjecture valid with our adversary model [WSG13]. If we assume a weaker adversary, not having more computational power than the benign users, one can design a name system having all three properties at the same time [Swa11a, Swa11b]. Zooko's triangle, with the three

possible properties names in name system can have and the possible systems realizing two of the properties and the respective efforts to achieve the missing properties are shown in Figure 6.4.



**Fig. 6.4:** Zooko's Triangle: Design Space for Name Systems

#### 6.4.1 Hierarchical Registration

Name systems based on hierarchical registration, like DNS, can provide global and memorable names. Names are globally unique since names are registered with a registrar having control of the namespace or portions of it. This approach prevents duplicate or inconsistent mappings and ensures consistency for the resolution process. Names in this system can be memorable when registered with the registrar since the registration is controlled by this party. Since names are global, the number of names is limited and important names can be precious creating an economic market to deal with names. With our adversary model this approach cannot be secure, since the adversary can assume any role in the system and can therefore also assume the role of the party managing the namespace. In this role the adversary can control the mappings and modify and remove them on demand. In addition, since names are memorable and lack entropy, the adversary can enumerate all available names and register valid mappings for these names based on the assumption that he can assume an unlimited number of identities and has significant computational power.

### 6.4.2 Adding Security to Hierarchical Registration

A system based on hierarchical registration, like DNS, can try to achieve the missing security property by trying to add authentication to the system. DNSSEC is the standard approach to add security to DNS, but alternatives trying to counteract the weaknesses of DNS exist [Ber08]. On the one hand we have approaches trying to improve the existing DNS system and on the other hand systems with a different design approach. DNSSEC, already described in Section 6.2.2, is an extension for DNS with the goal to add security to DNS based on the threats defined in [AA04]. DNSSEC is defined in [AAL+05a, AAL+05c, AAL+05b] and provides authenticity information for DNS names by cryptographically signing records using a X.509 security infrastructure. In addition to records, DNSSEC also provides functionalities to provide certificates for other services like SSH with SSHFP [SG06], IPsec with IPSECKEY [Ric05] or TLS with TLS Trust Anchors (TLSA) [HS12]. The records secured with DNSSEC are cryptographically signed and so clients can check the authenticity of the records. Using the X.509 security infrastructure, described in Section 2.5, with its hierarchical design, requires the use of so called *trust anchors*. Trust anchors are used to sign the certificates of subordinated organizations and provide a root of trust. DNSSEC is based on chains of trust and makes users validate the chains. The certificates of the trust anchors have to be provided to the user out-of-band, for example by shipping with the operating system.

With respect to the adversary model we have for a name system, DNSSEC cannot help counteracting the adversary. The adversary could put pressure on the organization managing DNS zones, and get them to sign the records to fake authenticity. With an adversary forcing an organization to remove or modify DNS entries, DNS and DNSSEC still return valid responses as the results will still be signed by the responsible organization. Since this approach still depends on the existing DNS system with its hierarchical organization structure and in addition introduces a new dependency to the X.509 security infrastructure, well-known to be prone to attacks and compromise by adversaries similar to our adversary as described in Section 2.5, the idea of extending an existing semi-centralized system prone to be compromised with a second system prone to compromise is not suitable in the focus of this work. A censorship-resistant communication infrastructure for a peer-to-peer system, which explicitly abdicates the use of any centralized authorities, cannot rely on DNS as well as X.509.

The first practical system that improves confidentiality with respect to DNS queries and responses was Bernstein's DNSCurve [Ber08]. In DNSCurve, session keys are exchanged using Curve25519 elliptic curve cryptography [Ber06] and then used to provide authentication and encryption between resolvers, caches and authoritative servers. DNSCurve can be used together with DNSSEC. DNSCurve is deployed with the OpenDNS<sup>6</sup> and OpenDNS resolvers fully support DNSCurve. DNSCurve improves existing DNS 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. To improve confidentiality for users, OpenDNS in addition to DNSCurve to also supports DNSCrypt<sup>7</sup> to encrypt communication between OpenDNS DNS servers and resolvers. Similar to CurveDNS, DNSCrypt is based on ECC using Curve25519 [Den14].

---

<sup>6</sup> <http://opendns.com>

<sup>7</sup> <http://www.opendns.com/about/innovations/dnscrypt/>

### 6.4.3 Cryptographic Identifiers

A name system providing global and secure can be realized using cryptographic identifiers. Each participant in such a system is identified by a globally unique cryptographic identifier other users can use to refer to this user. Since these identifiers contain enough entropy to prevent enumeration and if the name space of the system, in this case the key length, is well chosen, the system is considered to be secure. The adversary, even when assuming multiple identities and having more computational power than the benign users, cannot enumerate all names to insert malicious mappings into the system. Due to the high entropy in cryptographic identifiers, the names are globally unique but on the other hand not memorable for humans. An implementation of such a system is Tor's *.onion* naming system, described in [Pro12]. This system is used withing the Tor anonymization system<sup>8</sup>. It is used to address services within the Tor network, the so called *hidden services*. Tor *.onion* names use the base32 encoding of the SHA-1 hash of the cryptographic RSA-1024 public key of the system. This approach results in names with 16 characters containing a random sequence of the digits [2..7] and the letters [a..z]. This results in addresses like

`http://suw74isz7wqzpmgu.onion/`

users can use to access Tor hidden services.

### 6.4.4 Making Cryptographic Identifiers Memorable

Names based on cryptographic identifiers are hard or even impossible to memorize for humans, as we saw with the example in the previous section, counteracting the fundamental idea of name systems to provide mappings from names memorable for humans to addresses usable by machines so humans do not have to memorize these names. Systems based on cryptographic identifiers can try to achieve the missing property of memorable names.

Multiple approaches have been thought of to make the names memorable for humans, as we see with existing finger print encoding systems like [McD94] or [HMNS98]. [McD94] suggests a system to convert 128-bit public keys into a sequence of 12 words based on a dictionary of 2048 English words. So a 128-bit key

CCAC 2AED 5910 56BE 4F90 FD44 1C53 4766

would become

RASH BUSH MILK LOOK BAD BRIM AVID GAFF BAIT ROT POD LOVE

A similar approach exists to make IPv4 addresses more memorable with the *Four Little Words* system, translating any IPv4 in a combination of four words. When translating an IPv4 address from and to the Four Little Words system, every octet of the IPv4 address is translated to a word with lookup in predefined in the lookup table in [HMNS98]. So it is possible to translate the IP address 65.49.90.35, belonging to the website of the U.S. Whitehouse, to the words *TEN OWL BLUR LAM*.

The Tor project proposes the *mnemonic .url system* as described in [Sai12], to make Tor *.onion* addresses memorable to humans by translating the 80-bit hashes into human-memorable sentences based on template sentences and dictionaries. The mnemonic *.url* system defines a set of requirements to overcome the limitations of simple finger printing systems to create sentences (contrary to just a series) of words with a semantic and tries

---

<sup>8</sup> <https://www.torproject.org/>

to prevent spelling errors and support non-native speakers by using simple and common words.

An issue with this kind of systems is that they try to make names more memorable using human language or at least words borrowed from languages, but resulting names are often still hard to understand and memorize. So is it possible to represent the IPv4 address 65.49.90.35 with the four words *TEN OWL BLUR LAM*, but the result is still hard to memorize since the combination lacks any semantics. This approach may be usable with IPv4 addresses translated to four words, but when using this approach with IPv6 addresses, users already have to memorize at least eight words. In addition, this causes internationalization and localization problems for these approaches, since the original schemes are hard to use for non-native speakers and hard to translate to other languages. Even when internationalization is paid respect in the design, most systems and grammatical rules if used are based on ideas stemming from Indo-European languages, representing only a small fraction of the world's population.

#### 6.4.5 Petname Systems

A secure name system with memorable names can be achieved based on the principle of *petname systems*. In a petname system, as described in [Sti05], names are not globally unique but only valid in the local context of the current user. In a petname system each user can freely assign memorable names to values and use these names to resolve names to values to access information and services. A petname system is considered to be secure since and every user assigns names to values locally and no centralized instance is required an adversary can attack. Since an adversary does not attack the users' end systems, as defined in Section 6.3.1, the adversary cannot prevent name resolution and modify or remove local mappings. Since a user can freely choose names from the namespace available, he will choose names which he can easily memorize. Since name resolution is performed locally, the system is still secure with an adversary able to enumerate all possible names in the namespace. With a petname system, names are not globally unique. So every user can freely assign names locally within his name space and no coordination between users or the existence of a centralized registrar is required. But names assigned by a user are only valid for a particular user. A second user cannot resolve the users names, so linking information and forwarding addresses based on local names is not possible with a petname system without additional efforts. A prominent example for a petname system is the "*hosts*" file, a predecessor of today's DNS described in [HSF85], available on many operation systems. In this file users can freely map names to IP addresses only valid locally on their system and use this file to resolve their local mappings. Another example is instant messaging software. In instant messaging software, users are identified with a globally unique identifier. On first contact, users can add the new contact to their contact list and assign a petname to the contact to later identify this contact. This petname is only valid for the local user.

#### 6.4.6 Linking Local Namespaces

Petname systems are useful when names are only used in local context, but on today's Internet and in particular the Web, linking information scattered over different systems and exchanging links to information with other persons are the ideas which prepared the ground for the breakthrough of the Web and the Internet. To allow linking and forwarding links to information, petname systems try to attenuate the missing property of global unique names. To allow users to resolve names of other users not in the local context,

petname systems can combine their approach of local namespaces with the idea of linking local names using *delegation*. With delegation, users give control over a portion of the local namespace to other users and names in this delegated portion are resolved in the respective user's context. Here petname systems can employ the idea of SDSI/SPKI, previously introduced in Section 6.2.3.

Petname systems can strongly benefit from ideas provided by SDSI/SPKI when trying to make name globally accessible. A petname system can use globally unique cryptographic identifiers to make users globally unique. In addition these cryptographic identifiers can be based on a cryptographic public/private key pair. With this approach, a user can cryptographically bind names to values and other users can verify these bindings by checking the certificate with the user's public key. By using the idea of linked local namespaces, a dissected namespace graph can be created by delegating control over names in a local namespace to other users. With this approach the issue of linking and forwarding links can be solved, since with delegation a user can refer to another user's mappings and refer to this user's namespace. Since this approach does not depend on a centralized security infrastructure, like X.509, this approach is particularly suitable with respect to the adversary used in this work and for the use in decentralized peer-to-peer systems.

## 6.5 Practical Considerations

As a preparatory work, we evaluated in Section 6.3 the adversary and possible attacks an alternative censorship-resistant has to withstand and the functional requirements for an alternative name system has to provide. The previous section has outlined the possible design space for a name systems a possible design for an alternative name system can rely on with respect to the given adversary. However, before we can elaborate the design for GNS, we have to elaborate and discuss technical and practical considerations an alternative name system to be used on the Internet and to integrate with existing infrastructure. This in particular important to allow seamless transition to an alternative system and convenient user adaption and facilitate coexistence and integration with existing systems and application on the Internet.

### 6.5.1 Interoperability with DNS

To be accepted by users, a censorship-resistant name system should respect users' 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 names and clients 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 support and return standard DNS records (such as A [Moc87b] or AAAA [THKS03]) to 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.

Interoperability and using multiple name systems with the same configuration can be easily achieved by integrating the different namespaces into one (virtual) holistic namespace. One possibility how this can be achieved is with the use of DNS pseudo Top Level Domains (pTLDs) as described in [CK13] with respect to special-use domain names. A pseudo Top Level Domain (pTLD) is a top level domain that is not actually participating in the official DNS and is not resolved by DNS servers. Possible existing examples for such pTLDs are “test.” or “localhost.”. So by using the pTLDs “key.”, a user might specify “ID.key.” to access a name system based on cryptographic identifiers, or “NICK.pet.” to access a pTLDs “pet.” for petnames. Naturally, this only works as long as the names chosen for the pTLDs are not used by the global DNS.

Once pTLDs have been selected for an alternative system, an implementation of local DNS stub resolver can be configured (for example, using the Name Service Switch [Fou]) to apply special resolution logic for names in the pTLDs. 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.

### 6.5.2 End-to-End Security and Error Handling

Today, client systems typically only include a DNS stub resolver, delegating the name resolution process to a DNS resolver operated by their ISP. As ISPs according to our adversary model 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.

Achieving end-to-end security for name resolution may not only require updating operating system resolvers but also name resolution APIs and to a certain degree applications. So 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.

### **6.5.3 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 SSL/TLS 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, names provided by a petname system names involving delegation from a SDSI/SPKI-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/SPKI-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.

### **6.5.4 Censorship-Resistant Lookup**

Censorship-resistant distributed name systems not using any trusted third parties 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 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  $R^5N$  [EG11] 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 [MDD02] by building the overlay topology using existing social relationships.

### **6.5.5 Privacy-Preserving Name Resolution**

An additional requirement when designing a resilient name system and especially when using in this context a decentralized storage like a DHT is the users' privacy. In existing centralized name systems, infrastructure providers can easily observe which names are used by which users especially with existing DNS and DNSSEC providing no confidentiality. When the database is decentralized in a DHT, these central observation points are eliminated; however, now ordinary users (and of course adversaries) can observe other users' 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. Naturally, using anonymization techniques like Tor [DMS04] may further increase latency.

## 6.6 Design of the GNU Name System

To overcome the challenges and issues resulting from the design of current DNS described in 6.2 and to provide a resilient alternative, we present in this section the design of GNS, a fully decentralized, censorship-resistant and privacy-preserving name system, designed as an alternative to DNS providing a generic mechanism to map names to arbitrary values. Due to its design, it can also double as a partial replacement of current public key infrastructures, such as X.509. GNS is fully decentralized, so it does not require any centralized or trusted instances or registrars like in current DNS. It is censorship-resistant since it does not rely on centralized instances and names do not have owners. Instead every user manages his own namespace and can freely assign names to values. To make name resolution resilient, GNS and the implementation provided relies on the communication infrastructure described previously in this work and we chose a DHT resilient and resistant to malicious attacks to realize a decentralized storage. GNS is privacy-preserving since all information published and resolved with GNS are encrypted, preventing an attacker to analyze published or resolved data by intercepting network or peer-to-peer traffic. The design of GNS incorporates the capability to integrate and coexist with DNS.

With GNS, we propose a system in line with Richard Stallman stating in his essay “Why Software Should Not Have Owners“:

*When a program has an owner, the users lose freedom to control part of their own lives. [Sta02]*

and Pierre-Joseph Proudhon saying

*If I were asked to answer the following question: What is slavery? and I should answer in one word, It is murder, my meaning would be understood at once. No extended argument would be required to show that the power to take from a man his thought, his will, his personality, is a power of life and death; and that to enslave a man is to kill him. Why, then, to this other question: What is property! may I not likewise answer, It is robbery, without the certainty of being misunderstood; the second proposition being no other than a transformation of the first? [Pro40]*

The foundation of the GNS system is a petname system, as described in Section 6.4.5, where each individual user is provided with his own local namespace. The user can use this namespace to provide and resolve name/values mappings. By using a petname system for GNS, the user has full control over his namespace and the mappings in this namespace. With a petname system, names do not have owners and cannot be owned: in his local namespace the user can freely choose which names he assigns to a value. Since names are only valid in the local namespace, there are no conflicts with names assigned by other users: a name is always used with reference to the namespace it belongs. Therefore, every

user can freely create mappings and assign names to values. Giving users the possibility to freely choose names corresponds to the possibility of freedom of thought and expression with natural languages. Restricting and controlling the use of language can be used to control people and suppress free expression, lead to censorship and ultimately to an Orwellian, totalitarian tyranny [Kle47] as George Orwell stated in his essay “Politics and the English Language” [Orw46]:

*But if thought corrupts language, language can also corrupt thought.*

But with a petname these names are only valid in the local namespace and can therefore only be resolved by the local user. Since linking between information and resources is a key concept of the Web, the second central idea of GNS is to provide users with the ability to resolve names of other users by linking local namespaces as described in Section 6.4.6 with petnames system putting effort in making secure and memorable names global. This linking of local namespaces is achieved by allowing users to securely delegate control over a label in the local namespace to other users. Names under these delegated labels are then resolved in the namespace of the responsible user.

With the combination of a petname system giving users full control over their namespace and delegation allowing users to resolve other users’ names, GNS is fully decentralized and does not require nor depend on a centralized or trusted authority, a design property making the system robust against censorship attempts. Decentralization and additional censorship-resistance is achieved by using a DHT to enable the distribution and resolution of key-value mappings. Depending on the properties of the DHT in question, good censorship-resistance can be achieved. Authenticity of records is ensured since data published in the DHT are cryptographically encrypted and signed and can be verified during the resolution process. Finally, GNS is privacy-preserving since mappings when published in the DHT and responses for name resolutions 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.

To introduce a new relation with other users and for applications relying on globally unique names, GNS provides in addition to relative names the use of absolute identifiers based on globally unique cryptographic identifiers as described in Section 6.4.3. These absolute identifiers provide globally unique and secure names which are not memorable for humans and can be used to refer to namespaces no prior relation was established with.

GNS is developed as an alternative to DNS, but was also designed with a focus on coexistence with DNS. Labels in GNS have the same syntax as in DNS and names follow the same syntax as names in DNS: in GNS names are labels separated by dots as in DNS. Using the same syntax for both labels and names, allows a simplified integration of the hierarchical DNS namespace and the relative and absolute identifiers and of GNS.

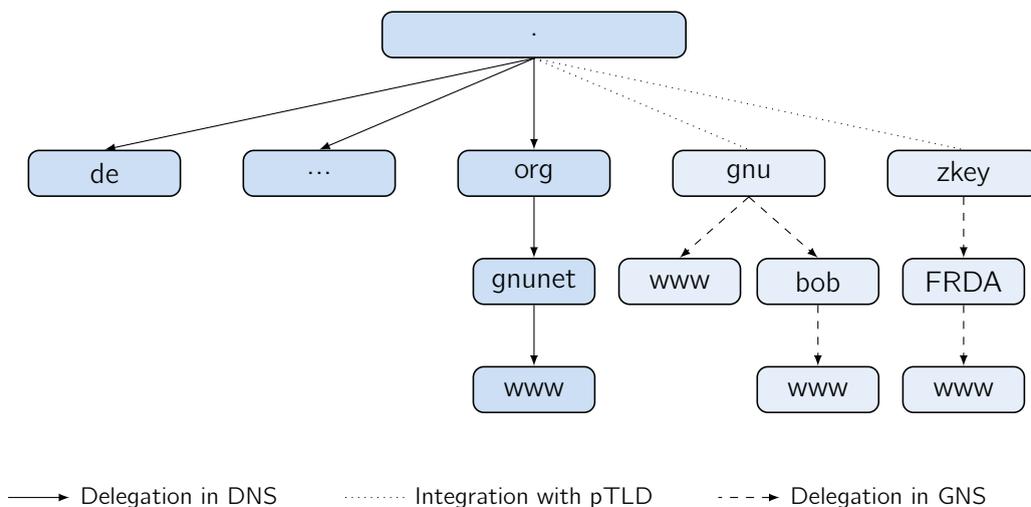
An additional focus with the design of GNS is the integration with existing, deployed technologies on the Internet and especially integration with the Web. Special focus was put on existing HTTP(S) infrastructures and to provide the mechanisms needed to smoothly integrate GNS with existing processes and procedures in Web browsers. Specifically, we show how GNS is able to transparently support many assumptions that the existing HTTP(S) infrastructure makes about globally unique names.

### 6.6.1 Names, Zones and Delegations

GNS employs the same notion of names as SDSI/SPKI, described in Section 6.2.3: principals are globally unique cryptographic public keys. Principals serve as namespaces with

names only valid in the local namespace defined by the principal. In this namespace defined by the principal, a user can freely assign local identifiers. These namespaces constitute the *zones* in GNS. In GNS, a zone is a public-private key pair and a set of *records* with labels assigned. In GNS, records consist of a label, type, value and expiration time. Labels of records are equivalent to local identifiers in SDSI/SPKI and have the same syntax as in DNS. With the possibility to link local namespaces using delegation, names in GNS consist of a sequence of labels, which identifies a delegation path in a directed graph, with the start of the path always being the local zone.

According to the principle of a petname system, in GNS each user manages his own zones. Each user can manage a set of zones, but particularly important is his designated personal *master zone*. Each user uses this master zone as his starting point for lookups in the directed graph in lieu of the root zone from DNS. For interoperability with DNS, 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. The special use of the “.gnu” pTLD for the purpose of GNS is publicly documented in [GWWA14] with respect to requirements defined for special-use domain names in [CK13]. This integration is depicted in Figure 6.5.



**Fig. 6.5:** Integration of GNS and DNS Namespaces

Users can freely manage mappings for memorable names in their zones and, using a special record type, delegate control over a subdomain to any other zone. Publicizing delegations allows transitive resolution by following delegation chains. The records of a zone are stored in a database on a machine under the control of the zone’s owner. Records can be flagged as private or public, and public records are made available to other users by publishing these records in the DHT. Users can specify for each zone a preferred *nickname* to other users. This nickname is used as a suggestion for a label when another users adds a delegation to this zone. This nickname functionality is realized using a special NICK record, containing the desired label according to the restrictions for label in GNS. This NICK record is stored with the special label “+” indicating in GNS the root entry of the zone and is automatically added to every set of records in the zone. Record validity in GNS is established using signatures and controlled using expiration values.

### 6.6.2 Zone Management with Nicknames and Petnames

We now explain how the actual management of names and zones 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 revoke her GNS zone in case her master zone key 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 all labels in “\*.carol.gnu” 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 ideally only not already be in use within Alice’s social group.

### 6.6.3 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 web page a link to Alice’s web page. For Bob, Alice’s website is “www.carol.gnu”. For Dave, Bob’s website is “www.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 as defined in [BLFM05], except that it works with hostnames.

### 6.6.4 Censorship-Resistant and Privacy-Preserving Publication and Name Resolution

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 increase the resilience with respect to availability and resilience of the name resolution process, it is beneficial to choose a DHT implementation focusing on resilience and resistance as described in Section 6.2.4. Here different types of DHTs exist with different features useful in the context of GNS, supporting redundancy by storing the value on multiple nodes at the

same time, DHTs being resilient to attacks and supporting restricted route networks. When using a DHT, it is required to refresh the information stored in the DHT in regular interval to prevent data from expiring and being removed and to prevent information from getting lost due to peers storing the respective information leaving the network. Details about the publication process is covered by Section 6.8 describing the implementation of GNS. To improve name resolution performance, it is beneficial to provide extensive caching of DHT results to reduce repeated lookup operations in the DHT. Our GNS implementation supports extensive caching using a dedicated caching component described in Section 6.8.6.

To resolve a name with GNS, the GNS has to follow the delegation path indicated by the GNS name. The resolver starts in the local master zone, and iteratively resolves every label contained in the name. If the name is found in the current zone, the resolver returns the records stored under this label and stops. If the solver finds a label indicating a delegation to another zone (so with a PKEY record returned as a result), the resolver continues to resolve the next label in the delegation path in this zone. If a delegation to legacy DNS is found (with a GNS2DNS record and additional A or AAAA records indicating the DNS name server to use as described in Section 6.6.7), the resolver synthesizes the name to resolve in GNS based on the remainder of the delegation path and resolves this synthesized name with DNS using the nameserver indicated in the result.

To maximize user privacy when using the DHT to look up records, both queries and replies are encrypted. The user publishing records in a DHT stores the encrypted information under a key in the DHT derived from the public key of the zone and the label to be resolved. The block stored in the DHT is encrypted using a symmetric encryption key derived from the identifier (public key) of the zone and the label. To resolve a name in a zone, the user can calculate the respective key to use for the DHT lookup based on the public key of the zone and the label to lookup and in addition derive the symmetric encryption key to decrypt the record block obtained as a result from the DHT. The full cryptographic protocol used with GNS is described with the implementation in Section 6.8.8.

For GNS, we rely on ECC cryptography using the Elliptic Curve Digital Signature Algorithm (ECDSA) signature scheme and Curve25519 [Ber06]. The cryptographic algorithms used with GNS are described with the implementation in Section 6.8.2.

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 and  $n := |G|$ . Let  $l \in \mathbb{Z}_n$  be a numeric representation of the label of a set of records  $R_{l,P}$ .

Using

$$h := x \cdot l \pmod n \quad (6.1)$$

$$j := \text{HKDF}(l, P) \quad (6.2)$$

$$Q_{l,P} := H(hG) \quad (6.3)$$

$$B_{l,P} := S_h(E_j(R_{l,P})), hG \quad (6.4)$$

we can then publish  $B_{l,P}$  under  $Q_{l,P}$  in the DHT, where  $H$  is a hash function,  $S_h$  represents signing with the private key  $h$ , HKDF is a hash-based key derivation function [KE10] and  $E$  represents symmetric encryption based on the derived key  $j$ .

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

$$IP = I \times G = hG$$

$$\rightarrow Q_{I,P} = H(IP) = H(I \times G) = H(hG)$$

**Theorem 1.** *An adversary monitoring GNS traffic and intercepting a DHT key  $Q_{I,P}$  and a record block stored under  $Q_{I,P}$  containing the signed and encrypted set of records  $S_h(E_j(R_{I,P}))$ , the public key  $hG$  and the block expiration time  $t$  but not knowing the label  $I$  and the public key  $P$  can only observe the size of the set of records and the expiration time of the record block.*

*Proof.* The adversary used in this work cannot break cryptographic primitives and ECDHE as defined in Section 6.3.1. From monitoring GNS traffic, the adversary can learn the DHT key  $Q_{I,P}$  used to store records, the derived public key  $hG$ , and the signature created with  $S_h$ .

To decrypt the encrypted records, the adversary needs to derive the symmetric encryption key  $j = \text{HKDF}(I, P)$ . This is not possible without knowing  $I$  and  $P$  due to the properties of [KE10]. Due to the properties of Elliptic Curve Diffie Hellman (ECDH), it is not possible to derive  $I$  or  $P$  from the derived public key  $hG$  or the DHT key  $Q_{I,P}$ .  $\square$

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. Users could in addition 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.

### 6.6.5 Automatic Shortening

With delegation and transitivity of names, one problem arising is the challenge of long delegation chains. Long delegation chains can lead to poor system performance when for a name lookup many different zones have to be queried and on the other hand decrease the resilience of the overall system, if a single element in the delegation path fails to be resolved. Therefore, we provide with GNS the idea of automatic shortening aiming to automatically decrease the length of delegation paths. With automatic shortening, information about zones obtained with name resolution (PKEY records) and not yet known in the current zone of the user, are automatically introduced into the current zone or a dedicated *shorten* zone (based on a NICK record if provided).

With respect to the example for relative names in the previous section, once Dave's client translates "www.carol.+" to "www.carol.buddy.gnu", Dave can resolve the name "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 available—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 to Dave 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. As a consequence resilience and performance of the resolution process is improved. Shortening is a variation of trust on first use (TOFU), as compromising Bob afterwards would no longer compromise Dave's path to Alice.

### 6.6.6 Absolute Names in GNS

In GNS, the ".gnu" pTLD is used to provide secure and memorable names which are only defined relative to some master zone. To introduce new zones into the system, it is ultimately required to be able to reference a zone by an absolute identifier, which must correspond to the public key of the zone. To be able to refer to zones, GNS supports in addition to names relative to zones, absolute name based on the public key of the zone as an identifier. To facilitate dealing with public keys directly, GNS uses the pseudo-TLD ".zkey", which indicates that the specified name contains the public key of a GNS zone. As a result, the ".zkey" pTLD allows users to use secure and globally unique identifiers. Applications can use the ".zkey" pTLD to generate a name for a GNS zone for which the user does not (yet) have a memorable name.

A label in the ".zkey" pTLD must be the Crockford Base32 [Cro, Jos06] encoded public key of a zone since with this encoding a public key can be represented within the 63 character limit imposed by DNS for labels. Based on the implementation of GNS as described in Section 6.8, 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" pTLD 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" pTLD. The special use of the ".zkey" pTLD for the purpose of GNS is publicly documented in [GWVA14] with respect to requirements as defined for special-use domain names in [CK13].

With absolute names it is possible to exchange information like links to websites between users or third parties where no delegation relationship exists. When a user contacts another user, he can send him a link containing an absolute link and the user will be able to access this link. With automatic shortening described in Section 6.6.5 enabled, a delegation to the zone contained in the absolute name will be automatically created in the respective shortenzone.

### 6.6.7 Delegation to Legacy Name Systems

To counteract censorship attempts in DNS, GNS provides the functionality to delegate control over a label in GNS to DNS and a DNS server specified. Names under this label will then be resolved using traditional DNS. This can be used when a particular DNS subdomain was censored in DNS. Here GNS offers the possibility to still access this DNS subdomain from GNS.

To delegate from GNS to DNS, GNS-specific GNS2DNS records are used. These records are semantically similar to NS records in DNS, delegating control over a subdomain in DNS to specific name server(s). A GNS2DNS record specifies the DNS subdomain and the authoritative DNS resolver to use to resolve the given domain.

If we assume, since "example.com" was blocked by Alice's ISP for some reason, she configured a delegation for this service in her GNS zone. She added a GNS2DNS for "example.gnu", delegating this label to "example.com" using "example.com"'s DNS servers a.iana-servers.net or b.iana-servers.net. When Alice tries to access "www.example.gnu", GNS finds the delegation to DNS and synthesizes for the label "www" the name new

## 174 6. GNS - A Decentralized, Privacy-Preserving and Censorship-Resistant Name System

name “www.example.com” and start over to resolve this synthesized name, by querying the DNS resolver included in the delegation.

Delegation back to legacy systems does not provide any of the security and privacy properties provided by GNS. Users can be put at risk since they assume to be protected by the GNS and based on the name given, it is not evident that during resolution name systems are switched to insecure DNS. Therefore, the use of GNS2DNS has to be exercised with caution. On the one hand the possibility of delegating control could be disabled by default and has to be enabled by default or GNS-to-DNS delegation could be restricted to a dedicated GNS zone. The latter approach only protects a local user and not users resolving a delegation path.

### 6.6.8 Handling TLSA and SRV Records in GNS

With respect to records provided by DNS, TLSA records are of particular interest for GNS, as they allow TLS applications to use DNSSEC as an alternative to CA-based the X.509 Public Key Infrastructure (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.

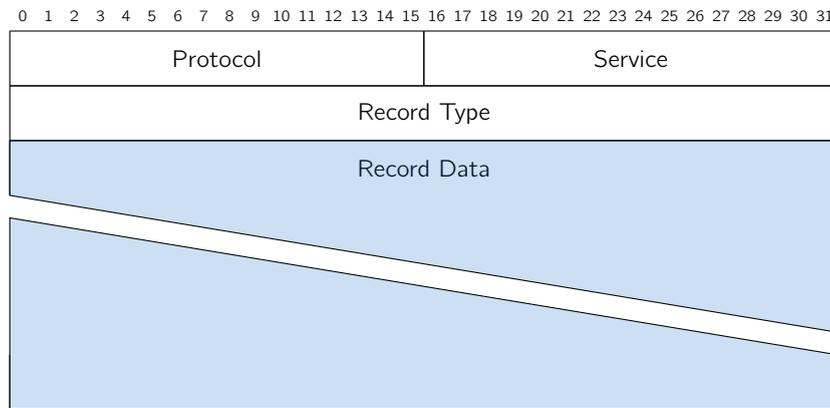
However, to support TLSA in GNS, a peculiar hurdle needs to be resolved. In DNS, both TLSA and SRV records are special in that their domain names are used to encode the service and protocol to which the record applies as defined with the [HS12]. For example, a TLSA record for HTTPS (port 443) on `www.example.com`. would be stored under the domain name `_443._tcp.www.example.com.`. The same issue applies for SRV records, so the SRV record for “example.com”’s SIP server reachable with TCP would be stored under `_sip._tcp.example.com.` in DNS.

In GNS, this way structuring names is a problem since dots in GNS names are supposed to always correspond to delegations to another zone. Furthermore even if a special rule would be 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 likely less secure).

GNS solves this problem by introducing an additional GNS-specific record type, the BOX record. A BOX record contains a 16-bit port protocol identifier (6 = TCP, 17 = UDP, etc), a 16-bit service identifier (port number), a 32-bit embedded record type (so far always SRV or TLSA) and the embedded record value as depicted in Figure 6.6. This way, BOX records can be stored directly under `www.example.gnu` 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 BOX record during the resolution process. Thus, in combination with a powerful user interface to manage these records, GNS effectively hides the existence of BOX records from DNS users.

### 6.6.9 Records in GNS

GNS is designed to provide a generic and extensible name/value mapping mechanism. So GNS supports resource record types used with DNS as described in Section 6.6.9.2 and



**Fig. 6.6:** Network Format of GNS BOX Records

resource record types specific to GNS as described in Section 6.6.9.1. GNS is designed to support additional (application-specific) record types to ensure extensibility with new record types in the future.

With GNS, records (locally, not published to the DHT) consist of a label assigned to the record and contain information about the record type, expiration time of the record, a set of flags, the length of the data contained in the record and a variable length data field. The label itself is not stored in the record but in GNS, a record is stored under a label in the respective zone. For the type field, GNS uses a 4 octet field, so providing a maximum of  $2^{32}$  record types, ensuring compatibility with existing DNS supporting a maximum number of  $2^{16}$  record types. Legacy DNS record types use their record types with record type numbers  $< 2^{16}$  and additional records types introduced with GNS have a record type numbers  $\geq 2^{16}$  to avoid conflicts with DNS record types that might be introduced in the future.

GNS records contain an 8-octet expiration field defining how long the record is valid. Locally, the user can specify an absolute or a relative record lifetime. With an absolute expiration time, a record can expire on a specific date (e.g. Jan 1 2020), while with a relative expiration time, a record can be valid for a certain period of time (e.g. 1 day). This approach is useful for mappings frequently changing, e.g. with dynamic addresses. When records are published to the DHT, relative expiration times are converted to absolute expiration times.

Records provide an additional field to store additional information and options related to this record. GNS distinguishes between local flags and flags also published in the DHT. Local flags supported by GNS are the *private* flag indicating that a record is a private record and must not be published in the DHT and the *relative expiration* flag indicating that the expiration time in the expiration time field is a relative value. The *shadow* flag is a flag published in the DHT to indicate that a record is a shadow record as described in Section 6.6.10.

When records get published to the DHT, as described in detail in Section 6.6.4, the records are serialized to a *record block*, containing all records assigned to a label in a zone (plus a *NICK* record if existing as described in Section 6.6.2). During this process local flags are removed, records marked as private are excluded, and relative expiration times are converted in absolute expiration times.

### 6.6.9.1 GNS-Specific Record Types

GNS itself introduces several new records required for its options:

**PKEY for delegation:** PKEY records securely delegate control over a label to another zone as described in Section 6.6.1. PKEY records contain the public key of the zone control is delegated to. The ECC Curve25519 public key is encoded in a format suitable for network transmission and signatures. GNS uses the Ed25519 standard compact format as described in [BDL+11] with a key having a size of 32 byte. Repeated delegation allows GNS to achieve transitivity of names. Secure delegation using PKEY records is central to GNS and 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 as described in Section 6.6.1. The value of the record consists of a label with the 63-character limit from DNS. If a nickname is specified for a zone, the same NICK record is added under *each* label of the respective zone when published; this ensures that the nickname is part of every response and thus no additional lookup is required to obtain the nickname, for example during automatic shortening.

**GNS2DNS:** GNS2DNS records delegate resolution for a label from GNS to DNS. Similar to NS records in DNS, the value in the GNS2DNS record is the name of the subdomain in DNS and the authoritative DNS server to use to resolve the subdomain. For example:

Name	RR Type	Value	Description
Q: www.example.gnu	A		
A: example.gnu	GNS2DNS	example.com a.iana-servers.net	DNS domain DNS server
Resolved in DNS:			
Q: www.example.com	A		
A: www.example.com	A	93.184.216.119	

Given the first response received, 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 a.iana-servers.net 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 DNS resolution process.

**BOX:** records are used in GNS to support the name structure required by DNS’ TLSA and SRV records in GNS as described in Section 6.6.8 and TLSA or SRV value. BOX records contain a 16-bit protocol, a 16-bit port number and a 32-bit field to specify the record type contained in the BOX record (e.g. TLSA or SRV) as depicted in Figure 6.6. BOX records are stored under the label in the DHT and when a lookup for SRV or TLSA records are performed, the lookup for the respective BOX record is synthesized.

For example:

Name	RR Type	Value	Description
Q: _443._tcp.www.example.gnu	TLSA		
Translated by GNS to :			
Q: www.example.gnu	BOX		
A: www.example.gnu	BOX	6	Protocol
		443	Port
		TLSA	RR Type
		TLSA RDATA	Value
Translated by GNS to :			
A: _443._tcp.www.example.gnu	TLSA	TLSA RDATA	See [HS12]

**VPN:** this record type is used with GUNet's VPN service to establish a redirection. VPN provides protocol translation and allows to tunnel IP traffic over GUNet<sup>9</sup>. A VPN record contains information about the peer to establish the tunnel with, the protocol to use (TCP or UDP) and a service name. When a connection with GUNet VPN was successful, as a result an A or AAAA is returned depending which type was requested.

Name	RR Type	Value	Description
Q: www.example.gnu	A		
A: www.example.gnu	VPN	ABCD	Peer ID
		tcp	Protocol
		srv_xyz	Service Name
A: www.example.gnu	A	192.0.2.3	

**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	Description
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.

### 6.6.9.2 DNS Record Types

As GNS is designed to allow integration and coexistence with DNS, most DNS resource records from [Moc87b, THKS03] (e.g., A, MX) are used with identical semantics and binary format in GNS. DNS resource records supported when this document was written are A, CNAME, SOA, PTR, CERT, MX, TXT, AAAA, SRV, TLSA.

For DNS, the format of resource records is defined in [Moc87b]: In DNS resource records consist of a variable length name field, a two octets TYPE field specifying the

<sup>9</sup> <https://gnunet.org/gnunet-vpn>

## 178 6. GNS - A Decentralized, Privacy-Preserving and Censorship-Resistant Name System

kind of record by containing one of the RR TYPE codes, a TTL field defining validity of the record, a CLASS field, a field containing the length of record data and variable length record data. With the type field  $2^{16}$  different record types can be defined with DNS.

One exception to DNS records supported in GNS are NS records, which we replace with GNS2DNS records as described in Section 6.6.7, used to delegate control for a name back to legacy DNS and minor semantic modifications to support relative names in record values.

GNS slightly modifies the rules for some existing record types in DNS when using these types in GNS. 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 [Moc87b], 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.

### 6.6.10 Shadow Records

A common use case scenario in current DNS is to use DNS to migrate between systems providing a service. Due to DNS servers caching name resolution results, it is not possible to switch between systems by just modifying the value of the record to point to a new system and switch off the old system. With DNS, migration is based on record lifetimes, reducing the lifetime of the record pointing to the old value and adding a new record pointing to the new system. This approach enables in theory a smooth transition between systems facilitating DNS caching approaches.

GNS supports a similar approach. In GNS, records can be marked as *shadow records*. A receiver only interprets these 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.

### 6.6.11 Revocation in GNS

In case a zone's private key is lost or compromised, it may be 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 revocation lists in X.509 have similar disadvantages in terms of bandwidth, latency increase and reduced privacy.

Instead of these traditional methods, GNS takes advantage of the peer-to-peer overlay below the DHT to distribute revocation information by flooding the overlay. 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-yet-Byzantine fault-tolerant algorithm for flooding in the peer-to-peer overlay could be used for denial of service attacks. Thus, to ensure that peers cannot abuse this mechanism, GNS requires that valid revocations include a revocation-specific proof of work. This proof is based on computing a script key derivation, which was designed to be computationally expensive to prevent hardware attacks as described in [PJ13]. 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 [EGUV11] realized in GUNet's SET service. 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 practices in the X.509 PKI today described in Section 2.5, where blocking of access to certificate revocation lists is an easy way for an adversary to render revocations ineffective [Ltd13]. This has forced vendors to include lists of revoked certificates with software updates [Lan12]. With CRLs, the client has for every operation to check the CRL if the certificate is revoked. This is expensive if the client obtains the CRL from the CRL operator frequently or unreliable if it does not use a recent version of the CRL. OCSP is also expensive since with OCSP the certificate issuer has to provide an OCSP responder cryptographically signing the response for every request. With OCSP stapling, the revocation certificate is provided by the owner of the certificate and not issued for every request. This approach improves performance and reduces the cost for a revocation check, but the issuer still has to provide an infrastructure to issue and sign the OCSP responses. With GNS' approach for revocation, revocation checks are performed locally without communicating with other participants. To issue a revocation, the required proof has to be calculated only once and can be used when a zone has to be actually revoked. Network communication to distribute revocation information is designed to be efficient using the SET service and efficient flooding where information is dismissed if invalid.

### 6.6.12 Dealing with Legacy Assumptions: Virtual Hosting and TLS

In order to integrate smoothly on application level 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 LEgacy H0stname 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, as described in Section 6.6.9.2, and thus using TLSA records instead of CAs would be a better alternative once browsers support it.

## 6.7 Security Analysis

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 [Ber]. 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.

We will now 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 [EG11]. For revocation, the flooding mechanism is even harder to disrupt. The adversary would have to be able to censor 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 [Ltd13].

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, even if the attacker succeeds the damage will be limited to users that resolve via the compromised zone, limiting damage to users that use that zone's mappings. If a malicious user modifies an established mapping to a rogue mapping, the user can still change the origin for a mapping using an *agile* approach in whom they trust. For important mappings, it would still be possible to use a somehow more trusted source for mappings like a trusted registrar.

## 6.8 Implementation of GNS

In this section, we present the implementation of the GNS based on the design presented in the previous sections. We give an overview how identities and zones in GNS are managed and decentralized and privacy-preserving name resolution is realized. In addition, we describe how caching is realized with GNS to improve performance and explain GNS integration in the name resolution process of the operating system and additional tools employed to improve integration with established applications on the Internet.

### 6.8.1 Architecture

Our implementation of the GNS is realized in the GNUet peer-to-peer framework and relies on the building blocks provided by GNUet. GNUet provides the functionality to create a censorship-resistant peer-to-peer overlay network relying on the communication infrastructure described in Chapter 4 and  $R^5N$ , GNUet's implementation of a censorship-resistant DHT with a focus on restricted route networks.

An implementation of the design presented in the previous sections has to realize different functionalities required to provide a working implementation of GNS:

**Management of cryptographic information:** Zones in GNS are based on a cryptographic public/private key pairs. With GNS, a user can use more than just a single zone. To manage the cryptographic key pairs, we employ an IDENTITY service.

**Administration of local GNS zones:** the NAMESTORE component is used to provide a persistent storage of local GNS zone information and provides programmatic access to this information for other components and to manage local GNS zone information.

**Publication of local information and name resolution:** the GNS service realizes GNS name resolution functionality and publishes public GNS information in the DHT to allow other users to resolve this information.

**Integration with the name resolution process:** To allow applications to natively use GNS, we integrate GNS directly with the name resolution process of the operating system. In addition, we describe how applications using their own name resolution mechanism can be supported.

**Revocation of GNS information:** In case a GNS zone gets compromised, the zone has to be revoked and this information has to be distributed to other users. This is realized by GNS' REVOCATION service.

**Caching of GNS information:** To increase the overall performance of GNS, we provide extensive caching of GNS information obtained from the DHT. This caching is realized by the GNS' NAMECACHE service.

According to the design principles of GUNet to implement functionality in separate collaborating services, we employ the same design principle to implement GNS the most important components to realize with GNS are the IDENTITY service (managing cryptographic information), the NAMESTORE service (administration of zone information and persistent storage), the NAMECACHE service (caching of GNS information) and the GNS service (publication of GNS information and name resolution). The interaction of these services is depicted in Figure 6.7.

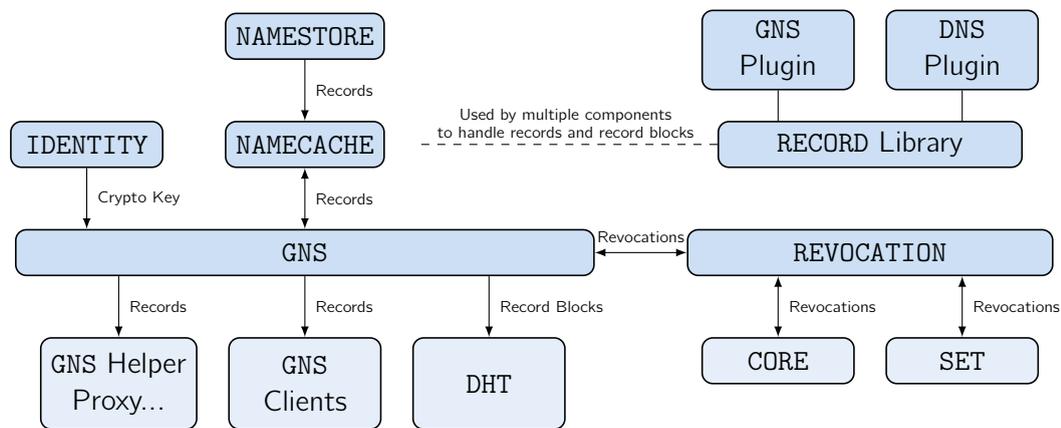


Fig. 6.7: GNS Components and Interaction

### 6.8.2 Cryptography Used in GNS

For our implementation of GNS, we rely on Elliptic Curve Cryptography (ECC) using Curve25519, and the ECDSA signature scheme, providing cryptographic authentication and integrity. With ECDSA, we benefit from the possibility to use ECDSA with key derivation functions to derive new cryptographic key material from existing cryptographic keys. For symmetric encryption of records in record blocks, GNS relies on both AES-256 and Twofish-256 [SKW<sup>+</sup>98]. We use SHA-512 to hash data when creating DHT keys.

### 6.8.3 Identity Management for GNS

As discussed with the design of GNS in Section 6.6.1, the basic principles of GNS are to provide each user with his own local namespace and to allow users to freely manage name/value mappings in their namespace. Namespaces in GNS are based on cryptographic public/private key pairs. A zone in GNS is a cryptographic public/private key pair and a set of records assigned to labels in this namespace. With GNS, a user is not restricted to a single zone but instead he can have multiple zones for different purposes (e.g. private mappings or automatic shortening). Every user has a designated *master* zone, used as the starting point for lookup operations mapped in GNS to the pTLD *.gnu*.

GNUet provides a convenient way to manage cryptographic identities (here called *egos*) and provide these identities to other GNUet services. Identities in GNUet are so called *egos*: *egos* are a cryptographic public/private key pairs with a human-memorable label assigned to refer to this ego. The label is an alphanumerical string with a maximum length of 63 chars. GNUet's IDENTITY subsystem provides the functionality to create and manage cryptographic identities and GNUet components can interact with IDENTITY to obtain this cryptographic key material by using the ego to refer to a specific key pair. The most important functionality provided by the IDENTITY service is to retrieve the public and the private key for existing *egos*, to add or delete new *egos*, and to assign *egos* as default identities to particular GNUet services. Interaction with the IDENTITY service is available using the IDENTITY CLI tool or for other components using the API provided.

With GNS, *egos* are used to refer to GNS zones using a human-memorable name. For every GNS zone, an ego is created with IDENTITY. The default ego assigned by the IDENTITY service to the GNS component is the ego of the designated GNS master zone used by GNS as a starting point for name resolutions. The respective GNS tools, like the `gnunet-namestore` command line tool used to manage zones as described in Section 6.8.5, or the `GNUNET-GNS` command line client, used to perform name resolutions, accept *egos* as an argument when a particular GNS zone has to be specified. These components interact with the IDENTITY service to obtain the required keys associated with this ego. Based on the IDENTITY service's functionality to assign default *egos* to services, the GNS service performs a lookup for the GNS master zones and the obtain associated key material.

When Alice sets up her GNS installation, she starts with creating her zones. Most importantly, she creates the master zone where she will store the records for the services she provides and used as start point for name lookups. To refer to this zone when creating records, she decides to assign the ego *masterzone* to this zone. In addition, she typically creates a zone for automatic shortening with the ego *shortenzone*.

#### 6.8.4 Records in GNS

To store resource records with GNS, GNS uses resource records similar to DNS. In GNS, records have a 32-bit record type, a 64-bit expiration time, a variable length data field, and a 32-bit field indicating the size of the value contained. The expiration time for a record can be defined as an absolute point in time (Jan 1 2020) or a relative point in time (1 week). In the 32-bit flag field, additional options for a record can be set:

**GNUNET\_RECORD\_RF\_PRIVATE:**

This is a private record. If this flag is set, the record will not be published in the DHT to make it available to other users.

**GNUNET\_RECORD\_RF\_RELATIVE\_EXPIRATION:**

The expiration time contained in the record has to be interpreted as relative time.

**GNUNET\_RECORD\_RF\_SHADOW\_RECORD:**

This record is a shadow record only to be used when all other records available have expired as described in Section 6.6.10. The shadow flag is the only flag published in the DHT.

The record itself does not include the label it is assigned to since in GNS records are stored *under* a label in a zone, so the record is assigned to a label. The structure

of a record is depicted in Figure 6.8 with the private flag denoted as P-bit, the relative expiration flag denoted as R-bit, and the shadow flag denoted as S-bit.

When records are published in the DHT, records are stored in *record blocks* containing all records assigned to a label. These record blocks have a maximum size of 63 KiB and are cryptographically signed using a ECDSA key derived from the zone’s private key and the label. The record payload is in addition encrypted with AES-256 and Twofish-256 using a symmetric key derived from the public key of the zone and the label. This signature and encryption process are described in detail in Section 6.8.8. A record block contains a 64-byte signature of the content, the 32-byte public key of the derived key, an 8-byte signature purpose and the 8-byte expiration time of the block followed by encrypted record data. For signed record blocks, the signature purpose with GNS is always GNUNET\_SIGNATURE\_PURPOSE\_GNS\_RECORD\_SIGN. When a record block for a label in a zone is created, all private records assigned to a label are left out so only public records get published in the DHT. In addition, relative expiration times in records are converted to absolute times. The structure and payload of a record block is depicted in Figure 6.9.

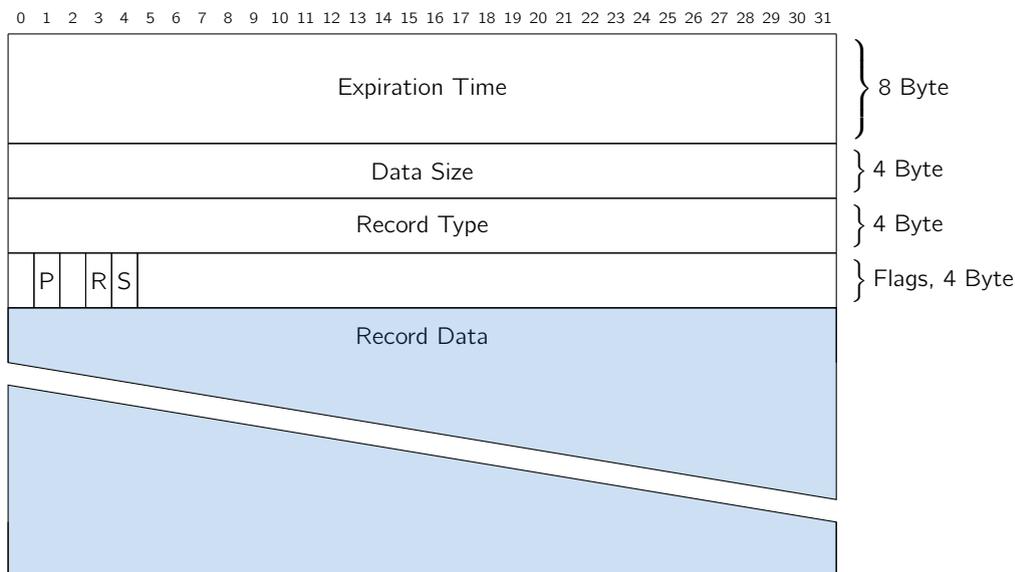


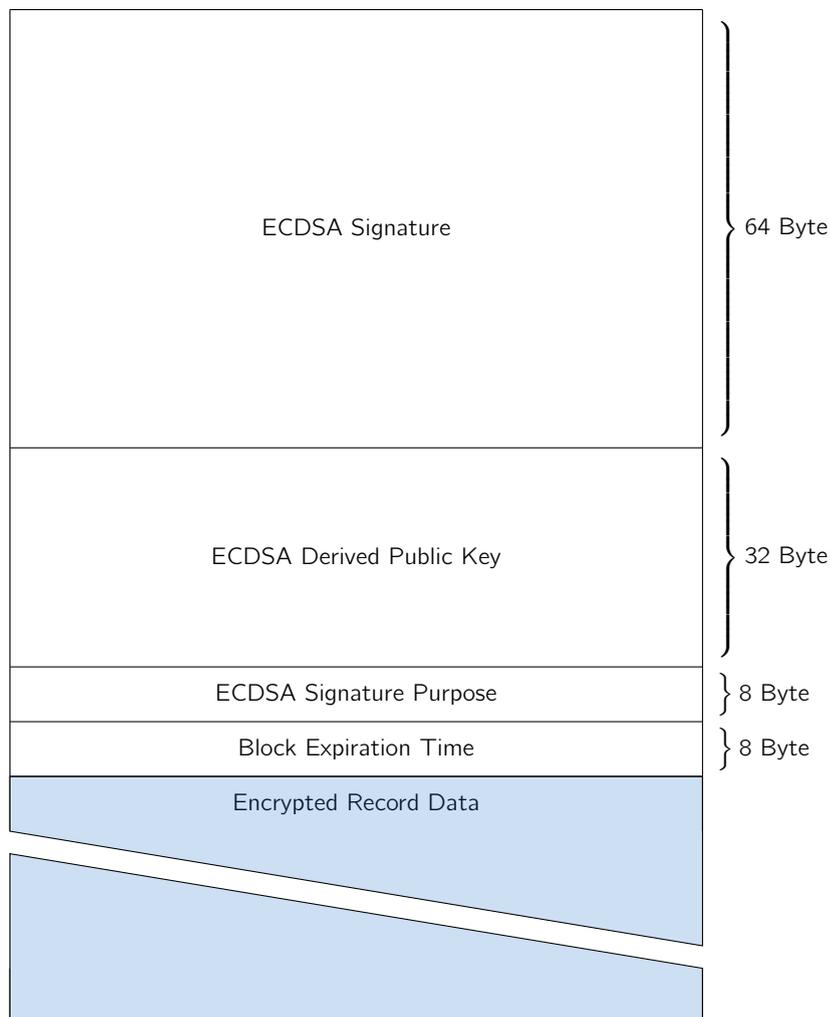
Fig. 6.8: Network Format of GNS Records

GNS supports an extensible architecture to support new application-specific record types. To support applications adding additional record types, GNS supports a plugin architecture with its RECORD library. So applications can add application specific record types by providing GNS with a loadable plugin providing the functionality to parse and process these records.

### 6.8.5 Managing GNS Zones and Persistent Storage

With GNS, a zone is cryptographic key pair and a set records assigned to labels in this zone. To store zone information, provide other GNS components with zone information and to allow users to administer mapping in zones, GNS requires a centralized, persistent storage for zone information.

Administration of GNS zones and persistent storage of GNS zone information is provided by the NAMESTORE service. With NAMESTORE, users can manage name and records in zones and zone delegations to other zones. Other GNS components can obtain zone



**Fig. 6.9:** Network Format of GNS Record Blocks

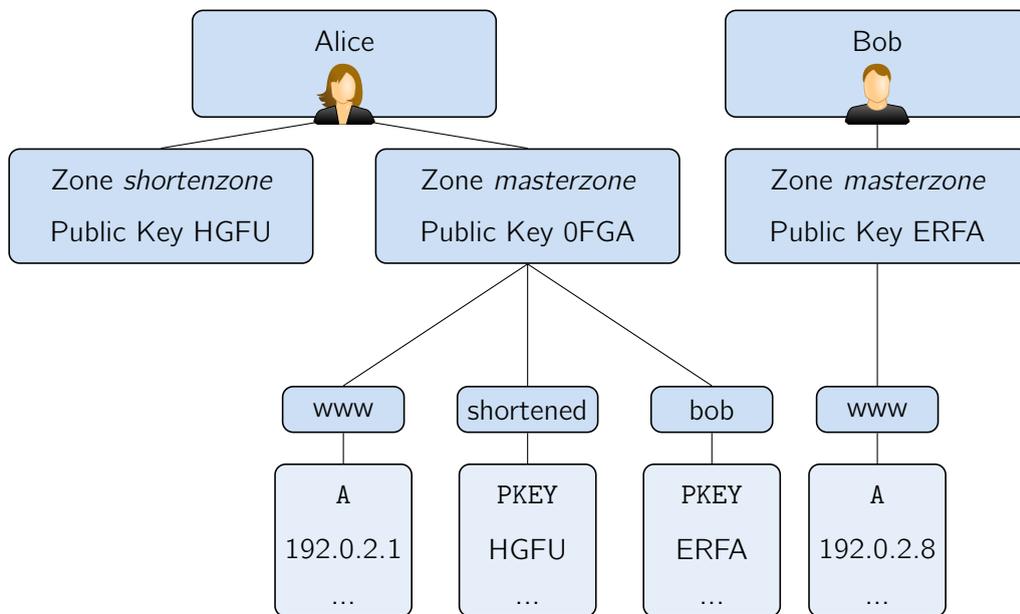
information from NAMESTORE. To administer GNS zone information, users can access the NAMESTORE information using the NAMESTORE CLI `gnunet-namestore` or other tools like the graphical user interface `gnunet-namestore-gtk`, providing a convenient interface to NAMESTORE. (Human) users can use egos provided by the IDENTITY service to refer to zones when using tools like the command line tool, while programmatic clients use the private key to refer to zones. The structure of zones in GNS is depicted in Figure 6.10.

To provide reliable and high-performance storage, NAMESTORE provides a plugin based storage mechanism and supports multiple database storage back ends. Supported storage backends are SQLite<sup>10</sup> and PostgreSQL<sup>11</sup> database systems. All storage back ends are implemented as loadable plugins and are accessed using an API, hiding the specific implementation. The NAMESTORE plugin API provides functions for NAMESTORE to add, delete and lookup zone information as well as to monitor zones, iterating over zone information and perform zone to name mappings. An overview over the plugin API is included in Appendix A.1.

The NAMESTORE service maintains a database (table) for every GNS zone, containing

<sup>10</sup> <https://www.sqlite.org/>

<sup>11</sup> <http://www.postgresql.org/>



**Fig. 6.10:** Structure of Zones in GNS

information about the records existing in this zone and the labels the records are stored under. To add a new label and records, users have to specify the zone where to store the mapping (e.g. using the ego provided by `IDENTITY`), the label they want to store the records under, the type of record to create and the value for the record plus additional information about the record (for example if the records are public or private). The `NICK` record is stored in `NAMESTORE` assigned to the root of the zone with the label “+”. The record is a private record to prevent the record from being published in the DHT. `NAMESTORE` merges this record if it exists automatically with the every set of records passed to the `NAMESTORE` clients.

When Alice wants to create a mapping with the label `web` for her web server in her master zone with the IP 192.0.2.1, she instructs `NAMESTORE` to create in the zone with the ego `masterzone` under the label `www` a public `A` record with the value 192.0.2.1. Alice can now use the name `www.gnu` to access her web server. `NAMESTORE` stores this information in the database for the respective zone. Clients like the GNS service can query `NAMESTORE` for this information, for example to publish the records in the DHT. `NAMESTORE` forwards the information to the `NAMECACHE` component to allow the GNS resolver to access the local mappings.

To administer and retrieve zone information, `NAMESTORE` provides an extensive API to clients to manage information in zones and to obtain information about zones. To manage records in GNS zones, `NAMESTORE` provides functions to programmatic clients to store, remove and lookup records in GNS zones.

To create records in a zone with `NAMESTORE`, the client has to provide the private key for the zone to store the records in, the desired label and the records to store. Storing and deleting records are internally realized with the same API function: to delete records stored under a name, a call to the store command with zero records included is used. When `NAMESTORE` receives the information, it first checks if the name is to be deleted based on the number of records included and if the name exists in the database: if not, the operation can be skipped. If the name exists, `NAMESTORE` stores the records in the database using the plugin API call to store records. This call replaces all existing records

if such records exist and if no new records are given, the name is deleted. No merging of existing records and new records is done. When the store operation was successful, NAMESTORE notifies all clients monitoring this zone about this change. In addition, it creates a new record block as described with record handling in Section 6.8.4 using the modified record data and the private key of the zone required to cryptographically sign and encrypt the record block. This block is then given to NAMECACHE and periodically refreshed to allow lookup operations for local information purposes as described in Section 6.8.6.

To look up a name in a zone, the client has to specify the private key of the zone, the desired label and an iterator to call with each result. NAMESTORE uses the plugin API to retrieve the required records from the database. If a NICK record exists, the records returned will be merged with the NICK record. The result are then given to the client which is notified by calling the iterator function with the results.

NAMESTORE can also perform a *reverse lookup* to check if a delegation for a given zone exists. NAMESTORE clients pass the public key of the zone to lookup to NAMESTORE and NAMESTORE returns (if existing) the PKEY record and the label the record is stored under. This functionality is used to check if a delegation for a public key exists, for example with automatic shortening.

NAMESTORE also provides the possibility to iterate over the content of a specific zone or all zones available, particularly useful for caching and to publish public records in the DHT. To monitor changes in a zone or to iterate over all records stored in a zone, the client has to specify the private key of the respective zone and a callback function to call with results. When iterating over zones, NAMESTORE provides the client with every single result it obtains from the database, passes this result to the client and waits for the client to process the result and notify NAMESTORE to continue. Similar to the lookup, for each result an existing NICK record is merged into the result. A client can also specify to be notified about changes to a zone. A client has to specify the private key of the zone and whenever a zone is updated, the client gets notified about this change from NAMESTORE.

### 6.8.6 Caching GNS Information

While the NAMESTORE component is used to store information for zones the specific user is *authoritative* for is the NAMECACHE subsystem intended to cache GNS information retrieved before, for example by a DHT lookup. This caching improves performance since the GNS resolver can reuse cached data and avoid repeated DHT lookups. NAMECACHE uses a plugin architecture similar to NAMESTORE to store the cached entries in a persistent way. The NAMECACHE plugin API consists of two functions to cache a record block and to lookup record blocks as shown in Appendix A.2.

To allow lookup operations for zones administrated by the local user, NAMECACHE caches blocks created by NAMESTORE when a record is stored in NAMESTORE. The GNS resolver stores records retrieved from a DHT lookup in the NAMECACHE and asks NAMECACHE for cached results before performing a DHT lookup to resolve a name as described with the lookup process in Section 6.8.8.

To store records in the cache, the client passes the GNS record block to be cached to NAMECACHE. If an entry is already existing in the cache, this block is replaced by the updated block. To lookup a cached block, the client specifies the derived hash for the zone and the name as described in Section 6.8.8. NAMECACHE performs a lookup in the database and returns the record block if it is existing.

### 6.8.7 Zone Revocation with GNS

In GNS, revocation and revocation checks processes are managed by a separate REVOCATION service. This service provides the possibility to check during a resolution process if a zone was revoked and manages the decentralized revocation process by maintaining revocation information received from the peer-to-peer system and providing the possibility to distribute revocation information.

To manage revocations, the REVOCATION service maintains a local, persistent list of all revocations it is currently aware of. This revocation list is stored in file written to disk to be available after a restart. To spread revocation information, REVOCATION exchanges revocation information with other peers. To exchange the revocation information with other peers, REVOCATION uses GNUet's CORE service to communicate with other participants in the network, previously described in Section 4.3.19. When two GNS and revocation enabled peers connect, the peers exchange revocation information in an efficient way using efficient set reconciliation provided by GNUet's SET service. Every peer maintains a set of all revocation it knows and when it connects to another peer, both peers exchange their revocation information by building the union of their revocation sets. New elements a peer learns from this process are added to this set and forwarded to other peers. Later on, these peers exchange new revocation information they acquire using a flooding mechanism: all new acquired revocation information is flooded to all connected peers which store this information and pass them to other peers.

To revoke a zone, a proof of work is required. This proof is based on computing a script key derivation, which was designed to be computational expensive to prevent hardware attacks as described in [PJ13]. To be able to immediately revoke a zone if required, a revocation proof of work can be pre-generated and stored in a file. Using this pre-generated file, the revocation can be performed immediately. If a zone is revoked locally by the user or the REVOCATION service receives new revocation information from other peers via the set or the flooding mechanism, the revocation service first checks the proof work and stores the revocation entry in the persistent file. The new revocation element is added to the revocation set of the SET service and sent to other peers on CORE level. A user can revoke his zone using the `gnunet-revocation` command or the service can be accessed by other applications using the REVOCATION API.

### 6.8.8 Censorship-Resistant and Privacy-Preserving Publication and Name Resolution

Name publication and resolution with GNS is realized by the GNS service. GNS is a separate GNUet process collaborating with the NAMESTORE, NAMECACHE and IDENTITY service described in previous sections. GNS is also responsible to publish users' public zone information obtained from NAMESTORE to the DHT. GNS service is responsible to resolve GNS names using local GNS zone information and by querying the DHT. To resolve names, the GNS service interacts with NAMECACHE to obtain local zone information stored with NAMECACHE by NAMESTORE and to obtain cached information from previous lookup operations in the DHT. In addition it can interact with NAMESTORE during name resolution if automatic shortening for GNS names is enabled.

The  $R^5N$  DHT implementation provided by GNUet supports an extensible block plugin architecture to validate blocks stored and retrieved in the DHT. To prevent attacks storing malformed blocks for zones and labels, GNS uses a  $R^5N$  block plugin to validate the signature of record blocks stored in the DHT. Every participant in the DHT overlay automatically checks the correct signature of GNS record blocks routed over this peer and

when data are retrieved from the DHT using the public key and the signature stored in the record block. So malformed or malignant blocks are automatically discarded by the DHT and not stored, forwarded or retrieved.

### 6.8.8.1 The GNS Publication Process

To publish GNS information in the DHT, the GNS service collaborates with the NAMESTORE service responsible to manage local zone information and publishes and refreshes local zone information in regular intervals in the DHT. Here GNS iterates over all local zones using the respective zone iteration functionality provided by NAMESTORE and publishes the public zone information in the DHT. The zone publication process uses an adaptive frequency: by default, zone information is refreshed every 4 hours. This interval is adapted based on the minimum relative expiration time (divided by 4) found in the GNS zone. This is required to keep records with a relative expiration time valid in the DHT.

In addition to periodic store operations, GNS also monitors NAMESTORE for changes to zones to immediately publish modified GNS information to the DHT. Here it relies on NAMESTORE's zone monitoring functionality as described with NAMESTORE in Section 6.8.5. When NAMESTORE notifies GNS about changes in a zone, it includes the (ECDSA) private key of the zone, the respective label and the records assigned to this label. Before records are published to the DHT, they are processed and converted to record blocks as described in Section 6.8.4. During this conversion, relative expiration times are converted to absolute times and all private or pending records are skipped. The resulting set of records plus the optional NICK record are serialized to a record block. The expiration time of the record block is based on the minimum expiration contained in the block. If a SHADOW record exists with a longer validity is the block expiration set to the expiration time of the SHADOW record.

The record data in the record block are encrypted using AES-256 and Twofish-256 with a symmetric encryption key and an initialization vector both derived from the label and the ECDSA private key of the zone. The record block containing the encrypted record is signed using an ECDSA key derived from the private key of the zone, the label this block belongs to and an additional context string (here "gns"). The signature, the signature purpose (for signing records in GNS GNUNET\_SIGNATURE\_PURPOSE\_GNS\_RECORD\_SIGN), the expiration time of the record block and the public key of the derived key used to sign the block is stored in the block. The signed record block, containing only encrypted records, is now stored in the DHT. This DHT key is the SHA-512 hash of a public key derived from the public key of the zone, the label and an additional context string (here "gns") used to allow different applications to use this publication mechanism without conflicting DHT data.

The process of creating a signed and encrypted record block is depicted in pseudo-code in Listing 6.1.

**List. 6.1:** Encryption and Signing of GNS Record Blocks

```

RECORD_Block
block_create (EcdsaPrivateKey zone_private_key, TIME_Absolute expire, string label, RECORD_Data ↔
    public_records)
{
    RECORD_Block block;
    EcdsaPrivateKey derived_private_key;
    EcdsaPublicKey zone_public_key;
    EcdsaPublicKey derived_public_key;
    SymmetricSessionKey symmetric_key;
    InitializationVector iv;

```

```

zone_public_key = ecdsa_key_get_public (zone_private_key);
derived_private_key = ecdsa_private_key_derive (zone_private_key, label, "gns");
derived_public_key = ecdsa_key_get_public (derived_private_key);

block->derived_key = derived_public_key;
block->pubkey = derived_public_key;
block->expiration_time = expire;

(symmetric_key,iv) = derive_block_symmetric_key (label, zone_public_key);
block->records = symmetric_encrypt (public_records, symmetric_key, iv);
(block->signature, block->purpose) = ecdsa_sign (block, derived_private_key, ←
PURPOSE_GNS_RECORD_SIGN)
return block;
}

DHT_Key
query_from_private_key (EcdsaPrivateKey zone_private_key, string label)
{
    EcdsaPublicKey zone_public_key;
    EcdsaPublicKey derived_public_key;
    DHT_Key key;

    zone_public_key = ecdsa_key_get_public (zone_private_key);
    derived_public_key = ecdsa_public_key_derive (zone_public_key, label, "gns");
    key = hash (derived_public_key);
    return key;
}

perform_dht_put (EcdsaPrivateKey zone_private_key, string label, RECORD_Data public_records)
{
    RECORD_Block block;
    TIME_Absolute expire;
    DHT_Key dht_key;
    size_t block_size;

    expire = get_mininum_expiration_time (public_records);
    block = block_create (zone_private_key, expire, label, public_records);
    dht_key = query_from_private_key (zone_private_key, label);
    DHT_put (dht_key, block);
}

```

### 6.8.8.2 The GNS Resolution Process

The second major functionality of the GNS service is to resolve names using local zone information and the DHT. This name resolution process is integrated with caching of name information with NAMECACHE as described in Section 6.8.6 to increase resolution performance and with automatic shortening of delegation chains as introduced in Section 6.6.5. Integration of GNS name resolution with the operating system will be explained in the following Section 6.8.11.

When a name has to be resolved, a (GNS) client requests a name resolution with the GNS service. Since GNS was designed to integrate with existing DNS and GNS names are integrated into the DNS namespace using the “.gnu” pTLD for relative and “.zkey” pTLD for absolute names, the GNS service supports to resolve both GNS and DNS names. GNS inspects the name to resolve and when the name is a GNS name it uses the GNS resolver or otherwise instructs a DNS resolver to resolve the name in DNS.

To resolve a name with GNS, the client passes the name to resolve, the desired record type to lookup (e.g. A) and the ego (as described in Section 6.8.3) of the GNS zone used as a starting point for name resolutions. If no zone is specified, by default the master zone configured by the user for GNS with the IDENTITY service is used. The GNS resolver first evaluates based on the TLD if the name passed is a DNS name or a IPv4 or IPv6 address (and therefore a (reverse) lookup using the DNS resolver is required) or if the name passed is a GNS name ending in one of the GNS TLDs “.gnu” or “.zkey”. If a GNS

name is passed, a GNS resolution process is initiated, otherwise the name is passed to the legacy DNS resolver.

With the GNS resolution process, GNS names (representing a delegation path in GNS) are resolved recursively from right to left, starting with the TLD “.gnu” or “.zkey”. Due to delegation to different zones, different authorities responsible for the labels exist. The resolver creates an *authority chain*, containing the zone information for the each specific label, to be able to resolve relative names in records and replaces relative names with absolute names.

When resolving a relative GNS name (using the “.gnu” pTLD), the resolver removes the TLD and starts the authority chain with the zone passed by the client as a starting point and begins to lookup the right-most label in the specified zone. For an absolute “.zkey” name, the resolver removes the “.zkey” TLD and converts the identifier to a public key representing the zone to start the recursive lookup for labels in.

The resolver now resolves recursively all labels in the given name from right to left, label by label. If resolution for one of the labels fails, the resolver cancels the resolution process and notifies the client about the failed resolution process.

When resolving a label, the resolver first performs a revocation check (as described in 6.8.7) to check if the current zone was revoked and fails the resolution process if the zone is marked as revoked. If the zone information was not revoked, the resolver performs a lookup for the label in the NAMECACHE (described in Section 6.8.6) to check if a cached version of the information exists. To perform this lookup in the cache, the resolver uses the same lookup key (based on the zone’s public key and label) as used with the DHT to perform a lookup. The key used for the lookup is the SHA-512 hash of an ECDSA public key derived from the zones ECDSA public key, the label and an additional context string (“gns”). If a record block is found in the NAMECACHE, the resolver checks the expiration time of the block and decrypts the encrypted record data stored in the block and uses this information to continue the resolution process. To decrypt an encrypted record block, the resolver derives a symmetric decryption key and initialization vector from the zone’s public key and the label to lookup. The symmetric key and initialization vector are used to decrypt the records in the record block. The correctness of the block’s signature is ensured by the  $R^5N$  block plugin when the block was obtained from the DHT and therefore the resolver does not have to check the signature again. If the block is not expired, the decrypted record block is used to continue the resolution process. If no block was found with the NAMECACHE lookup, a lookup in the DHT using the same key as with the NAMECACHE is performed. If the DHT returns a response, the signature of the record block returned is verified by the DHT automatically and the block is discarded if the signature is invalid. If a result was returned from the DHT the blocks expiration time is checked and the block is stored in NAMECACHE. The resolver decrypts the record block using the same symmetric decryption key as described with the cached block.

The process of calculating the required lookup key and the derivation of the symmetric decryption key is depicted in pseudo-code in Listing 6.2.

List. 6.2: Decryption of GNS Blocks

```

RECORD_Data
perform_dht_get (EcdsaPublicKey zone_public_key, string label)
{
    EcdsaPublicKey derived_public_key;
    SymmetricSessionKey symmetric_key;
    InitializationVector iv;
    DHT_Key dht_key;
    RECORD_Data public_records;

    derived_public_key = ecdsa_public_key_derive (zone_public_key, label, "gns");
    dht_key = hash (derived_public_key);
    record_block = DHT_get (dht_key);

    (symmetric_key,iv) = derive_block_symmetric_key (label, zone_public_key);
    public_records = symmetric_decrypt (block->records, symmetric_key, iv);

    return public_records;
}

```

When a decrypted block is retrieved from NAMECACHE or the DHT, the records contained in the decrypted block are evaluated. The resolver iterates over the records contained in the record block and extracts the required information.

GNS records can contain relative names as described in Section 6.6.9.2. Relative names in records (containing "+.") are translated to absolute name. This translation is based on the current authority in the authority chain. If the resolver encounters DNS specific records such as CNAME, SOA, MX, SRV, records, it parses the values of the records and translates relative to absolute names and continues name resolution or returns the resulting records to the clients.

If the record block contains GNS specific record types such as PKEY, NICK, GNS2DNS, VPN records or records types treated differently from DNS, the resolver handles these GNS specific records:

**PKEY records** are used to delegate labels to a zone. A PKEY records makes the resolver extract the public key in the record and create a new entry in authority chain based on this public key. Based on this delegation information, the resolver now initiates a new recursive lookup with the leftover name and the new authority information.

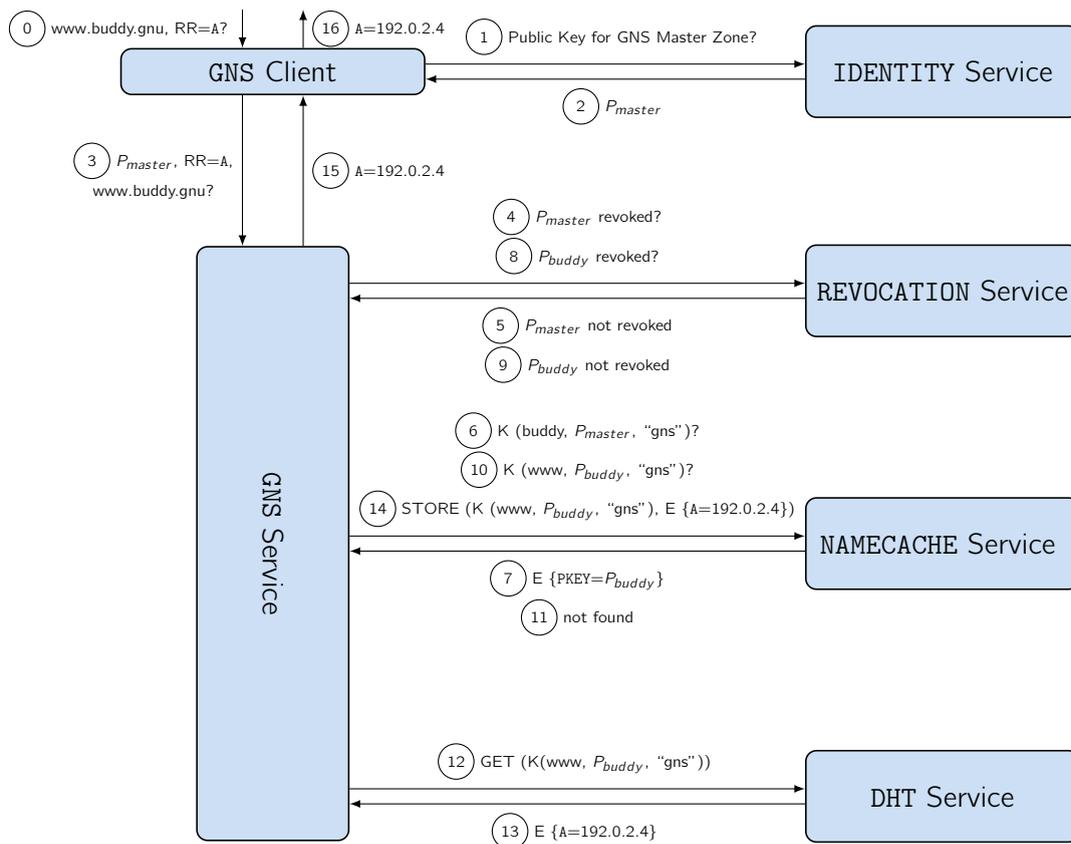
**NICK records** specify the desired label a user suggests for a zone when added to a zone e.g. due to automatic shortening. The solver extracts the proposed nickname for the zone and stores this information in the current authority entry in the authority chain. This NICK record is used with automatic shortening as described in Section 6.8.9: if additional records are found for a name and shortening is enabled on the system, the resolver tries to apply automatic shortening to the zone name with the given nickname.

**GNS2DNS records** are used to delegate a name from within the GNS system to legacy DNS. A GNS2DNS makes the resolver start a new recursive resolution based on the DNS domain included in the GNS2DNS record and the IP address of the DNS server as described with GNS2DNS in Section 6.6.9

**CNAME records** are used to specify an additional common name for a label. If such a record is encountered, this alternative is merged in the resolution chain and a new resolution process based on alternative name is initiated. Since CNAME records can contain relative names, the resolver has to check if the name is relative or absolute. If the name is relative, the resolver creates a new request for the absolute name based on the remaining name and the current zone the CNAME record originates from.

**VPN records** If the label resolved is the last (left-most) label and the record is a VPN record, a redirection to the destination specified in the VPN record using GUNet's VPN service is initiated. If the VPN redirection was successful, depending on the requested record type (A or AAAA) the IPv4 or IPv6 address of the redirected service is returned in an A or AAAA record.

If the last (left-most) label is resolved or no further recursion is required, the processed records are returned to the client and the resolution process is terminated. If any of the steps before fails, the resolution process is terminated and the clients is notified about the failed resolution. Figure 6.11 depicts how the name "www.buddy.gnu" is resolved to the IP address 192.0.2.4.



$E\{ \}$  : Encrypted Record Block  
 $P_x$  : Public Key of zone  $x$   
 $K()$  : Function to calculate DHT key

**Fig. 6.11:** The GNS Resolution Process

### 6.8.9 GNS Shortening

Delegation is a central concept in GNS, since it allows users to resolve names provided by other users. But having long delegation chains is also not desired. Long delegation chains make name resolution error prone if one of the elements in the chain cannot be resolved. Therefore, long delegation chains are shortened once the names could be resolved. The

shortened names are put in a special *shorten zone*, one of the default zones used by GNS. This default zone is addressed by the GNS using an ego as introduced in Section 6.8.3.

To enable automatic delegation chain shortening, NICK records stating a user's desired nickname are used. As described in Section 6.4.5, in a petname system a user is identified by a globally unique identifier. Each user can freely assign a petname to this user when referring to this user. But the user can propose a *nickname* to be used. In GNS this preferred nickname is distributed in GNS specific NICK records and NAMESTORE creates a private NICK record in the root of the zone assigned to "+" label. Since this record is private it will not be published, but instead the NAMESTORE service automatically includes this NICK record in the set of records for all labels in a zone. So the desired nickname gets published with all labels published for a zone.

The shortening (if enabled) occurs in the GNS resolver when records in a remote zone were successfully resolved. When a name is resolved, the resolver recursively resolves all labels in a name. By doing so, the resolver may also find delegations in the name. When it received a NICK records for this zone, it will initiate a shortening process based on the preferred nickname contained in the NICK record and the public key for the zone.

When a shortening process is initiated, the shortening logic first checks if for the zone to shorten a delegation entry in the NAMESTORE exists and if so the shortening process is not required and therefore aborted. In the next step, the shorten logic performs a lookup in the NAMESTORE if the preferred nickname is already taken and existing in the shorten zone, and if so the shortening is aborted. If the preferred nickname was taken, the shortening request is re-issued with the original label. If this request also fails, shortening is not possible. Otherwise a new private PKEY record is created in the shorten zone and the shortening process successfully terminated.

### 6.8.10 GNS on Multi-User Systems

GNS is designed to provide every user with his own namespace and the possibility to manage mappings in his zones. So name resolution with GNS is user-specific contrary to DNS resolving globally unique names similar for all users.

So in an environment where multiple users share an environment, the GNS implementation has to provide the functionality to support user-specific name resolution. This requirement is supported by the GNUnet framework GNS is implemented with. GNUnet can differentiate between *system* services, shared between all users, and *user* services, specific to a particular user on the system. With GNUnet, system services are services which are only required to be run once and which can be shared between users since they do not realize user-specific functionality. System services are for example services providing functionality to create the overlay network and provide connectivity between peers. User services provide user-dependent functionality with GNS being a prime example. GNUnet uses access control to limit access to services based on group and user ids. So user services can only be accessed from the user they belong to. To support multi-user environments with GNS, GNS services like NAMESTORE and GNS are executed as user services, running in a separate instance for every user, while using shared system services to communicate with the overlay network.

### 6.8.11 Integration with the Name Resolution Process

GNS is designed as an alternative to existing DNS to allow seamless migration and coexistence. It was designed and implemented to allow users to use GNS without having to learn and understand new concepts and only advanced users may choose to use advanced

features of GNS. This is particularly true for GNS names, designed to look like common DNS names and GNS names being integrated in DNS namespace using pTLDs.

But to allow users to use GNS and to resolve names provided by GNS, GNS name resolution has to be integrated in the name resolution process of the operating system to provide convenient user experience. Here it is important to provide GNS name resolution to applications without having to adapt and modify applications.

Several approaches have been implemented to integrate GNS name resolution with the operating system and existing applications:

#### 6.8.11.1 Name Service Switch

On GNU systems, libc's Name Service Switch (NSS) [Fou] provides an flexible and extensible method to provide configuration information to system components and name resolution mechanisms. Supported sources are local operating system files (e.g. `/etc/passwd`, `/etc/group`, and `/etc/hosts`), DNS or directory services like Network Information Service (NIS) or Lightweight Directory Access Protocol (LDAP). NSS supports to be extended with new information sources using plugins. To integrate GNS name resolution with the operating system, we provide a GNS plugin for NSS. With NSS, a set of plugins can be configured in a specific order to resolve service information. For names, the `hosts` service is used with a set of plugins to resolve hostnames. The GNS plugin has to be configured to be used before names are used with DNS. With the GNS plugin activated, the GNS NSS plugin is used before the DNS NSS plugin to resolve names. When the plugin cannot resolve a GNS name, name resolution fails. Otherwise, the next plugin configured is used to try to resolve the name.

This approach has the advantage of offering fully personalized resolution even on multi-user systems, but applications not using libc's name resolution mechanisms can bypass the GNS name resolution. Mechanisms similar to NSS exist for other platforms and we have an equivalent plugin working on Microsoft Windows using so called *Namespace Providers*. Details for the Windows specific implementation can be found on our website<sup>12</sup>.

#### 6.8.11.2 GNS-Enabled Resolver

Integration of GNS can also be achieved by using a GNS-enabled resolver. With our implementation, we provide a DNS-to-GNS gateway. The gateway resolves names in the ".gnu" and ".zkey" pTLD internally and acts as a proxy for all other TLDs, forwarding requests to an actual DNS server. The resolver configuration (for example `/etc/resolv.conf` on Linux systems) has to be modified to point to an IP address (i.e. 127.0.0.1) running the GNS-enabled resolver software.

Applications bypassing the system's name resolution mechanisms and implementing their own name resolution mechanism do not benefit from this approach and using this approach in a multi-user environment depends if the system supports per-user resolver configuration which is not true for most systems. On many systems, this approach interferes with other networking management software, using DHCP information to set DNS server information dynamically. These issues make this approach not suitable even for more advanced users.

---

<sup>12</sup> <https://gnunet.org/dev-gns-w32>

### 6.8.11.3 Intercepting Queries

An additional possibility to integrate GNS with name resolution, is to intercepts DNS queries from the local system using packet filtering rules e.g. using `iptables`. With our implementation, we support to intercept DNS traffic using policy based routing. With this approach, we MARK an outgoing DNS request if it is not send by the GNS service. By using a second routing table in the Linux kernel, these marked packets are then routed through a virtual network interface and can thus be captured unmodified. The interception application then processes the query and decides how to handle it: A query to an address ending in “.gnu” or “.zkey” is hijacked by the GNS service and resolved internally using GNS. Other queries not hijacked will be sent to the respective DNS nameserver. The answer to the query will always be sent back through the virtual interface with the original nameserver as source address.

This approach allows to intercept all name resolution traffic and is independent from the resolver and supports applications implementing their own resolution logic. But this approach is not suitable for multi-user environments with multiple users running GNS on the same system since the interception logic cannot distinguish to which user an intercepted request belongs to resolve names using the user’s GNS zone.

### 6.8.11.4 GNS Proxy Server

An approach in particular useful to integrate GNS with the name resolution in combination with the support if legacy applications is to use a proxy server to access GNS enabled websites. Here the proxy server resolves the GNS names and can in addition process the websites with respect to relative names and HTTPS connections.

With our GNS implementation, we provide the `gnunet-gns-proxy`, a SOCKS proxy allowing arbitrary networking applications to access services using GNS names. The SOCKS protocol [LGL<sup>+</sup>96] allows networking application to access services using a proxy server to traverse firewalls or other network filters. SOCKS proxies work on a lower level than HTTP proxies and supports both TCP and UDP as well as IPv6 and name resolution for proxy requests.

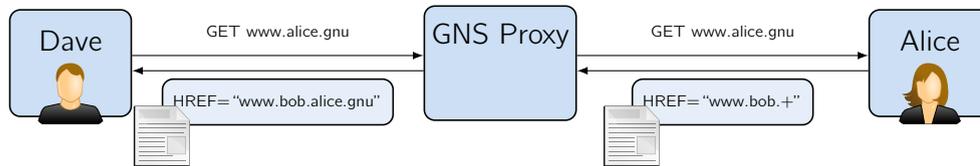
This approach is particularly useful to enable any networking application to use GNS names, but requires in a multi-user setup a per-user proxy instance. The use of a proxy requires to configure every application manually, especially when the application does not support to retrieve proxy information from the operating system environment.

With the `gnunet-gns-proxy`, we enable networking applications to establish connections using GNS or DNS names. When the `gnunet-gns-proxy` is instructed to establish a connection using a GNS name, the proxy resolves GNS using the local GNS installation. The `gnunet-gns-proxy` collaborates with GNS service to resolve the GNS names.

To link information scattered over different systems together with GNS, we introduced in Section 6.6.3 the idea of relative names for GNS. GNS uses relative names which have to be interpreted relative to the current zone of origin, similar to relative URLs commonly used with web sites.

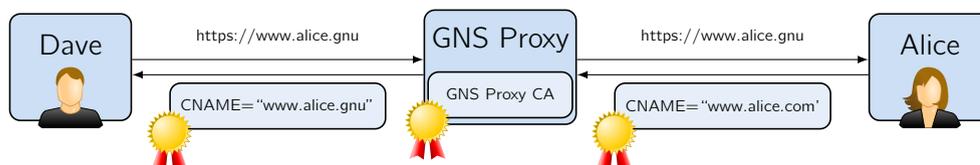
When Alice creates a website and puts a link to Bob’s web server (she can access using `www.bob.gnu`) on the site, she uses the relative name `www.bob.+` for the link. When Dave accesses Alice’s website using `www.alice.gnu` the the link on the website with the relative name `www.bob.+` has to be rewritten to `www.bob.alice.gnu` since `alice.gnu` is the current zone of origin. This translation is provided by the GNS proxy server which translates GNS names on websites according to the current zone of origin. This rewriting

is depicted in Figure 6.12. Issues related with this approach are URLs created on the client-side for example with JavaScript or URLs stored in cookies.



**Fig. 6.12:** Using Relative GNS Names with the Web

An additional issue when integrating GNS with Web surfing, is the use of HTTPS. HTTPS provides secure communication using X.509 certificates. A web server provides a X.509 certificate binding a name to a certificate to the client, the client can use to establish a secure connection. With X.509 and GNS, these certificates contain the Fully Qualified Domain Name (FQDN) of the web server. This approach is not useful with GNS, since GNS names are not globally unique and cannot be included in a X.509 certificate. GNS uses a GNS specific LEHO record type, containing the LEgacy H0stname to use when verifying the certificate received. GNS-PROXY uses the name contained in the LEHO record to verify the certificate. To prevent clients like browsers to display a warning to the user (since they accessed a GNS name but received a X.509 certificate for a different name), the `gnunet-gns-proxy` can create X.509 certificates on the fly containing the correct GNS hostname. To create these certificate, the `gnunet-gns-proxy` realizes a X.509 certification authority, which has to be trusted by the browser validating the certificates provided by the proxy. Therefore, it is important to run the proxy directly with the user to prevent adversaries using the proxy to offer malevolent certificates to the user. This approach is depicted in Figure 6.13.



**Fig. 6.13:** Using GNS Names with the HTTPS

A similar issue is related to virtual hosting, where multiple websites are hosted under one IP address. Here the HTTP client includes a header in the HTTP request to specifying which host he wants to access. With DNS the FQDN is included in the header, an approach not suitable with GNS. Therefore, the proxy includes the FQDN obtained from the LEHO record in the HTTP header.

However, compared to native support by browsers, using a proxy has the disadvantage that dynamic links, which might be generated by (e.g. JavaScript) code executing within the browser, cannot be translated. Native support for GNS by browsers would improve security and usability.

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 a user browses `www.example.gnu` JavaScript code from `www.example.com` is forbidden to run due to SOP. This can be an issue as the cookies and JavaScript code might use the legacy hostname instead of the GNS name and would then be ignored in accordance with SOP. To solve this issue, the proxy translates links pointing to the LEHO name 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 [vK14].

### 6.8.12 Accessing GNS from DNS

With GNS we, in addition to integrate GNS with DNS, provide the possibility for users only using DNS, to access GNS information from within DNS. This allows GNS users to make their services available to DNS users and enables DNS users to access services available with GNS.

We provide a DNS-to-GNS gateway acting as an intermediary between DNS and GNS. When this gateway software is installed on a GNS enabled system and functions as a DNS server, authoritative for a DNS domain, on the other hand, it is possible to access (absolute) GNS names from DNS as subdomains of the domain the server is authoritative for. The gateway receives DNS requests for the DNS domain it is authoritative for and resolves names in subdomains as names in GNS as an absolute name.

The DNS-to-GNS gateway is useful to allow legacy systems to access the GNS distributed database without installing GNS or changing their system configuration. 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 gateway software to create such a gateway.

While this approach can help users to access GNS information without installing GNS, it only offers advantages with respect to security and censorship-resistance if the gateway operator and the network to the gateway can be trusted. Using a DNS domain as an entry point makes this approach susceptible to all issues described with DNS in the previous sections, allowing such a service to be easily monitored and censored.

## 6.9 Related Work and Comparison

Besides the systems mentioned before, there are other approaches to make DNS resilient against failures and attacks. The approaches described in the following sections are presented here, even though they have a different focus, goal or adversary model.

### 6.9.1 OpenDNS

OpenDNS, previously mentioned in Section 6.4.2, is a commercial DNS service, providing DNS services to customers as an alternative to using ISPs’ DNS servers. “Open” refers to the property to accept name resolution requests from all sources. Its revenue is based on providing an ad-based service and paid premium services and OpenDNS provides additional services including phishing filter, domain blocking and typo correction.

Most important with respect to this work, OpenDNS fully supports CurveDNS to encrypt and authenticate DNS communication between resolvers and authoritative servers. To provide confidentiality for users, OpenDNS also supports to encrypt communication between OpenDNS servers and users using DNSCrypt<sup>13</sup>. DNSCrypt uses similar to CurveDNS ECC with Curve25519 [Ber06] to authenticate DNS responses between users and OpenDNS servers.

OpenDNS aims to increase security for users by adding confidentiality and authentication to DNS communication but (based in its business model) still relies on existing DNS. This approach does not help against the adversary used in this paper, since our adversary

<sup>13</sup> <http://www.opendns.com/about/innovations/dnscrypt/>

still can make name mappings change or fail by tampering directly with the authoritative name servers. Since OpenDNS is a company falling under jurisdiction and legal power of a state, a censor could also try to put pressure on OpenDNS directly.

### 6.9.2 Namecoin

An alternative to DNS is the Namecoin system, described in [Swa11a], based on the idea of the Bitcoin cryptographic currency. Bitcoin is intended as an alternative digital currency, as described in [Nak08], not requiring any centralized authorities to create money or perform transfers.

Since with Bitcoin no centralized authority is required, a common history and consensus within the peer-to-peer network is required. With Bitcoin this is achieved by having a public transaction and consensus log. This public consensus log, in Bitcoin called the *block chain*, is a central and important component for crypto currencies. Bitcoin's block chain contains information about the history of all transactions with a cryptographically signed timestamp. It is used to achieve a consensus in the network on transactions and to prevent double spending, a hard to solve issue with electronic currencies. This system is used as a proof-of-work system, since to append a new block to the chain, the solution of a computational expensive cryptographic problem has to be computed. With Bitcoin the proof required is to find a partial hash collision of a SHA256 hash: a client has to increase an initial starting value, the *nonce*, and apply the SHA256 hash function two times, until the resulting has a certain number of leading zeros. If such a collision is found, a new block can be appended to the block chain and in addition new *bitcoins* are generated. A *bitcoin* is a monetary unit of value in this system. It is created by solving these proofs of work to append blocks to the block chain. The longest block chain is the one peers in the network have consensus about representing the current state of the system: the block chain cannot be changed, because to do so all prior work would have to be repeated.

Aaron Swartz had the idea [Swa11a] that decentralized electronic currencies and decentralized name systems basically have the same challenges and issues. Therefore, he suggested to transfer the Bitcoin idea to a decentralized alternative naming system called *Namecoin*. Namecoin is a decentralized name system providing a generic and decentralized key/value store serving as an alternative DNS system. It is based on the Bitcoin system, but uses a different block chain. It is designed to securely register and transfer names in a censorship-resistant way with a peer-to-peer based infrastructure without trusted authorities. It is supposed to not only be used as an alternative to DNS, but also as a generic key/value store used for any application like identity, messaging, voting or login systems and many many more.

In Namecoin names are registered within a transaction of a small fee in the system of the cryptographic currency. The transaction is directed to no one, so the transaction fee will be destroyed. It only ensures that names cannot be registered in unlimited number. All these registrations/transfers are broadcast within the system. *Miners*, the nodes trying to find the proofs-of-work, collect these registrations in a block and try to find the proof-of-work hash collision. If a node finds the required collision, it broadcasts the block in the network. Peers now start to work on the next block based on the hash of this block. The system also provides the possibility to transfer the possession of a name from user to user with a transaction in the system. Within such an transaction, the name and the value are included. When a block is added to the chain this registration is fixed within the block chain. Names have to be renewed about every 250 days with an additional transaction. To resolve a name a client has to check the blockchain if the name is contained in it.

## 200 6. GNS - A Decentralized, Privacy-Preserving and Censorship-Resistant Name System

With Namecoin it is possible to *square Zooko's triangle* and achieve all three properties of Zooko's triangle at the same time: memorable, global and secure names. Names are globally unique since with the blockchain a consensus in the network about the current state of the system and therefore all registered names exist. Names are memorable since they are explicitly chosen that way when registering these names. But with initial cheap cost when bootstrapping the system, the problem of name squatting exists. Initially the registration cost of names are low and therefore early adopters register names with the expectation to sell these names when the system is popular. So with Namecoin, names are an economic object possessed by an owner and are object of trade and economic behavior. The system is said to be secure, since with a majority of benign nodes in the network, the consensus of the block chain cannot be attacked. This assumption is only valid with an adversary model weaker than the adversary model used for this work. Note that our adversary model is not a far-fetched assumption in this context: as we saw with recent revelations about a single mining pool in the Bitcoin network possessing more than 51% of the computational power in the network [Far14a], 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.

Security can also be lowered by the concept of the Namecoin resolution process: a client resolving a name has to be in possession of the complete block chain to validate a name. The complete block chain can be large in size and therefore not be available on devices with restricted resources. These devices would then have to rely on third party resolvers and so creating a trusted third party, which may not be trustworthy or may be manipulated, just like in DNS.

### 6.9.3 TrickleDNS

Another system with similar goals is TrickleDNS, an extension to DNS that pushes (“critical”) DNS records between DNS resolvers of participating domains to provide “better availability, lower query resolution times, and faster update propagation” [SCSR12]. TrickleDNS is focused on defeating attacks on the availability (and performance) of record propagation in DNS, for example via Distributed Denial of Service (DDoS) attacks on DNS root servers. TrickleDNS is thus only concerned with ensuring distribution of authoritative records.

### 6.9.4 CoDNS

CoDNS, described in [PPPW04], is an approach to increase the reliability and resilience of the existing DNS system. CoDNS tries to increase the possibility of successful name lookup operations in case of a name server failure. Therefore, they provide the possibility of collaborative lookup operations. If a local name server fails, the client can use the CoDNS network to resolve the name. CoDNS is based on a large number of geographically distributed nodes which provide a DHT based DNS cache. These nodes cache DNS information in the DHT and all peers can retrieve this information for resolution processes from the DHT.

CoDNS's focus is to achieve resilience in case of technical issues by replicating information over a DHT. Against an adversary directly manipulating DNS information with the organizations providing this information or monitoring DNS traffic this approach cannot help.

### 6.9.5 Unmanaged Internet Architecture

Our idea shares some properties with the name system in the Unmanaged Internet Architecture (UIA) [For08], as both systems are inspired by SDSI/SPKI. 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 inter-operate 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/SPKI with respect to correctness. UIA's main focus is to provide a decentralized approach to connect user devices and to allow users to communicate with each other. UIA assumes users to communicate most of the time in local environment. Due to this assumption does UIA not focus on user privacy and censorship-resistance. Devices in a local environment try to automatically locate each other and to exchange information. [For08] proposes that information exchange could be restricted to device groups the respective devices are interested in, but this approach is proposed for performance reasons and not to improve the system's privacy.

## 6.10 Use Cases for GNS

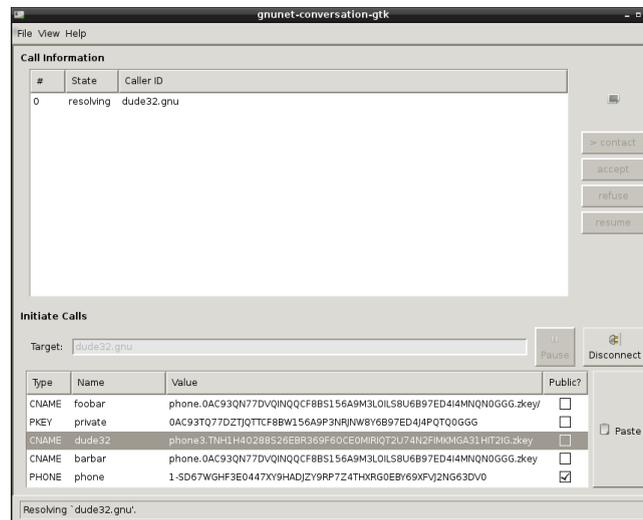
While using GNS primarily as an alternative to DNS may seem appealing, there are other applications where GNS is much more likely to succeed as DNS has failed to be competitive. In particular, we are deploying GNS in the broader domain of social networking applications, as GNS is a natural fit when it comes to handling social relationships.

### 6.10.1 Telephony

A simple use case for GNS is to use GNS for peer-to-peer voice applications. Existing peer-to-peer 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.

We implemented Conversation (Figure 6.14), a peer-to-peer voice application included in GNUet which uses GNS to establish a secure connection between the participants. Conversation strongly benefits from the possibility to extend GNS with additional, application-specific record types. When Conversation is started, it automatically creates a PHONE record in the GNS zone of the local user. A call can then be made by specifying the respective GNS domain name. The name is then resolved to obtain the PHONE record, which includes the necessary addressing information to initiate a secure channel to the callee.

From the user's perspective, a GNS zone in Conversation is simply an address book, and our Graphical User Interface (GUI) allows the user to manipulate the contents of a GNS zone as if it were a simple address book: Alice can add entries (via copy-and-paste) and call bob.gnu if "bob" is in her address book. In Conversation, the address book of a user contains PHONE records, CNAME records (aliases) and PKEY records (public keys) as depicted in Figure 6.14. The user specifies the "Target" of the call using a GNS name. If Bob calls Alice, she will see that the call is from bob.gnu, because the reverse lookup of Bob's public key results in bob.gnu. If the reverse lookup fails, Conversation displays an



**Fig. 6.14:** Screenshot of the Conversation User Interface

identifier in the `.zkey` TLD with the respective public key of the caller as the label. The callee can then choose to import the caller’s public key to his address book.

Thus, in Conversation the public key of a GNS zone functions as the caller identity. Naturally, a GNS user can create many zones as described in Section 6.6.2. Consequently, a Conversation user can easily assume many identities, which are called *egos* in Conversation. This way, a user can assume multiple identities to easily separate professional and personal interactions.

Note that Conversation does allow users to benefit from GNS’ delegations. For example, Bob could choose to make `carol.gnu` a public record in his address book and then tell Alice to call “carol”. Alice could then call Carol using `carol.bob.gnu` as she got the referral from Bob — and if Bob was trustworthy, she would be assured of a connection to the correct Carol. As described with automatic shortening in Section 6.6.5, Alice’s GNS can then shorten `carol.bob.gnu` to `carol.gnu` such that her future interactions with Carol no longer depend upon Bob.

### 6.10.2 Decentralized Online Social Networking

Most current alternatives to centralized online social networks use a federated architecture [vdV13]. In these architectures, social data is still hosted at more or less centralized servers, and identities are still represented using accounts of the form `username@host` (or the equivalent `host/~username`). This is a key reason why federated architectures offer at best minimal privacy benefits over centralized systems.

GNS offers a natural, secure and decentralized way to name users in online social networks. As with Conversation, GNS can easily function as an address book for social circles, and the initial key exchange can be facilitated via the usual social interactions of the respective groups. Unlike certain commercial offerings, GNS users will enjoy the benefits of being able to create multiple egos and thus are more able to segregate their online identities when desired.

At the time of writing this work, a concrete design for a fully decentralized peer-to-peer social networking application using GNS in combination with an extensible messaging protocol exists [Tot13] and the community is working on its implementation.

### 6.10.3 Messaging

A first simple application to use GNS for messaging is to store Pretty Good Privacy (PGP)/GnuPG public keys in GNS. Once users do this, GNS provides an alternative to the Web of Trust (WoT). However, unlike the WoT, GNS provides query privacy. Furthermore, the identity model is different. In key signing parties for the WoT, users are expected to correctly perform a complicated seven-step process<sup>14</sup> to assure that government-issued identity cards match e-mail addresses, thus linking PGP 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 e-mail address. Furthermore, the use of real-world identities makes the protocol fundamentally unsuitable for users that would like to be pseudonymous.

With GNS, the key exchange protocol can be dramatically simplified. After creating his key, Bob only needs to give his public key to Alice via a secure channel. Alice then assigns a label for the key (or confirms the suggested nickname) in her zone.

Naturally, given that GNS can contain more information than just the PGP key, it will be possible to use GNS as a PKI for the various “secure” messaging systems that are currently being developed. In most of these designs, users still rely on the provider of the “secure” messaging system to provide some form of identity management. Systems like ZRTP [ZJC11] and PANDA<sup>15</sup> that side-step the identity provider issue using out-of-band information can be easily integrated with GNS to facilitate the GNS key exchange as well, broadening the utility of these mechanisms beyond their current domain.

### 6.10.4 DNSSEC Done Right: Securing the Web

Many of the envisioned benefits of DNSSEC are currently not realized as DNSSEC signatures are typically only validated by the ISP’s resolver and not by the stub resolver of the end-user. Furthermore, even if DNSSEC records were validated by the stub resolver, existing legacy DNS resolution APIs fail to provide validation information to the applications. Finally, legacy DNS resolution APIs also fail to expose details of most DNS record types (such as TLSA records) to the application. Thus, to really benefit from DNSSEC a major overhaul of existing applications and operating system functions would be needed. Even on the provider side further changes are needed, as for example many commercial DNS providers currently do not support users adding DNSSEC, TLSA or other “unusual” record types to their hosted zones.

The situation is much simpler with GNS’ clean slate approach. GNS-enabled applications always receive the locally cryptographically verified, decrypted raw values for all record types. Furthermore, the GNS zone management GUI makes it easier to plugin support for additional record types and can, for example, create advanced DNS records such as TLSA records. The user can remain unaware of the technical details when creating TLSA records in the GNS zone management interface as depicted in Figure 6.15.

With respect to using TLSA records to secure connections of legacy Web clients, we have created an HTTP SOCKS proxy that enables legacy applications to resolve domain names via GNS (or DNS, depending on the TLD). Furthermore, the proxy automatically validates TLS connections, using X.509 for DNS and TLSA records (if present) for GNS. If the validation is successful, the proxy acts as a certificate authority and uses its own private key to certify the GNS name to the application using an X.509 certificate that is created on-the-fly. The legacy HTTP client using the proxy must be configured to accept

<sup>14</sup> <https://wiki.debian.org/Keysigning>

<sup>15</sup> <https://pond.imperialviolet.org/tech.html>

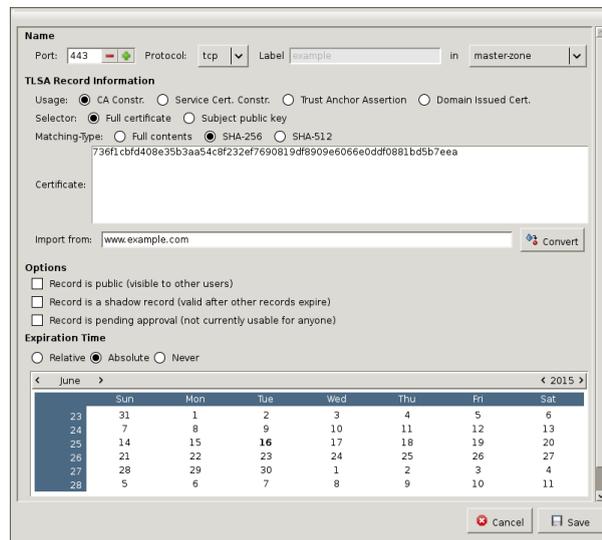


Fig. 6.15: GNS User Interface to Create TLSA Records

the public key of the proxy's certificate authority. As the proxy is expected to run on the local host, this has the effect of end-to-end certificate validation using TLSA records.

### 6.10.5 Other Applications

Other applications that would be a good fit for GNS include naming for Tor hidden services (memorable names for ".onion"), and assigning names for the future Internet-of-Things where effectively each user will need to manage his own namespace to address the plethora of embedded networked devices under his control.

### 6.10.6 Synergy

GNS as a name system, public key infrastructure and identity management approach is not limited to one particular application domain. However, once users start to populate their zones for one application, they can trivially use the resulting structure for all of the other GNS-enabled applications. For example, Alice might follow Bob's social status updates at `social://bob.gnu/`, visit his website at `https://bob.gnu/` and call him using `conversation://bob.gnu/`.

### 6.10.7 Out-of-Band Exchange of Zone Information

To be able to resolve names from other users it is required to establish a relationship between users and exchange zone information between users to be able to securely resolve names from other users. While in trust-based systems like the PGP's WoT a complicated seven step exchange and verification process is required, as described in Section 6.10.3, GNS provides a convenient way to introduce new delegation relationship between zones.

When a trusted secure online channel between users is already established, it is possible to exchange GNS information via this channel. GNS zone information can be easily exported as human-readable URI and can be imported using the `gnunet-uri` tool. A GNS URI contains the public of the zone to import and the desired nickname the user

specified. When using the `gnunet-uri` tool to import such an URI, the tool automatically creates the zone delegation for the new zone in GNS.

With GNS, we provide to encode GNS zone information in QR codes and exchange this information bound to an identification of the person as an out-band-band mechanism. An example for a business card containing a GNS QR-code is depicted in Figure 6.16. The QR-codes contains the GNUet URI and can be read using any standard QR reader. With `gnunet-qr`, we provide a tool to read QR code using a web cam and automatically invoke the `gnunet-uri` tool to import the information contained in the code.



Fig. 6.16: A Business Card Containing a GNS QR-Code

## 6.11 Conclusion and Findings

With the GNU Name System, we have presented a fully decentralized, censorship-resistant privacy-preserving name system and public key infrastructure. Already from the development of DNSSEC it seems logical that name systems and public key infrastructures should merge in the future. The PGP Web-of-Trust also already links identities, keys, and names (in the form of e-mail addresses). Having a unified system for name, identity, and key management is thus the next logical step. Specifically, we expect that the identity management functionality of the GNU Name System will result in names that are not merely used to address network services, but also individuals and their pseudonyms. Having such an integrated system will simplify the realization of end-to-end encrypted messaging systems, as keys can be easily linked to the names used for addressing.

As the world is not inherently hierarchical, it is logical that future name systems should depart from the restrictions imposed by a tree structure, and instead opt for directed graphs as they are more flexible and robust. That does not mean hierarchical addressing is dead. Users and services are still likely to be assigned names in the various hierarchies that they participate in. However, the same GNS zone may be reachable via multiple hierarchies, and users may choose to use different entry points into the various planned structures. For example, a community may build a hierarchy to address community services (`service.neighborhood.city.county.community`), but citizens of `city` are likely to establish a direct link with their local organization (`city`), making the operation of the local organization largely independent of that of `county` and `community`.

By encrypting the record data in the network, the GNU Name System enables applications that were previously impossible. In particular, by using passwords for the labels, the name system can now be used for authentication based on a shared secret. One possible application for this technique is TCP Stealth [Kir14], an extension of TCP where the user

## **206 6. GNS - A Decentralized, Privacy-Preserving and Censorship-Resistant Name System**

---

must authenticate himself within the first SYN packet. The GNU Name System could be used to easily communicate the shared secret to users that are aware of the name of the service, while hiding the existence of the TCP server from brute force attacks using port scanning. We expect further applications will be created based on this new possibility of communicating non-public information via the name system.

To have a system like GNS being adopted by users, it is important to not require users to learn how to use a system but just use it. When designing GNS, we focused on adapting the syntax for names known from DNS and to integrate the namespaces provided by GNS and DNS. At the moment integration of GNS with applications does not perfectly cover all possible use cases as we saw with integration in the name resolution process or web surfing. But we showed that it is possible to integrate GNS with existing applications and usage patterns. Operating systems and applications with native support for GNS and improved tools for users to use and administer GNS are therefore a desirable functionality for the future.

## 7. CONCLUSION AND FINDINGS

In this work, we have presented an architecture for building a decentralized and censorship-resistant overlay network as a foundation for future decentralized networking applications. The work was set out to explore the requirements to re-establish unrestricted end-to-end connectivity between users in a peer-to-peer network by realizing a communication infrastructure overcoming limitations with respect to restricted end-to-end connectivity and best-effort communication on today's Internet and being resistant against degradation and censorship attempts. Unrestricted communication between participants is in particular important for decentralized peer-to-peer networks as they strongly rely on the possibility to establish connections between participants and are often affected by service degradation attempts. In addition, this work has focused on how a secure, resilient, and censorship-resistant security infrastructure for the use with future secure Internet applications can be realized. Such an infrastructure is, in combination with resilient communication between users, important for the realization of future decentralized applications to allow users to refer to services and other information in a secure and privacy-preserving way. Existing hierarchical architectures have inherent problems causing issues with respect to the user's privacy and censorship-resistance of the system. This work sought to answer the following questions:

- What are restrictions to end-to-end connectivity and best-effort communication on today's Internet? How do adversaries try to prevent access to information and services on the Internet?
- Does the Internet provide a suitable foundation to re-establish end-to-end communication between users with a resilient peer-to-peer overlay?
- What are the requirements and a possible design for a secure and censorship-resistant communication infrastructure?
- How can a communication infrastructure support multiple communication protocols and optimally satisfy applications with a priori unknown requirements in a dynamic peer-to-peer environment?
- How can services and information be accessed in a resilient, decentralized and privacy-preserving way without relying on centralized authorities?

The main findings found in this work with respect to these questions are:

- A main restriction towards end-to-end connectivity between participants of a peer-to-peer network are approaches to mitigate the exhaustion of IPv4 addresses like NAT, CGN, or DS-Lite, often employed by ISPs and particularly affecting users in perimeter networks. In addition, security appliances can impose limitations to end-to-end connectivity. Traffic management and degradation attempts impact best-effort communication on today's Internet.

- Manipulation and censorship of the DNS is an easy to realize and therefore often employed approach to make services unavailable or remove information from the Internet. Name systems are a key service for the functioning of the Internet but current architectures are prone to censorship due to their design and names being owned by organizations. Efforts to extend DNS with source authentication and integrity do not help against an adversary modeled after a nation state.
- Security infrastructures, like the X.509 public key infrastructure, suffer from the same problems as hierarchical name systems and do not withstand an adversary as the one used with this work as they both rely on a hierarchical architecture. On today's Internet, these security infrastructures play an important to provide secure communication but are not suitable for the use with future decentralized applications and alternative approaches are required.
- The Internet can provide a suitable infrastructure as its routing infrastructure is surprisingly resilient against failures of networks and communication links. It requires a large numbers of networks or links to fail to partition the Internet and impact routing in a peer-to-peer overlay. Special focus has to be put on making communication resilient for perimeter networks, not as well connected as networks in the Internet's core.
- A design for a communication infrastructure has to focus on four aspects: increasing connectivity for peers in restricted environments, counteracting degradation attempts, providing communication in case of failing communication infrastructure, and allowing users to name services and other information in a secure way. Providing mechanisms to increase connectivity and detect degradation attempts and switching to an alternative mean of communication can be key approaches such an infrastructure can employ in combination with the use of a decentralized name system and security infrastructure to refer to entities on the network in a censorship-resistant way.
- A communication infrastructure has to pay respect to both requirements of higher-layer applications and communication properties when finding the optimal communication mechanism and distributing resources. Combining address selection and resource allocation in an integrated approach provides better solutions than solving both problems separately.
- Existing name systems and security infrastructures suffer from inherent problems due to their design and are not suitable for the use in decentralized peer-to-peer systems and as a foundation for future secure Internet applications. Privacy and security are main requirements for an alternative approach, not provided by current systems and or alternative approaches when using a strong adversary as the one used in this work. An alternative system can be based on human-memorable names which are not globally unique. The missing feature of uniqueness can be mitigated by linking local namespaces and additionally providing a namespace with absolute but not memorable names.
- An alternative security infrastructure should unify the functionalities of name systems and security infrastructures.

The communication infrastructure presented in this work tries to tackle both requirements to provide resilient communication between users and to provide a secure and

ensorship-resistant way to refer to services and information in the network. As with our communication infrastructure both application requirements as well as the tactics deployed by adversaries to free communication vary and can change over time, a key concept of our solution is using an extensible architecture: from pluggable transports to pluggable resource allocation strategies to pluggable record types in the name system. While we could demonstrate that the strategies developed in this thesis work as expected to address our pre-conceived notions of system requirements and adversarial behavior, the real strength is in the flexibility of the design. Like with TCP congestion control, we expect that the specific details will continue to evolve with the underlying network, application requirements and the adversary and therefore provide a system which provides the possibility to adapt to changes in the future.

## 7.1 Future Work

One challenge for designing realistic network experiments to guide future evolution of the presented methods will be to realize an approach for distributed experimentation and privacy-preserving and secure data collection in deployed distributed systems. With a decentralized system, we have the opportunity to test the system under realistic conditions in real-world environments and directly with the user. But deploying, conducting and controlling such experiments with a decentralized system, where participating nodes may be distributed and may not be under the developers direct control, rises challenges with respect to security and privacy and needs further investigation.

While the current system tries to detect and counteract network failures and degradation attempts, future work will also have to understand and analyze such events. Analyzing anomalies in decentralized system with a developer's, operator's, or user's restricted local view is hard to achieve. To make communication resilient, it is also required to understand the cause and the impact of anomalies in the underlying communication infrastructure used and also in the peer-to-peer overlay. Here a promising first approach is currently realized with the work done in [Tar14], presenting a design for a decentralized and resilient monitoring infrastructure for peer-to-peer networks which can serve as a foundation for future research on this topic.

For the pluggable transports, one area where we expect that future extensions may require non-trivial changes to the design and implementation is NAT traversal. We are aware of dozens of different NAT traversal techniques, and our current design and implementation only supports a small fraction. Here an extensible architecture to extend the infrastructure with new traversal approaches and to manage the traversal process is major focus for the future. This also includes usability as such an approach should work without user intervention or configuration.

An additional concern are existing, more generic systems for NAT traversal as many of those approaches neglect security and privacy issues. Security and privacy are major concerns on today's Internet and in particular for a communication infrastructure as the one presented in this work. Future work should therefore focus on providing secure, privacy-preserving, and user-friendly alternatives to current NAT traversal techniques and also pay attention to the requirements of decentralized networking applications.

Finally, a key issue this thesis does not investigate is usability. Our extensible design may be flexible enough to provide robust communication in theory, but in practice it also means that the system is complex to configure and use correctly. This is particularly true for the GNU Name System, which will always require a bit more involvement from end-users compared to DNS. Thus, one key challenge will be to hide this complexity from

the user and to, where possible, automatically configure the system to achieve usability, performance, and security.

## Bibliography

- [3rd11] D. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066 (Proposed Standard), January 2011.
- [AA04] D. Atkins and R. Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833 (Informational), August 2004.
- [AAL<sup>+</sup>05a] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), March 2005. Updated by RFCs 6014, 6840.
- [AAL<sup>+</sup>05b] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014, 6840.
- [AAL<sup>+</sup>05c] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014, 6840, 6944.
- [AAMS01] Z. Albanna, K. Almeroth, D. Meyer, and M. Schipper. IANA Guidelines for IPv4 Multicast Address Assignments. RFC 3171 (Best Current Practice), August 2001. Obsoleted by RFC 5771.
- [AKW09] Brice Augustin, Balachander Krishnamurthy, and Walter Willinger. Ixps: mapped? In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, IMC '09*, pages 336–349, New York, NY, USA, 2009. ACM.
- [Alp04] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2004.
- [Ano12] Anonymous. The collateral damage of internet censorship by dns injection. *ACM SIGCOMM Comp. Comm. Review*, 42(3):22–27, July 2012.
- [AvEM12] Hadi Asghari, Michel van Eeten, and Milton Mueller. Unraveling the economic and political drivers of deep packet inspection. Technical report, Delft University of Technology, November 2012.
- [AZH09] Riyadh Alshammari and A Nur Zincir-Heywood. Machine learning based encrypted traffic classification: identifying ssh and skype. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–8. IEEE, 2009.
- [BBC<sup>+</sup>98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475 (Informational), December 1998. Updated by RFC 3260.

- [BDL<sup>+</sup>11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 2011.
- [BDL<sup>+</sup>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.
- [Ber] Daniel J. Bernstein. Notes on the domain name system. <http://cr.yp.to/djbdns/notes.html>. accessed: 2014-07-7.
- [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. accessed: 2014-10-08.
- [Ber14] Michael Berkens. ICANN Votes To Kill .Amazon. <http://www.thedomains.com/2014/05/17/icann-votes-to-kills-amazon/>, May 2014. accessed: 2014-07-1.
- [BHMW08a] 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.
- [BHMW08b] R. Bless, C. Hübsch, S. Mies, and O.P. Waldhorst. The underlay abstraction in the spontaneous virtual networks (spovnet) architecture. In *Next Generation Internet Networks, 2008. NGI 2008*, pages 115 –122, April 2008.
- [BJ93] Thang Nguyen Bui and Curt Jones. A heuristic for reducing fill-in in sparse matrix factorization. In *PPSC*, pages 445–452, 1993.
- [BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.
- [BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (INTERNET STANDARD), January 2005. Updated by RFC 6874.
- [BLS10] George Dean Bissias, Brian Neil Levine, and Ramesh K. Sitaraman. Assessing the vulnerability of replicated network services. In *Proceedings of the 6th International Conference, Co-NEXT '10*, pages 24:1–24:12, New York, NY, USA, 2010. ACM.
- [BM95] S. Bradner and A. Mankin. The Recommendation for the IP Next Generation Protocol. RFC 1752 (Proposed Standard), January 1995.
- [Bon10] Paul S. Bonsma. Most balanced minimum cuts. *Discrete Applied Mathematics*, 158(4):261–276, 2010.

- [BZB<sup>+</sup>97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), September 1997. Updated by RFCs 2750, 3936, 4495, 5946, 6437, 6780.
- [CAG05] S. Cheshire, B. Aboba, and E. Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927 (Proposed Standard), May 2005.
- [CB02] B. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. RFC 3234 (Informational), February 2002.
- [CCG<sup>+</sup>02] Qian Chen, Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott Shenker, and Walter Willinger. The origin of power-laws in internet topologies revisited. In *INFOCOM*, 2002.
- [CET<sup>+</sup>11] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire. Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335 (Best Current Practice), August 2011.
- [CK13] S. Cheshire and M. Krochmal. Special-Use Domain Names. RFC 6761 (Proposed Standard), February 2013.
- [CM01] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), February 2001.
- [CNP11] G. Camarillo, O. Novo, and S. Perreault. Traversal Using Relays around NAT (TURN) Extension for IPv6. RFC 6156 (Proposed Standard), April 2011.
- [Coh08] Bram Cohen. Bep 3: The bittorrent protocol specification. [http://www.bittorrent.org/beps/bep\\_0003.html](http://www.bittorrent.org/beps/bep_0003.html), February 2008. accessed: 2014-06-04.
- [Cow14] Pam Cowburn. Org's blocked project finds almost 1 in 5 sites are blocked by filters. <https://www.openrightsgroup.org/blog/2014/blockedproject>, July 2014.
- [CR06] Rami Cohen and Danny Raz. The internet dark matter - on the missing links in the as connectivity map. In *INFOCOM*, 2006.
- [Cro] Douglas Crockford. Base32 encoding. <http://www.crockford.com/wrmg/base32.html>. accessed: 2014-07-10.
- [CV10] M. Cotton and L. Vegoda. Special Use IPv4 Addresses. RFC 5735 (Best Current Practice), January 2010. Obsoleted by RFC 6890, updated by RFC 6598.
- [DA99] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176.
- [Dan63] George Dantzig. *Linear programming and extensions*. Princeton Univ. Press, August 1963.

- [DCRS13] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer #38; Communications Security, CCS '13*, pages 61–72, New York, NY, USA, 2013. ACM.
- [DD95] Wolfgang Domschke and Andreas Drexler. *Einführung in die Operations Research*. Springer, 1995.
- [DDWL11] A. Durand, R. Droms, J. Woodyatt, and Y. Lee. Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion. RFC 6333 (Proposed Standard), August 2011.
- [Den14] Frank Denis. Dnscrypt protocol and implementation details. <https://github.com/jedisct1/dnscrypt-proxy/blob/master/TECHNOTES>, August 2014. accessed: 2014-09-17.
- [DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.
- [Din07] Roger Dingledine. Behavior for bridge users, bridge relays, and bridge authorities. <https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/125-bridges.txt>, November 2007. accessed: 2014-10-08.
- [DK12] Politi DK. Fejl blokerede internetsider kortvarigt. <https://www.politi.dk/da/aktuelt/nyheder/2012/Fejl+blokerer+internetsider+kortvarigt.htm>, March 2012. accessed: 2014-10-08.
- [DMG<sup>+</sup>10] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, Krishna P. Gummadi, Ratul Mahajan, and Stefan Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.
- [DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium*, August 2004.
- [DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, Berlin, Heidelberg, New York, 2002.
- [DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.
- [DSKM12] Casey Deccio, Jeff Sedayao, Krishna Kant, and Prasant Mohapatra. Quantifying dns namespace influence. *Comput. Netw.*, 56(2):780–794, February 2012.
- [EFL<sup>+</sup>99] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693 (Experimental), September 1999.

- [EG11] Nathan S. Evans and Christian Grothoff. R5n : Randomized recursive routing for restricted-route networks. In *5th International Conference on Network and System Security (NSS 2011)*, Milan, Italy, 09/2011 2011. IEEE, IEEE.
- [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.
- [EI96] Carl Ellison and Cybercash Inc. Establishing identity without certification authorities. In *6th USENIX Security Symposium*, pages 67–76, 1996.
- [Ell99] C. Ellison. SPKI Requirements. RFC 2692 (Experimental), September 1999.
- [Ess14] Loek Essers. German court finds domain registrar liable for torrent site's copyright infringement. <http://www.itworld.com/print/403869>, February 2014. accessed: 2014-10-08.
- [Eur11] European Parliament. Resolution on the EU-US Summit of 28 November 2011, November 2011. P7-RC-2011-0577.
- [Eva11] Nathan S. Evans. *Methods for Secure Decentralized Routing in Open Networks*. PhD thesis, Technische Universität München, Garching bei München, August 2011.
- [Far14a] Cyrus Farivar. After reaching 51% network power, Bitcoin mining pool says “trust us”. <http://arstechnica.com/security/2014/06/after-reaching-51-network-power-bitcoin-mining-pool-says-trust-us/>, June 2014. accessed: 2014-10-08.
- [Far14b] Cyrus Farivar. In sudden announcement, us to give up control of dns root zone. <http://arstechnica.com/tech-policy/2014/03/in-sudden-announcement-us-to-give-up-control-of-dns-root-zone/>, March 2014. accessed: 2014-07-25.
- [FGM<sup>+</sup>99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.
- [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.
- [FHE<sup>+</sup>12] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Dan Boneh, Roger Dingledine, and Phil Porras. Evading censorship with browser-based proxies. In Simone Fischer-Hübner and Matthew Wright, editors, *Privacy Enhancing Technologies*, volume 7384 of *Lecture Notes in Computer Science*, pages 239–258. Springer, 2012.
- [Fin01] Seth Finkelstein. Thoughts On Winning An EFF Pioneer Award. <http://www.spectacle.org/0401/sethfin.html>, March 2001. accessed: 2014-10-15.

- [Fiv14] Kelly Fiveash. France frostily foists flat fizz fear on icann's .wine plans. [http://www.theregister.co.uk/2014/06/23/winemakers\\_whine\\_on\\_about\\_icann/](http://www.theregister.co.uk/2014/06/23/winemakers_whine_on_about_icann/), June 2014. accessed: 2014-10-08.
- [FKK11] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), August 2011.
- [FM06] Uriel Feige and Mohammad Mahdian. Finding small balanced separators. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, pages 375–384, New York, NY, USA, 2006. ACM.
- [For03] UPnP Forum. UPnP device architecture 1.0, December 2003.
- [For08] Bryan Alexander Ford. *UIA: A Global Connectivity Architecture for Mobile Personal Devices*. PhD thesis, Massachusetts Institute of Technology, 2008.
- [Fou] Free Software Foundation. The GNU C Library - System Databases and Name Service Switch. [http://www.gnu.org/software/libc/manual/html\\_node/Name-Service-Switch.html](http://www.gnu.org/software/libc/manual/html_node/Name-Service-Switch.html). accessed: 2014-07-03.
- [FR14] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard), June 2014.
- [GA92] P. Gross and P. Almquist. IESG Deliberations on Routing and Addressing. RFC 1380 (Informational), November 1992.
- [Gol99] Yaron Y. Goland. Multicast and Unicast UDP HTTP Messages. draft-goland-http-udp-01.txt, November 1999.
- [GRW05] Stefan Götz, Simon Rieche, and Klaus Wehrle. *Selected DHT Algorithms*, volume 3485 of *Lecture Notes in Computer Science*, chapter 8, pages 95–117. Springer, 2005.
- [GSC<sup>+</sup>96] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889 (Proposed Standard), January 1996. Obsoleted by RFC 3550.
- [GWWA14] Christian Grothoff, Matthias Wachs, Hellekin Wolf, and Jacob Appelbaum. Special-Use Domain Names of Peer-to-Peer Systems. draft-grothoff-iesg-special-use-p2p-names-02, March 2014.
- [HA14] Selcan Hacaoglu and Onur Ant. Turkey Blocks Google DNS as Erdogan Defends Twitter Action . <http://www.bloomberg.com/news/2014-03-22/turkey-blocks-google-s-dns-after-move-against-twitter-hurriyet.html>, March 2014. accessed: 2014-10-08.
- [Hac09] M. Hachman. Sabotage Suspected in Silicon Valley Cable Cut. <http://www.pcmag.com/article/print/239065>, April 2009. accessed: 2014-10-08.
- [HD06] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), February 2006. Updated by RFCs 5952, 6052, 7136.

- [HHA<sup>+</sup>] Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, Matthew Luckie, kc claffy, and Colleen Shannon. The ipv4 routed /24 as links dataset - 12/30/2010,12/29/2010. [http://www.caida.org/data/active/ipv4\\_routed\\_topology\\_aslinks\\_dataset.xml](http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml).
- [HJ10] Erik Hjelmvik and Wolfgang John. Breaking and Improving Protocol Obfuscation. Technical report, Chalmers University of Technology, 2010.
- [HL95] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Supercomputing '95, New York, NY, USA, 1995. ACM.
- [HMNS98] N. Haller, C. Metz, P. Nesser, and M. Straw. A One-Time Password System. RFC 2289 (INTERNET STANDARD), February 1998.
- [Hol12] Martin Holland. Daenischer Polizist sperrt versehentlich 8000 Websites. <http://heise.de/-1447571>, March 2012. accessed: 2014-10-08.
- [HS12] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698 (Proposed Standard), August 2012. Updated by RFC 7218.
- [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.
- [HSF85] K. Harrenstien, M.K. Stahl, and E.J. Feinler. DoD Internet host table specification. RFC 952, October 1985. Updated by RFC 1123.
- [Hui06] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), February 2006. Updated by RFCs 5991, 6081.
- [IEE12] IEEE. IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, 2012.
- [IF02] K. Ingham and S. Forrest. A History and Survey of Network Firewalls. Technical report, University of New Mexico, 2002.
- [ISO94] ISO/IEC JTC 1. Information Technology — Open Systems Interconnection — Basic Reference Model: The Basic Model. ISO/IEC 7498-1:1994, ISO, Geneva, Switzerland, November 1994.
- [Jos06] S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648 (Proposed Standard), October 2006.
- [Kan14] Ruby Kannan. Uk gov compiling 'whitelist' to counter porn filter over-blocking. <http://www.ecotopiaproject.com/975640/uk-gov-compiles-whitelist-counter-porn-filter-blocking/index.htm>, February 2014. accessed: 2014-10-08.

- [Kar72] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
- [Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311, New York, NY, USA, 1984. ACM.
- [Kar12] Jiten Karia. Danish Police Accidentally Censor 8,000 Sites With Child Porn Filter. <http://www.techweekeurope.co.uk/news/danish-police-accidentally-censors-8000-sites-with-child-porn-filter-64365>, March 2012. accessed: 2014-10-08.
- [KE10] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational), May 2010.
- [Kel14] Simon Kelley. dnsmasq - A lightweight DHCP and caching DNS server. <http://manpages.ubuntu.com/manpages/trusty/man8/dnsmasq.8.html>, 2014.
- [Kir14] Julian Kirsch. Improved kernel-based port-knocking in linux. Master's thesis, Technische Universität München, August 2014.
- [KK98] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, December 1998.
- [KL70] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 49(1):291–307, 1970.
- [Kle47] Victor Klemperer. *LTI; Notizbuch eines Philologen*. Aufbau-Verlag, 1947.
- [KSS08] Jon Kleinberg, Mark Sandler, and Aleksandrs Slivkins. Network failure detection and graph connectivity. *SIAM J. Comput.*, 38:1330–1346, August 2008.
- [Lan12] Adam Langley. Revocation checking and chrome's crl. <http://www.imperialviolet.org/2012/02/05/crlsets.html>, February 2012. accessed: 2014-07-1.
- [LD60] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):pp. 497–520, 1960.
- [LGL<sup>+</sup>96] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. RFC 1928 (Proposed Standard), March 1996.
- [LMP<sup>+</sup>12] Patrick Lincoln, Ian Mason, Phillip Porras, Vinod Yegneswaran, Zachary Weinberg, Jeroen Massar, William Simpson, Paul Vixie, and Dan Boneh. Bootstrapping Communications into an Anti-Censorship System. In *Free and Open Communications on the Internet*, Bellevue, WA, USA, 2012. USENIX.
- [LP09] Yangyang Liu and Jianping Pan. The impact of nat on bittorrent-like p2p systems. In *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, pages 242–251, September 2009.

- [Ltd13] Netcraft Ltd. How certificate revocation (doesn't) work in practice. Blog entry at Netcraft blog: <http://news.netcraft.com/archives/2013/05/13/how-certificate-revocation-doesnt-work-in-practice.html>, May 2013. accessed: 2014-07-7.
- [LuTP06] Ludde, uau, The 8472, and PargNolar. Message stream encryption protocol version 1.0. [http://wiki.vuze.com/w/Message\\_Stream\\_Encryption](http://wiki.vuze.com/w/Message_Stream_Encryption), January 2006. accessed: 2014-06-04.
- [M13] Andreas Müller. *Analysis and Control of Middleboxes in the Internet*. PhD thesis, Technische Universität München, July 2013.
- [Mar06] Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351:394–406, February 2006.
- [Mar09] Stephen Marsland. *Machine Learning - An Algorithmic Perspective*. Chapman and Hall / CRC machine learning and pattern recognition series. CRC Press, 2009.
- [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.
- [McD94] D. McDonald. A Convention for Human-Readable 128-bit Keys. RFC 1751 (Informational), December 1994.
- [MCK08] A. Müller, G. Carle, and A Klenk. Behavior and classification of nat devices and implications for nat traversal. *Network, IEEE*, 22(5):14–19, September 2008.
- [MDD02] John Douceur Microsoft, John R. Douceur, and Judith S. Donath. The sybil attack. In *Lecture Notes in Computer Science*, pages 251–260, 2002.
- [MEGK10] Andreas Müller, Nathan S. Evans, Christian Grothoff, and Samy Kamkar. Autonomous nat traversal. In *10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P'10)*, Delft, The Netherlands, 2010. IEEE, IEEE.
- [Mey14] Stefan Mey. Ferrero und das .kinder-monopol im netz. <http://www.heise.de/newsticker/meldung/Ferrero-und-das-kinder-Monopol-im-Netz-2213681.html>, June 2014. accessed: 2014-10-08.
- [Mic09] Microsoft Technet. Understanding DNSSEC in Windows. <http://technet.microsoft.com/de-de/library/ee649277%28v=ws.10%29.aspx>, October 2009. accessed: 2014-10-08.
- [MMLDG12] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12*, pages 97–108, New York, NY, USA, 2012. ACM.
- [MMR10] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard), April 2010.

- [Moc87a] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (INTERNET STANDARD), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936.
- [Moc87b] 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.
- [Mog84] J.C. Mogul. Broadcasting Internet Datagrams. RFC 919 (INTERNET STANDARD), October 1984.
- [MSF11] Gregor Maier, Fabian Schneider, and Anja Feldmann. Nat usage in residential broadband networks. In Neil Spring and George F. Riley, editors, *Passive and Active Measurement*, volume 6579 of *Lecture Notes in Computer Science*, pages 32–41. Springer Berlin Heidelberg, 2011.
- [Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <http://bitcoin.org/bitcoin.pdf>, 2008. accessed: 2014-10-08.
- [NALB10] P. Natarajan, P. Amer, J. Leighton, and F. Baker. Using SCTP as a Transport Layer Protocol for HTTP. draft-natarajan-http-over-sctp-02.txt, January 2010.
- [NB98] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474 (Proposed Standard), December 1998. Updated by RFCs 3168, 3260.
- [NG05] E. Nordmark and R. Gilligan. Basic Transition Mechanisms for IPv6 Hosts and Routers. RFC 4213 (Proposed Standard), October 2005.
- [Nor09] Arvid Norberg. Bep 29: utorrent transport protocol. [http://www.bittorrent.org/beps/bep\\_00029.html](http://www.bittorrent.org/beps/bep_00029.html), June 2009. accessed: 2014-06-04.
- [NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.
- [Oeh14] Fabian Oehlmann. Machine learning for bandwidth management in decentralized networks. Master's thesis, Technische Universität München, Garching bei München, February 2014.
- [Orw46] George Orwell. *Politics and the English Language*. Horizon, 1946.
- [Par12] Brid-Aine Parnell. Epic net outage in Africa as FOUR undersea cables chopped: Ship blunders allegedly to blame. [http://www.theregister.co.uk/2012/02/28/east\\_africa\\_undersea\\_cables\\_cut/](http://www.theregister.co.uk/2012/02/28/east_africa_undersea_cables_cut/), February 2012. accessed: 2014-10-08.
- [PG14] Bartłomiej Polot and Christian Grothoff. Cadet: Confidential ad-hoc decentralized end-to-end transport. In *Med-Hoc-Net 2014*, June 2014.

- [PJ13] Colin Percival and Simon Josefsson. The scrypt password-based key derivation function - draft-josefsson-scrypt-kdf-01. <http://tools.ietf.org/id/draft-josefsson-scrypt-kdf-01.txt>, March 2013.
- [Pol10] Bart Polot. Adapting blackhat approaches to increase the resilience of white-hat application scenarios. Master's thesis, Technische Universität München, 2010.
- [Pos80] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980.
- [Pos81a] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. Updated by RFCs 1349, 2474, 6864.
- [Pos81b] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.
- [PPPW04] Kyoungsoo Park, Vivek S. Pai, Larry Peterson, and Zhe Wang. Codns: Improving dns performance and reliability via cooperative lookups. In *In Proceedings of the Sixth Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.
- [Pro40] Pierre-Joseph Proudhon. *What is Property? Or, an Inquiry into the Principle of Right and of Government*. Humboldt Publishing Company, 1st english translation, 1890 edition, 1840.
- [Pro12] Tor Project. Tor hidden service names. <https://trac.torproject.org/projects/tor/wiki/doc/HiddenServiceNames>, February 2012. accessed: 2014-10-08.
- [Ram99] B. Ramsdell. S/MIME Version 3 Message Specification. RFC 2633 (Proposed Standard), June 1999. Obsoleted by RFC 3851.
- [Raw14] Kevin Rawlinson. Turkey steps up bid to block Twitter after users flout ban. <http://www.theguardian.com/world/2014/mar/23/turkey-twitter-ban>, March 2014. accessed: 2014-10-08.
- [Res99] E. Rescorla. Diffie-Hellman Key Agreement Method. RFC 2631 (Proposed Standard), June 1999.
- [Res00] Certicom Research. Standards for efficient cryptography, SEC 1: Elliptic curve cryptography, September 2000. Version 1.0.
- [Ric05] M. Richardson. A Method for Storing IPsec Keying Material in DNS. RFC 4025 (Proposed Standard), March 2005.
- [RL96] Ronald L. Rivest and Butler Lampson. Sdsi - a simple distributed security infrastructure, 1996.
- [RLH06] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006. Updated by RFCs 6286, 6608, 6793.
- [RM12] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), January 2012.

- [RMK<sup>+</sup>96] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996. Updated by RFC 6761.
- [RMMW08] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), October 2008.
- [RMRW10] Amir H. Rasti, Nazanin Magharei, Reza Rejaie, and Walter Willinger. Eyeball ases: from geography to connectivity. In *Internet Measurement Conference*, pages 192–198, 2010.
- [Ros10] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Proposed Standard), April 2010. Updated by RFC 6336.
- [RTM08] Matthew Roughan, Simon Jonathan Tuke, and Olaf Maennel. Bigfoot, sasquatch, the yeti and other missing links: what we don't know about the as graph. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement, IMC '08*, pages 325–330, New York, NY, USA, 2008. ACM.
- [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. <https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/194-mnemonic-urls.txt>, February 2012. accessed: 2014-10-08.
- [Sau08] R. Sauder. Connecting The Many Undersea Cut Cable Dots. <http://www.rense.com/general80/cable.htm>, February 2008. accessed: 2014-10-08.
- [SB98] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [Sch12] Martin Schanzenbach. Design and implementation of a censorship resistant and fully decentralized name system. Master's thesis, Technische Universität München, Garching bei München, September 2012.
- [Sch13] Robert Schirmer. Evaluation of operation research approaches to solve allocation problems in decentralized networks. In *Proceedings of the Seminars Future Internet (FI)*, pages 91–99. Chair for Network Architectures and Services, Technische Universität München, 2013.
- [SCSR12] S. Sankararaman, J. Chen, L. Subramanian, and V. Ramasubramanian. Trickledns: Bootstrapping dns security using social trust. In *Communication Systems and Networks (COMSNETS)*, pages 1–10, 2012.
- [SG06] J. Schlyter and W. Griffin. Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints. RFC 4255 (Proposed Standard), January 2006.
- [SH99] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663 (Informational), August 1999.

- [Sha14] David Shamah. Israeli, US terror victims could 'own' Iran's Internet. <http://www.timesofisrael.com/israeli-us-terror-victims-now-own-irans-internet/>, June 2014. accessed: 2014-07-01.
- [Sin14] Supriti Singh. Experimental comparison of byzantine fault tolerant distributed hash tables. Master's thesis, Technische Universität München, Garching bei München, October 2014.
- [SJP<sup>+</sup>11] Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner. Bridgespa: Improving tor bridges with single packet authorization. In *WPES'11 - Proceedings of the Workshop on Privacy in the Electronic Society*, Chicago, IL, United States, 10/2011 2011. ACM, ACM.
- [SKW<sup>+</sup>98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher. In *in First Advanced Encryption Standard (AES) Conference*, 1998.
- [SM02] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, page 261–269, London, UK, 2002. Springer-Verlag, Springer-Verlag.
- [SMA<sup>+</sup>13] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960 (Proposed Standard), June 2013.
- [SNDW06] Atul Singh, Tsuen-Wan Ngan, Peter Druschel, and Dan S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *INFOCOM*. IEEE, 2006.
- [SRC81] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. In *ICDCS*, pages 509–512. IEEE Computer Society, 1981.
- [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*, March 2011.
- [Sta02] Richard M. Stallman. Free Software: Freedom and Cooperation. In Joshua Gay, editor, *Free Software, Free Society: Selected Essays of Richard M. Stallman*, chapter Free Software: Freedom and Cooperation, pages 157–189. GNU Press, Boston, 2002. <http://www.gnu.org/doc/book13.html>.
- [Sti05] Marc Stiegler. An introduction to petname systems. <http://www.skyhunter.com/marcs/petnames/IntroPetNames.html>, February 2005. accessed: 2014-10-08.
- [Swa11a] Aaron Swartz. Squaring the triangle: Secure, decentralized, human-readable names. <http://www.aaronsw.com/weblog/squarezooko>, January 2011. accessed: 2014-10-08.
- [Swa11b] Aaron Swartz. Squaring the triangle: Secure, decentralized, human-readable names: Frequently asked questions. <https://squaretriangle.jottit.com/faq>, January 2011. accessed: 2014-10-08.

- [TA12] National Telecommunications and Information Administration. Commerce department awards contract for management of key internet functions to icann. <http://www.ntia.doc.gov/press-release/2012/commerce-department-awards-contract-management-key-internet-functions-icann>, July 2012. accessed: 2014-10-08.
- [Tar14] Omar Tarabai. A decentralized and autonomous anomaly detection infrastructure for decentralized peer-to-peer networks. Master's thesis, Technische Universität München, October 2014.
- [Tea14] The GLPK Team. Gnu linear programming kit reference manual for glpk version 4.54. <http://ftp.gnu.org/gnu/glpk/glpk-4.54.tar.gz>, March 2014.
- [TGT08] F. Templin, T. Gleeson, and D. Thaler. Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). RFC 5214 (Informational), March 2008.
- [The] The Tor Project. Tor: Pluggable transports. <https://www.torproject.org/docs/pluggable-transports.html.en>. accessed: 2014-10-08.
- [The14] The Tor Project. How do i use pluggable transports? <https://trac.torproject.org/projects/tor/wiki/doc/TorBrowserBundle3FAQ#HowdoIusepluggabletransports>, April 2014. accessed: 2014-10-08.
- [THKS03] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. DNS Extensions to Support IP Version 6. RFC 3596 (Draft Standard), October 2003.
- [TNJ07] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), September 2007.
- [Tor13] TorrentFreak.com. Bittorrent traffic drops in america, grows in europe. <https://torrentfreak.com/bittorrent-traffic-drops-in-america-grows-in-europe-131111/>, November 2013. accessed: 2014-08-29.
- [Tot13] Gabor X Toth. Design of a social messaging system using stateful multicast. Master's, University of Amsterdam, Amsterdam, 2013.
- [TZL10] D. Thaler, L. Zhang, and G. Lebovitz. IAB Thoughts on IPv6 Network Address Translation. RFC 5902 (Informational), July 2010.
- [US96] International Telecommunication Union and International Telecommunication Union Telecommunication Standardization Sector. *Coding of Speech at 8 Kbit/s Using Conjugate-structure Algebraic-code-excited Linear-prediction (CS-ACELP): A silence compression scheme for G.729 optimized for terminals conforming to Recommendation V.70*. Number v. 70 in ITU-T recommendation. International Telecommunication Union, 1996.
- [VAC<sup>+</sup>08] Fabien Viger, Brice Augustin, Xavier Cuvellier, Clémence Magnien, Matthieu Latapy, Timur Friedman, and Renata Teixeira. Detection, understanding, and prevention of traceroute measurement artifacts. *Comput. Netw.*, 52:998–1018, April 2008.

- [vdV13] Lonneke van der Velden. *Unlike Us Reader*, chapter Meeting the Alternatives: Notes about making profiles and joining hackers, pages 312–322. Institute of Network Cultures, Amsterdam, 2013.
- [vK14] Anne van Kersteren. Cross-origin resource sharing. Technical report, W3c Working Draft 3, <http://www.w3.org/TR/cors/>, January 2014. accessed: 2014-10-08.
- [VTRB97] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC 2136 (Proposed Standard), April 1997. Updated by RFCs 3007, 4035, 4033, 4034.
- [WB11] M. Wasserman and F. Baker. IPv6-to-IPv6 Network Prefix Translation. RFC 6296 (Experimental), June 2011.
- [WGT12] Matthias Wachs, Christian Grothoff, and Ramakrishna Thurimella. Partitioning the internet. In *7th International Conference on Risks and Security of Internet and Systems (CRiSIS)*, Cork, Ireland, October 2012. IEEE Computer Society.
- [WKD<sup>+</sup>12] J. Weil, V. Kuarsingh, C. Donley, C. Liljenstolpe, and M. Azinger. IANA-Reserved IPv4 Prefix for Shared Address Space. RFC 6598 (Best Current Practice), April 2012.
- [WO01] Zooko Wilcox-O’Hearn. Names: Distributed, Secure, Human-Readable: Choose Two. <http://web.archive.org/web/20011020191610/http://zooko.com/distnames.html>, January 2001. accessed: 2014-10-08.
- [WOG14] Matthias Wachs, Fabian Oehlmann, and Christian Grothoff. Automatic transport selection and resource allocation for resilient communication in decentralized networks. In *IEEE Fourteenth International Conference on Peer-to-Peer Computing (P2P)*, IEEE, September 2014. IEEE, IEEE.
- [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)*, La Rochelle, France, October 2013. Springer Verlag, Springer Verlag.
- [WSG14] Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. On the feasibility of a censorship resistant decentralized name system. In *The 13th International Conference on Cryptology and Network Security (CANS 2014)*, Heraklion, Greece, October 2014. Springer Verlag, Springer Verlag.
- [XYK<sup>+</sup>08] Haiyong Xie, Y. Richard Yang, Arvind Krishnamurthy, Yanbin Grace Liu, and Abraham Silberschatz. P4p: Provider portal for applications. *SIGCOMM Comput. Commun. Rev.*, 38(4):351–362, August 2008.
- [ZH11] Bassam Zantout and Ramzi Haraty. I2p data communication system. In *Proceedings of ICN 2011, The Tenth International Conference on Networks*, January 2011.
- [ZJC11] P. Zimmermann, A. Johnston, and J. Callas. ZRTP: Media Path Key Agreement for Unicast Secure RTP. RFC 6189 (Informational), April 2011.

# List of Figures

2.1	Packet Filter Firewall Used as Gateway to Protect a Network Segment . . . . .	16
2.2	Classification of Different NAT Types According to [RWHM03] . . . . .	18
	(a) Full Cone NAT . . . . .	18
	(b) Port Restricted Cone NAT . . . . .	18
	(c) Restricted Cone NAT . . . . .	18
	(d) Symmetric NAT . . . . .	18
2.3	IPv6 Transition Mechanisms: Carrier-grade NAT (CGN) and Dual-Stack Lite (DS-Lite) . . . . .	21
	(a) Carrier-grade NAT (CGN) . . . . .	21
	(b) Dual-Stack Lite (DS-Lite) . . . . .	21
2.4	Comparison of Client/Server and Peer-to-Peer Architectures . . . . .	23
	(a) Client/Server Architecture . . . . .	23
	(b) Peer-to-Peer Network . . . . .	23
2.5	GNUnet's Layered Service Architecture . . . . .	25
3.1	Weight Distribution for Websites in Alexa's Top 100,000 Websites . . . . .	35
3.2	Weight Distribution in Terms of Page Views for ASes . . . . .	36
3.3	Size of the Computed Separators for the Unweighted AS Graph. . . . .	37
	(a) Size of the Edge Separator . . . . .	37
	(b) Size of the Node Separator . . . . .	37
3.4	Size of the Computed Separators for the Weighted AS Graph. . . . .	38
	(a) Size of the Edge Separator . . . . .	38
	(b) Size of the Node Separator . . . . .	38
4.1	Components of GNUnet's Transport Infrastructure . . . . .	44
4.2	The HOSTLIST Daemon . . . . .	49
4.3	The TRANSPORT Service . . . . .	54
4.4	The TRANSPORT Service Three Way Handshake . . . . .	64
4.5	The TRANSPORT Service State Machine . . . . .	66
4.6	GNUnet's HTTP(S) Transport . . . . .	76
4.7	GNUnet's HTTP(S) Transport Using a Reverse Proxy . . . . .	76
4.8	GNUnet's Distance Vector Routing Transport Plugin . . . . .	79
4.9	Comparison of TRANSPORT Performance with Different Plugins on a Local System . . . . .	84
4.10	Local Performance UNIX Plugin on a DELL Precision T3500 . . . . .	84
4.11	Local Performance TCP Plugin on a DELL Precision T3500 . . . . .	85
4.12	Local Performance UDP Plugin on a DELL Precision T3500 . . . . .	85
4.13	Local Performance UDP Plugin on a DELL Precision T3500: Impact of Fragmentation . . . . .	86
4.14	Local Performance HTTP Plugin on a DELL Precision T3500 . . . . .	86

4.15	Local Performance HTTP Plugin on a Lenovo T440s . . . . .	87
4.16	Local Performance HTTPS Plugin on a DELL Precision T3500 not supporting AES-NI . . . . .	87
4.17	Local Performance HTTPS Plugin on Lenovo T440s supporting AES-NI . . . . .	88
4.18	Performance Comparison Between Plugins over Gigabit Ethernet . . . . .	89
4.19	Performance TCP Plugin over Gigabit Ethernet . . . . .	90
4.20	Performance UDP Plugin over Gigabit Ethernet . . . . .	90
4.21	Performance HTTP Plugin over Gigabit Ethernet . . . . .	91
4.22	Performance HTTPS Plugin over Gigabit Ethernet . . . . .	91
4.23	Performance WLAN Plugin . . . . .	92
5.1	Impact of the Proportionality Factor $f_{prop}$ on Weights . . . . .	113
	(a) Impact of Proportionality Factor $f_{prop}$ on Weights for Exemplary Relative Weights $r_1, r_2, r_3$ . . . . .	113
	(b) Impact of Proportionality Factor $f_{prop}$ on Weights . . . . .	113
5.2	ATS Interaction with Applications and Transport Infrastructure . . . . .	127
5.3	ATS Service with Internal Structure and Interaction . . . . .	133
5.4	Performance of the Heuristic Solver . . . . .	136
	(a) Execution Time of the Heuristic Solver for a Full Solution . . . . .	136
	(b) Execution Time of the Heuristic Solver for an Incremental Solution . . . . .	136
5.5	Performance of the Mixed Integer Linear Programming Solver . . . . .	140
	(a) Execution Time of the Mixed Integer Linear Programming Solver for a Full Solution . . . . .	140
	(b) Execution Time of the Mixed Integer Linear Programming Solver for an Incremental Solution . . . . .	140
5.6	Performance of the Reinforcement Learning Solver . . . . .	144
	(a) Execution Time of the Reinforcement Learning Solver for a Full Solution . . . . .	144
	(b) Execution Time of the Reinforcement Learning Solver for an Incremental Solution . . . . .	144
5.7	Solution Quality of ATS Solvers in Different Scenarios . . . . .	147
6.1	DNSSEC Adoption in ccTLDs in March 2014 . . . . .	154
6.2	Validation Rate of DNSSEC Requests in September 2014 . . . . .	155
6.3	SDSI/SPKI : Principals, Certificates and Linked Local Name Spaces . . . . .	157
6.4	Zooko's Triangle: Design Space for Name Systems . . . . .	160
6.5	Integration of GNS and DNS Namespaces . . . . .	169
6.6	Network Format of GNS BOX Records . . . . .	175
6.7	GNS Components and Interaction . . . . .	182
6.8	Network Format of GNS Records . . . . .	184
6.9	Network Format of GNS Record Blocks . . . . .	185
6.10	Structure of Zones in GNS . . . . .	186
6.11	The GNS Resolution Process . . . . .	193
6.12	Using Relative GNS Names with the Web . . . . .	197
6.13	Using GNS Names with the HTTPS . . . . .	197
6.14	Screenshot of the Conversation User Interface . . . . .	202
6.15	GNS User Interface to Create TLSA Records . . . . .	204
6.16	A Business Card Containing a GNS QR-Code . . . . .	205

## List of Tables

3.1	Characterization of the AS Graphs Generated Using Route View's BGP Snapshot . . . . .	33
3.2	Characterization of the AS Graphs Generated Using CAIDA's Routed AS Links Dataset . . . . .	34
4.1	Performance of Cryptographic Cipher Suites on an Intel Xeon W3520 and i5-4200U . . . . .	82
5.1	Configuration Values for the Heuristic Solver . . . . .	135
5.2	Configuration Values for the MILP Solver . . . . .	139
5.3	Configuration Values for the RIL Solver . . . . .	143
5.4	Normalized Quality of the Solutions Produced by the Solvers . . . . .	147

## **APPENDIX**



## A. HEADERS

### A.1 The NAMESTORE Plugin API

**List. A.1:** The NAMESTORE Plugin API

```
/**
 * Function called by for each matching record.
 *
 * @param cls closure
 * @param zone_key private key of the zone
 * @param label name that is being mapped (at most 255 characters long)
 * @param rd_count number of entries in @a rd array
 * @param rd array of records with data to store
 */
typedef void (*GNUNET_NAMESTORE_RecordIterator) (void *cls,
const struct GNUNET_CRYPT0_EcdsaPrivateKey *private_key,
const char *label,
unsigned int rd_count,
const struct GNUNET_GNSRECORD_Data *rd);

/**
 * @brief struct returned by the initialization function of the plugin
 */
struct GNUNET_NAMESTORE_PluginFunctions
{
    /**
     * Closure to pass to all plugin functions.
     */
    void *cls;

    /**
     * Store a record in the datastore for which we are the authority.
     * Removes any existing record in the same zone with the same name.
     *
     * @param cls closure (internal context for the plugin)
     * @param zone private key of the zone
     * @param label name of the record in the zone
     * @param rd_count number of entries in @a rd array, 0 to delete all records
     * @param rd array of records with data to store
     * @return #GNUNET_OK on success, else #GNUNET_SYSERR
     */
    int (*store_records) (void *cls,
const struct GNUNET_CRYPT0_EcdsaPrivateKey *zone,
const char *label,
unsigned int rd_count,
const struct GNUNET_GNSRECORD_Data *rd);

    /**
     * Lookup records in the datastore for which we are the authority.
     *
     * @param cls closure (internal context for the plugin)
     * @param zone private key of the zone
     * @param label name of the record in the zone
     * @param iter function to call with the result
     * @param iter_cls closure for @a iter
     * @return #GNUNET_OK on success, else #GNUNET_SYSERR
     */
    int (*lookup_records) (void *cls,
const struct GNUNET_CRYPT0_EcdsaPrivateKey *zone,
const char *label,
GNUNET_NAMESTORE_RecordIterator iter, void *iter_cls);
};
```

```

/**
 * Iterate over the results for a particular zone in the
 * datastore. Will return at most one result to the iterator.
 *
 * @param cls closure (internal context for the plugin)
 * @param zone private key of the zone, NULL for all zones
 * @param offset offset in the list of all matching records
 * @param iter function to call with the result
 * @param iter_cls closure for @a iter
 * @return #GNUNET_OK on success, #GNUNET_NO if there were no results, #GNUNET_SYSERR on error
 */
int (*iterate_records) (void *cls,
    const struct GNUNET_CRYPT0_EcdsaPrivateKey *zone,
    uint64_t offset,
    GNUNET_NAMESTORE_RecordIterator iter, void *iter_cls);

/**
 * Look for an existing PKEY delegation record for a given public key.
 * Returns at most one result to the iterator.
 *
 * @param cls closure (internal context for the plugin)
 * @param zone private key of the zone to look up in, never NULL
 * @param value_zone public key of the target zone (value), never NULL
 * @param iter function to call with the result
 * @param iter_cls closure for @a iter
 * @return #GNUNET_OK on success, #GNUNET_NO if there were no results, #GNUNET_SYSERR on error
 */
int (*zone_to_name) (void *cls,
    const struct GNUNET_CRYPT0_EcdsaPrivateKey *zone,
    const struct GNUNET_CRYPT0_EcdsaPublicKey *value_zone,
    GNUNET_NAMESTORE_RecordIterator iter, void *iter_cls);
};

```

## A.2 The NAMECACHE Plugin API

**List. A.2:** The NAMECACHE Plugin API

```

/**
 * Function called for matching blocks.
 *
 * @param cls closure
 * @param block lookup result
 */
typedef void (*GNUNET_NAMECACHE_BlockCallback) (void *cls,
    const struct GNUNET_GNSRECORD_Block *block);

/**
 * @brief struct returned by the initialization function of the plugin
 */
struct GNUNET_NAMECACHE_PluginFunctions
{
    /**
     * Closure to pass to all plugin functions.
     */
    void *cls;

    /**
     * Cache a block in the datastore. Overwrites existing blocks
     * for the same zone and label.
     *
     * @param cls closure (internal context for the plugin)
     * @param block block to cache
     * @return #GNUNET_OK on success, else #GNUNET_SYSERR
     */
    int (*cache_block) (void *cls,
        const struct GNUNET_GNSRECORD_Block *block);

    /**

```

```
* Get the block for a particular zone and label in the
* datastore. Will return at most one result to the iterator.
*
* @param cls closure (internal context for the plugin)
* @param query hash of public key derived from the zone and the label
* @param iter function to call with the result
* @param iter_cls closure for @a iter
* @return #GNUNET_OK on success, #GNUNET_NO if there were no results, #GNUNET_SYSERR on error
*/
int (*lookup_block) (void *cls,
                    const struct GNUNET_HashCode *query,
                    GNUNET_NAMECACHE_BlockCallback iter, void *iter_cls);
};
```

ISBN 3-937201-45-9  
ISSN 1868-2634 (print)  
ISSN 1868-2642 (electronic)  
DOI: 10.2313/NET-2015-02-1