GNUnet Reference Manual

Installing, configuring, using and contributing to GNUnet

The GNUnet Developers

Edition 0.11.0pre66
1 January 1970

Copyright © 2001-2018 GNUnet e.V.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

A copy of the license is also available from the Free Software Foundation Web site at `http://www.gnu.org/licenses/fdl.html`.

Alternately, this document is also available under the General Public License, version 3 or later, as published by the Free Software Foundation. A copy of the license is included in the section entitled "GNU General Public License".

A copy of the license is also available from the Free Software Foundation Web site at `http://www.gnu.org/licenses/gpl.html`.

Short Contents

Introduction 1
1 Preface 2
2 Philosophy 4
3 Using GNUnet 10
4 GNUnet Contributors Handbook 59
5 GNUnet Developer Handbook 60
A GNU Free Documentation License 175
B GNU General Public License 183
Concept Index 194
Programming Index 196

Table of Contents

Introduction 1

1 Preface 2
1.1 About this book 2
1.2 Introduction 2
1.3 Project governance 3
1.4 General Terminology 3
1.5 Typography 3

2 Philosophy 4
2.1 Design Goals 4
2.2 Security and Privacy 4
2.3 Versatility 5
2.4 Practicality 5
2.5 Key Concepts 5
2.5.1 Authentication 5
2.5.2 Accounting to Encourage Resource Sharing 6
2.5.3 Confidentiality 7
2.5.4 Anonymity 7
2.5.4.1 How file-sharing achieves Anonymity 7
2.5.5 Deniability 8
2.5.6 Peer Identities 8
2.5.7 Zones in the GNU Name System (GNS Zones) 9
2.5.8 Egos 9

3 Using GNUnet 10
3.1 Checking the Installation 10
3.1.1 gnunet-gtk 10
3.1.2 Statistics 10
3.1.3 Peer Information 11
3.2 First steps - File-sharing 11
3.2.1 Publishing 11
3.2.2 Searching 12
3.2.3 Downloading 12
3.3 First steps - Using the GNU Name System 12
3.3.1 Preliminaries 13
3.3.2 Managing Egos 13
3.3.3 The GNS Tab 13
3.3.4 Creating a Record 13
3.3.5 Resolving GNS records 14
3.3.6 Integration with Browsers 14
3.3.7 Creating a Business Card 15

3.3.8 Be Social ... 16
3.3.9 Backup of Identities and Egos ... 16
3.3.10 Revocation ... 16
3.3.11 What's Next? ... 17
3.4 First steps - Using GNUnet Conversation ... 17
3.4.1 Testing your Audio Equipment ... 17
3.4.2 GNS Zones ... 18
3.4.2.1 Picking an Identity ... 18
3.4.2.2 Calling somebody ... 18
3.5 First steps - Using the GNUnet VPN ... 19
3.5.1 VPN Preliminaries ... 19
3.5.2 Exit configuration ... 19
3.5.3 GNS configuration ... 19
3.5.4 Accessing the service ... 20
3.5.5 Using a Browser ... 20
3.6 File-sharing ... 20
3.6.1 File-sharing Concepts ... 20
3.6.1.1 Files ... 21
3.6.1.2 Keywords ... 21
3.6.1.3 Directories ... 21
3.6.1.4 Pseudonyms ... 21
3.6.1.5 Namespaces ... 21
3.6.1.6 Advertisements ... 21
3.6.1.7 Anonymity level ... 22
3.6.1.8 Content Priority ... 22
3.6.1.9 Replication ... 22
3.6.2 File-sharing Publishing ... 22
3.6.2.1 Important command-line options ... 22
3.6.2.2 Indexing vs Inserting ... 23
3.6.3 File-sharing Searching ... 23
3.6.4 File-sharing Downloading ... 24
3.6.5 File-sharing Directories ... 24
3.6.6 File-sharing Namespace Management ... 24
3.6.6.1 Creating Pseudonyms ... 25
3.6.6.2 Deleting Pseudonyms ... 25
3.6.6.3 Advertising namespaces ... 25
3.6.6.4 Namespace names ... 25
3.6.6.5 Namespace root ... 25
3.6.7 File-Sharing URIs ... 25
3.6.7.1 Encoding of hash values in URIs ... 25
3.6.7.2 Content Hash Key (chk) ... 26
3.6.7.3 Location identifiers (loc) ... 26
3.6.7.4 Keyword queries (ksk) ... 26
3.6.7.5 Namespace content (sks) ... 26
3.7 The GNU Name System ... 26
3.7.1 Creating a Zone ... 27
3.7.2 Maintaining your own Zones ... 27
3.7.3 Obtaining your Zone Key ... 27

3.7.4 Adding Links to Other Zones ... 28
3.7.5 Using Public Keys as Top Level Domains ... 28
3.7.6 Resource Records in GNS ... 28
3.7.6.1 NICK ... 29
3.7.6.2 PKEY ... 29
3.7.6.3 BOX ... 29
3.7.6.4 LEHO ... 29
3.7.6.5 VPN ... 30
3.7.6.6 A AAAA and TXT ... 30
3.7.6.7 CNAME ... 30
3.7.6.8 GNS2DNS ... 30
3.7.6.9 SOA SRV PTR and MX ... 31
3.7.7 Synchronizing with legacy DNS ... 31
3.8 Using the Virtual Public Network ... 32
3.8.1 Setting up an Exit node ... 32
3.8.2 Fedora and the Firewall ... 33
3.8.3 Setting up VPN node for protocol translation and tunneling ... 33
3.9 The graphical configuration interface ... 34
3.9.1 Configuring your peer ... 34
3.9.2 Configuring the Friend-to-Friend (F2F) mode ... 35
3.9.3 Configuring the hostlist to bootstrap ... 35
3.9.4 Configuration of the HOSTLIST proxy settings ... 36
3.9.5 Configuring your peer to provide a hostlist ... 37
3.9.6 Configuring the datastore ... 37
3.9.7 Configuring the MySQL database ... 38
3.9.8 Reasons for using MySQL ... 38
3.9.9 Reasons for not using MySQL ... 38
3.9.10 Setup Instructions ... 38
3.9.11 Testing ... 38
3.9.12 Performance Tuning ... 39
3.9.13 Setup for running Testcases ... 39
3.9.14 Configuring the Postgres database ... 39
3.9.15 Reasons to use Postgres ... 39
3.9.16 Reasons not to use Postgres ... 39
3.9.17 Manual setup instructions ... 39
3.9.18 Testing the setup manually ... 40
3.9.19 Configuring the datacache ... 40
3.9.20 Configuring the file-sharing service ... 40
3.9.21 Configuring logging ... 41
3.9.22 Configuring the transport service and plugins ... 41
3.9.23 Configuring the wlan transport plugin ... 43
3.9.23.1 Requirements for the WLAN plugin ... 43
3.9.23.2 Configuration ... 43
3.9.23.3 Before starting GNUnet ... 44
3.9.23.4 Limitations and known bugs ... 44
3.9.24 Configuring HTTP(S) reverse proxy functionality using Apache or nginx ... 44

3.9.24.1 Reverse Proxy - Configure your Apache2 HTTP webserver . . . 45
3.9.24.2 Reverse Proxy - Configure your Apache2 HTTPS webserver . . . 45
3.9.24.3 Reverse Proxy - Configure your nginx HTTPS webserver . . . 45
3.9.24.4 Reverse Proxy - Configure your nginx HTTP webserver . . 46
3.9.24.5 Reverse Proxy - Configure your GNUnet peer . . . 46
3.9.25 Blacklisting peers . . . 46
3.9.26 Configuration of the HTTP and HTTPS transport plugins . . 47
3.9.27 Configuring the GNU Name System . . . 48
3.9.27.1 Configuring system-wide DNS interception . . . 48
3.9.27.2 Configuring the GNS nsswitch plugin . . . 49
3.9.27.3 Configuring GNS on W32 . . . 49
3.9.27.4 GNS Proxy Setup . . . 50
3.9.27.5 Setup of the GNS CA . . . 50
3.9.27.6 Testing the GNS setup . . . 51
3.9.28 Configuring the GNUnet VPN . . . 51
3.9.28.1 IPv4 address for interface . . . 51
3.9.28.2 IPv6 address for interface . . . 52
3.9.28.3 Configuring the GNUnet VPN DNS . . . 52
3.9.28.4 Configuring the GNUnet VPN Exit Service . . . 52
3.9.28.5 IP Address of external DNS resolver . . . 52
3.9.28.6 IPv4 address for Exit interface . . . 52
3.9.28.7 IPv6 address for Exit interface . . . 53
3.9.29 Bandwidth Configuration . . . 53
3.9.30 Configuring NAT . . . 53
3.9.31 Peer configuration for distributions . . . 54
3.10 How to start and stop a GNUnet peer . . . 54
3.10.1 The Single-User Setup . . . 55
3.10.2 The Multi-User Setup . . . 55
3.10.3 Killing GNUnet services . . . 56
3.10.4 Access Control for GNUnet . . . 56
3.10.4.1 Recommendation - Disable access to services via TCP . . 57
3.10.4.2 Recommendation - Run most services as system user "gnunet" . . . 57
3.10.4.3 Recommendation - Control access to services using group "gnunet" . . . 57
3.10.4.4 Recommendation - Limit access to certain SUID binaries by group "gnunet" . . . 58
3.10.4.5 Recommendation - Limit access to critical gnunet-helper-dns to group "gnunetdns" . . . 58
3.10.4.6 Differences between "make install" and these recommendations . . . 58

4 GNUnet Contributors Handbook 59

4.1 Contributing to GNUnet 59
4.2 Licenses of contributions 59
4.3 Copyright Assignment 59
4.4 Contributing to the Reference Manual 59

5 GNUnet Developer Handbook 60

5.1 Developer Introduction 60
5.1.1 Project overview 61
5.2 Internal dependencies 62
5.3 Code overview 64
5.4 System Architecture 68
5.5 Subsystem stability 68
5.6 Naming conventions and coding style guide 70
5.6.1 Naming conventions 70
5.6.1.1 include files 70
5.6.1.2 binaries 70
5.6.1.3 logging 71
5.6.1.4 configuration 71
5.6.1.5 exported symbols 71
5.6.1.6 private (library-internal) symbols (including structs and macros) 71
5.6.1.7 testcases 71
5.6.1.8 performance tests 72
5.6.1.9 src/ directories 72
5.6.2 Coding style 72
5.7 Build-system 75
5.8 Developing extensions for GNUnet using the gnunet-ext template 75
5.9 Writing testcases 75
5.10 Building GNUnet and its dependencies 76
5.11 TESTING library 80
5.11.1 API 80
5.11.2 Finer control over peer stop 81
5.11.3 Helper functions 81
5.11.4 Testing with multiple processes 82
5.12 Performance regression analysis with Gauger 82
5.13 TESTBED Subsystem 83
5.13.1 Supported Topologies 85
5.13.2 Hosts file format 86
5.13.3 Topology file format 87
5.13.4 Testbed Barriers 87
5.13.4.1 Implementation 88
5.13.5 Automatic large-scale deployment in the PlanetLab testbed 89
5.13.5.1 PlanetLab Automation for Fedora8 nodes 89
5.13.5.2 Install buildslave on PlanetLab nodes running fedora core 8 89
5.13.5.3 Setup a new PlanetLab testbed using GPLMT 89

5.13.5.4 Why do i get an ssh error when using the regex profiler? 90
5.13.6 TESTBED Caveats 91
5.13.6.1 CORE must be started 91
5.13.6.2 ATS must want the connections 91
5.14 libgnunetutil 91
5.14.1 Logging 92
5.14.1.1 Examples 95
5.14.1.2 Log files 96
5.14.1.3 Updated behavior of GNUNET_log 96
5.14.2 Interprocess communication API (IPC) 97
5.14.2.1 Define new message types 97
5.14.2.2 Define message struct 98
5.14.2.3 Client - Establish connection 98
5.14.2.4 Client - Initialize request message 98
5.14.2.5 Client - Send request and receive response 98
5.14.2.6 Server - Startup service 99
5.14.2.7 Server - Add new handles for specified messages 99
5.14.2.8 Server - Process request message 100
5.14.2.9 Server - Response to client 101
5.14.2.10 Server - Notification of clients 101
5.14.2.11 Conversion between Network Byte Order (Big Endian) and Host Byte Order 102
5.14.3 Cryptography API 103
5.14.4 Message Queue API 103
5.14.5 Service API 105
5.14.6 Optimizing Memory Consumption of GNUnet's (Multi-) Hash Maps 106
5.14.6.1 Analysis 106
5.14.6.2 Solution 107
5.14.6.3 Migration 107
5.14.6.4 Conclusion 108
5.14.6.5 Availability 108
5.14.7 CONTAINER_MDLL API 108
5.15 Automatic Restart Manager (ARM) 109
5.15.1 Basic functionality 109
5.15.2 Key configuration options 110
5.15.3 ARM - Availability 111
5.15.4 Reliability 111
5.16 TRANSPORT Subsystem 112
5.16.1 Address validation protocol 112
5.17 NAT library 113
5.18 Distance-Vector plugin 114
5.19 SMTP plugin 115
5.19.1 Why use SMTP for a peer-to-peer transport? 115
5.19.2 How does it work? 116
5.19.3 How do I configure my peer? 116
5.19.4 How do I test if it works? 117

5.19.5 How fast is it? 117
5.20 Bluetooth plugin 118
5.20.1 What do I need to use the Bluetooth plugin transport? .. 118
5.20.2 How does it work2? 119
5.20.3 What possible errors should I be aware of? 119
5.20.4 How do I configure my peer2? 120
5.20.5 How can I test it? 120
5.20.6 The implementation of the Bluetooth transport plugin .. 121
5.20.6.1 Linux functionality 121
5.20.6.2 THE INITIALIZATION 121
5.20.6.3 THE LOOP 122
5.20.6.4 Details about the broadcast implementation 122
5.20.6.5 Windows functionality 123
5.20.6.6 Pending features 124
5.21 WLAN plugin 124
5.22 ATS Subsystem 124
5.23 CORE Subsystem 124
5.23.1 Limitations 125
5.23.2 When is a peer "connected"? 126
5.23.3 libgnunetcore 126
5.23.4 The CORE Client-Service Protocol 127
5.23.4.1 Setup2 127
5.23.4.2 Notifications 128
5.23.4.3 Sending 128
5.23.5 The CORE Peer-to-Peer Protocol 128
5.23.5.1 Creating the EphemeralKeyMessage 128
5.23.5.2 Establishing a connection 129
5.23.5.3 Encryption and Decryption 129
5.23.5.4 Type maps 129
5.24 CADET Subsystem 130
5.24.1 libgnunetcadet 131
5.25 NSE Subsystem 132
5.25.1 Motivation 132
5.25.1.1 Security 132
5.25.2 Principle 132
5.25.2.1 Example 133
5.25.2.2 Algorithm 133
5.25.2.3 Target value 133
5.25.2.4 Timing 133
5.25.2.5 Controlled Flooding 133
5.25.2.6 Calculating the estimate 134
5.25.3 libgnunetnse 134
5.25.3.1 Results 134
5.25.3.2 libgnunetnse -Examples 135
5.25.4 The NSE Client-Service Protocol 135
5.25.5 The NSE Peer-to-Peer Protocol 135
5.26 HOSTLIST Subsystem 136
5.26.1 HELLOs 136

5.26.2 Overview for the HOSTLIST subsystem 136
5.26.2.1 Features 137
5.26.2.2 HOSTLIST - Limitations 137
5.26.3 Interacting with the HOSTLIST daemon 137
5.26.4 Hostlist security address validation 137
5.26.5 The HOSTLIST daemon 138
5.26.6 The HOSTLIST server 138
5.26.6.1 The HTTP Server 138
5.26.6.2 Advertising the URL 139
5.26.7 The HOSTLIST client 139
5.26.7.1 Bootstrapping 139
5.26.7.2 Learning 139
5.26.8 Usage 140
5.27 IDENTITY Subsystem 140
5.27.1 libgnunetidentity 140
5.27.1.1 Connecting to the service 140
5.27.1.2 Operations on Egos 141
5.27.1.3 The anonymous Ego 141
5.27.1.4 Convenience API to lookup a single ego 141
5.27.1.5 Associating egos with service functions 142
5.27.2 The IDENTITY Client-Service Protocol 142
5.28 NAMESTORE Subsystem 142
5.28.1 libgnunetnamestore 143
5.28.1.1 Editing Zone Information 143
5.28.1.2 Iterating Zone Information 143
5.28.1.3 Monitoring Zone Information 144
5.29 PEERINFO Subsystem 144
5.29.1 PEERINFO - Features 144
5.29.2 PEERINFO - Limitations 144
5.29.3 DeveloperPeer Information 144
5.29.4 Startup 145
5.29.5 Managing Information 145
5.29.6 Obtaining Information 145
5.29.7 The PEERINFO Client-Service Protocol 145
5.29.8 libgnunetpeerinfo 146
5.29.8.1 Connecting to the PEERINFO Service 146
5.29.8.2 Adding Information to the PEERINFO Service 146
5.29.8.3 Obtaining Information from the PEERINFO Service . . 146
5.30 PEERSTORE Subsystem 147
5.30.1 Functionality 147
5.30.2 Architecture 147
5.30.3 libgnunetpeerstore 147
5.31 SET Subsystem 148
5.31.1 Local Sets 148
5.31.2 Set Modifications 148
5.31.3 Set Operations 149
5.31.4 Result Elements 149
5.31.5 libgnunetset 149

5.31.5.1 Sets ... 149
5.31.5.2 Listeners ... 149
5.31.5.3 Operations ... 149
5.31.5.4 Supplying a Set ... 150
5.31.5.5 The Result Callback ... 150
5.31.6 The SET Client-Service Protocol ... 150
5.31.6.1 Creating Sets ... 150
5.31.6.2 Listeners2 ... 150
5.31.6.3 Initiating Operations ... 150
5.31.6.4 Modifying Sets ... 150
5.31.6.5 Results and Operation Status ... 150
5.31.6.6 Iterating Sets ... 151
5.31.7 The SET Intersection Peer-to-Peer Protocol ... 151
5.31.7.1 The Bloom filter exchange ... 151
5.31.7.2 Salt ... 151
5.31.8 The SET Union Peer-to-Peer Protocol ... 152
5.32 STATISTICS Subsystem ... 152
5.32.1 libgnunetstatistics ... 153
5.32.1.1 Statistics retrieval ... 153
5.32.1.2 Setting statistics and updating them ... 154
5.32.1.3 Watches ... 154
5.32.2 The STATISTICS Client-Service Protocol ... 154
5.32.2.1 Statistics retrieval2 ... 154
5.32.2.2 Setting and updating statistics ... 154
5.32.2.3 Watching for updates ... 155
5.33 Distributed Hash Table (DHT) ... 155
5.33.1 Block library and plugins ... 155
5.33.1.1 What is a Block? ... 155
5.33.1.2 The API of libgnunetblock ... 156
5.33.1.3 Queries ... 156
5.33.1.4 Sample Code ... 156
5.33.1.5 Conclusion2 ... 156
5.33.2 libgnunetdht ... 157
5.33.2.1 GET ... 157
5.33.2.2 PUT ... 157
5.33.2.3 MONITOR ... 157
5.33.2.4 DHT Routing Options ... 158
5.33.3 The DHT Client-Service Protocol ... 158
5.33.3.1 PUTting data into the DHT ... 158
5.33.3.2 GETting data from the DHT ... 158
5.33.3.3 Monitoring the DHT ... 159
5.33.4 The DHT Peer-to-Peer Protocol ... 159
5.33.4.1 Routing GETs or PUTs ... 159
5.33.4.2 PUTting data into the DHT2 ... 160
5.33.4.3 GETting data from the DHT2 ... 160
5.34 GNU Name System (GNS) ... 160
5.34.1 libgnunetgns ... 161
5.34.1.1 Looking up records ... 161

5.34.1.2 Accessing the records 162
5.34.1.3 Creating records 162
5.34.1.4 Future work 162
5.34.2 libgnunetgnsrecord 162
5.34.2.1 Value handling 163
5.34.2.2 Type handling 163
5.34.3 GNS plugins 163
5.34.4 The GNS Client-Service Protocol 163
5.34.5 Hijacking the DNS-Traffic using gnunet-service-dns 164
5.34.5.1 Network Setup Details 164
5.34.6 Serving DNS lookups via GNS on W32 164
5.35 GNS Namecache 165
5.35.1 libgnunetnamecache 166
5.35.2 The NAMECACHE Client-Service Protocol 166
5.35.2.1 Lookup 166
5.35.2.2 Store 166
5.35.3 The NAMECACHE Plugin API 166
5.35.3.1 Lookup2 167
5.35.3.2 Store2 167
5.36 REVOCATION Subsystem 167
5.36.1 Dissemination 167
5.36.2 Revocation Message Design Requirements 167
5.36.3 libgnunetrevocation 168
5.36.3.1 Querying for revoked keys 168
5.36.3.2 Preparing revocations 168
5.36.3.3 Issuing revocations 168
5.36.4 The REVOCATION Client-Service Protocol 168
5.36.5 The REVOCATION Peer-to-Peer Protocol 169
5.37 File-sharing (FS) Subsystem 169
5.37.1 Encoding for Censorship-Resistant Sharing (ECRS) 170
5.37.1.1 Namespace Advertisements 170
5.37.1.2 KSBlocks 170
5.37.2 File-sharing persistence directory structure 170
5.38 REGEX Subsystem 171
5.38.1 How to run the regex profiler 172
5.39 REST Subsystem 173
5.39.1 Namespace considerations 174
5.39.2 Endpoint documentation 174

Appendix A GNU Free Documentation License .. 175

Appendix B GNU General Public License 183

Concept Index 194

Programming Index 196

Introduction

This document is the Reference Manual for GNUnet version 0.11.0pre66.

1 Preface

This collection of manuals describes how to use GNUnet, a framework for secure peer-to-peer networking with the high-level goal to provide a strong foundation Free Software for a global, distributed network that provides security and privacy. GNUnet in that sense aims to replace the current Internet protocol stack. Along with an application for secure publication of files, it has grown to include all kinds of basic applications for the foundation of a new Internet.

1.1 About this book

The books (described as "book" or "books" in the following) bundled as the "GNUnet Reference Manual" are based on the historic work of all contributors to GNUnet's documentation. The documentation existed in various formats before it came to be in the format you are currently reading. It is our hope that the content is described in a way that does not require any academic background, although some concepts will require further reading.

Our (long-term) goal with these books is to keep them self-contained. If you see references to Wikipedia and other external sources (except for our academic papers) it means that we are working on a solution to describe the explanations found there which fits our use-case and licensing.

The first chapter ("Preface") as well as the the second chapter ("Philosophy") give an introduction to GNUnet as a project, what GNUnet tries to achieve.

1.2 Introduction

GNUnet in its current version is the result of almost 20 years of work from many contributors. So far, most contributions were made by volunteers or people paid to do fundamental research. Thus, significant parts of the software lack a reasonable degree of professionalism in its implementation. Furthermore, we are aware of a significant number of existing bugs and critical design flaws, as some unfortunate early design decisions remain to be rectified. There are still known open problems; GNUnet remains an active research project.

The project was started in 2001 when some initial ideas for improving Freenet's file-sharing turned out to be too radical to be easily realized within the scope of the existing Freenet project. We lost our first contributor on 11.9.2001 as the contributor realized that privacy may help terrorists. The rest of the team concluded that it was now even more important to fight for civil liberties. The first release was called "GNet" – already with the name GNUnet in mind, but without the blessing of GNU we did not dare to call it GNUnet immediately. A few months after the first release we contacted the GNU project, happily agreed to their governance model and became an official GNU package.

Within the first year, we created GNU libextractor, a helper library for meta data extraction which has been used by a few other projects as well. 2003 saw the emergence of pluggable transports, the ability for GNUnet to use different mechanisms for communication, starting with TCP, UDP and SMTP (support for the latter was later dropped due to a lack of maintenance). In 2005, the project first started to evolve beyond the original file-sharing application with a first simple P2P chat. In 2007, we created GNU libmicrohttpd to support a pluggable transport based on HTTP. In 2009, the architecture was radically modularized into the multi-process system that exists today. Coincidentally,

the first version of the ARM service was implemented a day before systemd was announced. From 2009 to 2014 work progressed rapidly thanks to a significant research grant from the Deutsche Forschungsgesellschaft. This resulted in particular in the creation of the R5N DHT, CADET, ATS and the GNU Name System. In 2010, GNUnet was selected as the basis for the SecuShare online social network, resutling in a significant growth of the core team. In 2013, we launched GNU Taler to address the challenge of convenient and privacy-preserving online payments. In 2015, the pEp project announced that they will use GNUnet as the technology for their meta-data protection layer, ultimately resulting in GNUnet e.V. entering into a formal long-term collaboration with the pEp foundation. In 2016, Taler Systems SA, a first startup using GNUnet technology, was founded with support from the community.

GNUnet is not merely a technical project, but also a political mission: like the GNU project as a whole, we are writing software to achieve political goals with a focus on the human right of informational self-determination. Putting users in control of their computing has been the core driver of the GNU project. With GNUnet we are focusing on informational self-determination for collaborative computing and communication over networks.

The Internet is shaped as much by code and protocols as by its associated political processes (IETF, ICANN, IEEE, etc.), and its flaws are similarly not limited to the protocol design. Thus, technical excellence by itself will not suffice to create a better network. We also need to build a community that is wise, humble and has a sense of humor to achieve our goal to create a technical foundation for a society we would like to live in.

1.3 Project governance

GNUnet, like the GNU project and many other free software projects, follows the governance model of a benevolent dictator. This means that ultimately, the GNU project appoints the GNU maintainer and can overrule decisions made by the GNUnet maintainer. Similarly, the GNUnet maintainer can overrule any decisions made by individual developers. Still, in practice neither has happened in the last 20 years, and we hope to keep it that way.

The GNUnet project is supported by GNUnet e.V., a German association where any developer can become a member. GNUnet e.V. servers as a legal entity to hold the copyrights to GNUnet. GNUnet e.V. may also choose to pay for project resources, and can collect donations. GNUnet e.V. may also choose to adjust the license of the software (with the constraint that it has to remain free software).

1.4 General Terminology

In the following manual we may use words that can not be found in the Appendix. Since we want to keep the manual selfcontained, we will explain words here.

1.5 Typography

When giving examples for commands, shell prompts are used to show if the command should/can be issued as root, or if "normal" user privileges are sufficient. We use a `#` for root's shell prompt, a `%` for users' shell prompt, assuming they use the C-shell or tcsh and a `$` for bourne shell and derivatives.

2 Philosophy

The foremost goal of the GNUnet project is to become a widely used, reliable, open, non-discriminating, egalitarian, unconstrained and censorship-resistant system of free information exchange. We value free speech above state secrets, law-enforcement or intellectual monopoly. GNUnet is supposed to be an anarchistic network, where the only limitation for participants (devices or people making use of the network, in the following sometimes called peers) is that they must contribute enough back to the network such that their resource consumption does not have a significant impact on other users. GNUnet should be more than just another file-sharing network. The plan is to offer many other services and in particular to serve as a development platform for the next generation of Internet Protocols.

2.1 Design Goals

These are the core GNUnet design goals, in order of relative importance:

- GNUnet must be implemented as Free Software (`https://www.gnu.org/philosophy/free-sw.html`)[1]
- GNUnet must only disclose the minimal amount of information necessary.
- GNUnet must be fully distributed and survive Byzantine failures (`https://en.wikipedia.org/wiki/Byzantine_fault_tolerance`)[2] at any position in the network.
- GNUnet must make it explicit to the user which entities are considered to be trustworthy when establishing secured communications.
- GNUnet must use compartmentalization to protect sensitive information.
- GNUnet must be open and permit new peers to join.
- GNUnet must be self-organizing and not depend on administrators.
- GNUnet must support a diverse range of applications and devices.
- The GNUnet architecture must be cost effective.
- GNUnet must provide incentives for peers to contribute more resources than they consume.

2.2 Security and Privacy

GNUnet's primary design goals are to protect the privacy of its users and to guard itself against attacks or abuse. GNUnet does not have any mechanisms to control, track or censor users. Instead, the GNUnet protocols aim to make it as hard as possible to find out what is happening on the network or to disrupt operations.

---

[1] This means that you you have the four essential freedoms: to run the program, to study and change the program in source code form, to redistribute exact copies, and to distribute modified versions. Refer to https://www.gnu.org/philosophy/free-sw.html (`https://www.gnu.org/philosophy/free-sw.html`)

[2] https://en.wikipedia.org/wiki/Byzantine_fault_tolerance (`https://en.wikipedia.org/wiki/Byzantine_fault_tolerance`)

2.3 Versatility

We call GNUnet a peer-to-peer framework because we want to support many different forms of peer-to-peer applications. GNUnet uses a plugin architecture to make the system extensible and to encourage code reuse. While the first versions of the system only supported anonymous file-sharing, other applications are being worked on and more will hopefully follow in the future. A powerful synergy regarding anonymity services is created by a large community utilizing many diverse applications over the same software infrastructure. The reason is that link encryption hides the specifics of the traffic for non-participating observers. This way, anonymity can get stronger with additional (GNUnet) traffic, even if the additional traffic is not related to anonymous communication. Increasing anonymity is the primary reason why GNUnet is developed to become a peer-to-peer framework where many applications share the lower layers of an increasingly complex protocol stack. If merging traffic to hinder traffic analysis was not important, we could have just developed a dozen stand-alone applications and a few shared libraries.

2.4 Practicality

GNUnet allows participants to trade various amounts of security in exchange for increased efficiency. However, it is not possible for any user's security and efficiency requirements to compromise the security and efficiency of any other user.

For GNUnet, efficiency is not paramount. If there were a more secure and still practical approach, we would choose to take the more secure alternative. `telnet` is more efficient than `ssh`, yet it is obsolete. Hardware gets faster, and code can be optimized. Fixing security issues as an afterthought is much harder.

While security is paramount, practicability is still a requirement. The most secure system is always the one that nobody can use. Similarly, any anonymous system that is extremely inefficient will only find few users. However, good anonymity requires a large and diverse user base. Since individual security requirements may vary, the only good solution here is to allow individuals to trade-off security and efficiency. The primary challenge in allowing this is to ensure that the economic incentives work properly. In particular, this means that it must be impossible for a user to gain security at the expense of other users. Many designs (e.g. anonymity via broadcast) fail to give users an incentive to choose a less secure but more efficient mode of operation. GNUnet should avoid where ever possible to rely on protocols that will only work if the participants are benevolent. While some designs have had widespread success while relying on parties to observe a protocol that may be sub-optimal for the individuals (e.g. TCP Nagle), a protocol that ensures that individual goals never conflict with the goals of the group is always preferable.

2.5 Key Concepts

In this section, the fundamental concepts of GNUnet are explained. Most of them are also described in our research papers. First, some of the concepts used in the GNUnet framework are detailed. The second part describes concepts specific to anonymous file-sharing.

2.5.1 Authentication

Almost all peer-to-peer communications in GNUnet are between mutually authenticated peers. The authentication works by using ECDHE, that is a DH (Diffie—Hellman) key

exchange using ephemeral eliptic curve cryptography. The ephemeral ECC (Eliptic Curve Cryptography) keys are signed using ECDSA (ECDSA (`http://en.wikipedia.org/wiki/ECDSA`)). The shared secret from ECDHE is used to create a pair of session keys (using HKDF) which are then used to encrypt the communication between the two peers using both 256-bit AES (Advanced Encryption Standard) and 256-bit Twofish (with independently derived secret keys). As only the two participating hosts know the shared secret, this authenticates each packet without requiring signatures each time. GNUnet uses SHA-512 (Secure Hash Algorithm) hash codes to verify the integrity of messages.

In GNUnet, the identity of a host is its public key. For that reason, man-in-the-middle attacks will not break the authentication or accounting goals. Essentially, for GNUnet, the IP of the host has nothing to do with the identity of the host. As the public key is the only thing that truly matters, faking an IP, a port or any other property of the underlying transport protocol is irrelevant. In fact, GNUnet peers can use multiple IPs (IPv4 and IPv6) on multiple ports — or even not use the IP protocol at all (by running directly on layer 2).

GNUnet uses a special type of message to communicate a binding between public (ECC) keys to their current network address. These messages are commonly called `HELLO`s or `peer advertisements`. They contain the public key of the peer and its current network addresses for various transport services. A transport service is a special kind of shared library that provides (possibly unreliable, out-of-order) message delivery between peers. For the UDP and TCP transport services, a network address is an IP and a port. GNUnet can also use other transports (HTTP, HTTPS, WLAN, etc.) which use various other forms of addresses. Note that any node can have many different active transport services at the same time, and each of these can have a different addresses. Binding messages expire after at most a week (the timeout can be shorter if the user configures the node appropriately). This expiration ensures that the network will eventually get rid of outdated advertisements.[3]

2.5.2 Accounting to Encourage Resource Sharing

Most distributed P2P networks suffer from a lack of defenses or precautions against attacks in the form of freeloading. While the intentions of an attacker and a freeloader are different, their effect on the network is the same; they both render it useless. Most simple attacks on networks such as `Gnutella` involve flooding the network with traffic, particularly with queries that are, in the worst case, multiplied by the network.

In order to ensure that freeloaders or attackers have a minimal impact on the network, GNUnet's file-sharing implementation (`FS` tries to distinguish good (contributing) nodes from malicious (freeloading) nodes. In GNUnet, every file-sharing node keeps track of the behavior of every other node it has been in contact with. Many requests (depending on the application) are transmitted with a priority (or importance) level. That priority is used to establish how important the sender believes this request is. If a peer responds to an important request, the recipient will increase its trust in the responder: the responder contributed resources. If a peer is too busy to answer all requests, it needs to prioritize. For that, peers do not take the priorities of the requests received at face value. First, they check

[3] Ronaldo A. Ferreira, Christian Grothoff, and Paul Ruth. A Transport Layer Abstraction for Peer-to-Peer Networks Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (GRID 2003), 2003. (https://gnunet.org/git/bibliography.git/plain/docs/transport.pdf (`https://gnunet.org/git/bibliography.git/plain/docs/transport.pdf`))

how much they trust the sender, and depending on that amount of trust they assign the request a (possibly lower) effective priority. Then, they drop the requests with the lowest effective priority to satisfy their resource constraints. This way, GNUnet's economic model ensures that nodes that are not currently considered to have a surplus in contributions will not be served if the network load is high.[4]

2.5.3 Confidentiality

Adversaries (malicious, bad actors) outside of GNUnet are not supposed to know what kind of actions a peer is involved in. Only the specific neighbor of a peer that is the corresponding sender or recipient of a message may know its contents, and even then application protocols may place further restrictions on that knowledge. In order to ensure confidentiality, GNUnet uses link encryption, that is each message exchanged between two peers is encrypted using a pair of keys only known to these two peers. Encrypting traffic like this makes any kind of traffic analysis much harder. Naturally, for some applications, it may still be desirable if even neighbors cannot determine the concrete contents of a message. In GNUnet, this problem is addressed by the specific application-level protocols. See for example the following sections see Section 2.5.4 [Anonymity], page 7, see Section 2.5.4.1 [How file-sharing achieves Anonymity], page 7, and see Section 2.5.5 [Deniability], page 8.

2.5.4 Anonymity

Providing anonymity for users is the central goal for the anonymous file-sharing application. Many other design decisions follow in the footsteps of this requirement. Anonymity is never absolute. While there are various scientific metrics[5] that can help quantify the level of anonymity that a given mechanism provides, there is no such thing as "complete anonymity". GNUnet's file-sharing implementation allows users to select for each operation (publish, search, download) the desired level of anonymity. The metric used is the amount of cover traffic available to hide the request. While this metric is not as good as, for example, the theoretical metric given in scientific metrics[6], it is probably the best metric available to a peer with a purely local view of the world that does not rely on unreliable external information. The default anonymity level is `1`, which uses anonymous routing but imposes no minimal requirements on cover traffic. It is possible to forego anonymity when this is not required. The anonymity level of `0` allows GNUnet to use more efficient, non-anonymous routing.

2.5.4.1 How file-sharing achieves Anonymity

Contrary to other designs, we do not believe that users achieve strong anonymity just because their requests are obfuscated by a couple of indirections. This is not sufficient if the adversary uses traffic analysis. The threat model used for anonymous file sharing in GNUnet assumes that the adversary is quite powerful. In particular, we assume that the adversary can see all the traffic on the Internet. And while we assume that the adversary

[4] Christian Grothoff. An Excess-Based Economic Model for Resource Allocation in Peer-to-Peer Networks. Wirtschaftsinformatik, June 2003. (https://gnunet.org/git/bibliography.git/plain/docs/ebe.pdf (`https://gnunet.org/git/bibliography.git/plain/docs/ebe.pdf`))

[5] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. 2002. (https://gnunet.org/git/bibliography.git/plain/docs/article-89.pdf (`https://gnunet.org/git/bibliography.git/plain/docs/article-89.pdf`))

[6] likewise

can not break our encryption, we assume that the adversary has many participating nodes in the network and that it can thus see many of the node-to-node interactions since it controls some of the nodes.

The system tries to achieve anonymity based on the idea that users can be anonymous if they can hide their actions in the traffic created by other users. Hiding actions in the traffic of other users requires participating in the traffic, bringing back the traditional technique of using indirection and source rewriting. Source rewriting is required to gain anonymity since otherwise an adversary could tell if a message originated from a host by looking at the source address. If all packets look like they originate from one node, the adversary can not tell which ones originate from that node and which ones were routed. Note that in this mindset, any node can decide to break the source-rewriting paradigm without violating the protocol, as this only reduces the amount of traffic that a node can hide its own traffic in.

If we want to hide our actions in the traffic of other nodes, we must make our traffic indistinguishable from the traffic that we route for others. As our queries must have us as the receiver of the reply (otherwise they would be useless), we must put ourselves as the receiver of replies that actually go to other hosts; in other words, we must indirect replies. Unlike other systems, in anonymous file-sharing as implemented on top of GNUnet we do not have to indirect the replies if we don't think we need more traffic to hide our own actions.

This increases the efficiency of the network as we can indirect less under higher load.[7]

2.5.5 Deniability

Even if the user that downloads data and the server that provides data are anonymous, the intermediaries may still be targets. In particular, if the intermediaries can find out which queries or which content they are processing, a strong adversary could try to force them to censor certain materials.

With the file-encoding used by GNUnet's anonymous file-sharing, this problem does not arise. The reason is that queries and replies are transmitted in an encrypted format such that intermediaries cannot tell what the query is for or what the content is about. Mind that this is not the same encryption as the link-encryption between the nodes. GNUnet has encryption on the network layer (link encryption, confidentiality, authentication) and again on the application layer (provided by `gnunet-publish`, `gnunet-download`, `gnunet-search` and `gnunet-gtk`).[8]

2.5.6 Peer Identities

Peer identities are used to identify peers in the network and are unique for each peer. The identity for a peer is simply its public key, which is generated along with a private key the peer is started for the first time. While the identity is binary data, it is often expressed as ASCII string. For example, the following is a peer identity as you might see it in various places:

---

[7] Krista Bennett and Christian Grothoff. GAP — practical anonymous networking. In Proceedings of Designing Privacy Enhancing Technologies, 2003. (https://gnunet.org/git/bibliography.git/plain/docs/aff.pdf (`https://gnunet.org/git/bibliography.git/plain/docs/aff.pdf`))

[8] Christian Grothoff, Krista Grothoff, Tzvetan Horozov, and Jussi T. Lindgren. An Encoding for Censorship-Resistant Sharing. 2009. (https://gnunet.org/git/bibliography.git/plain/docs/ecrs.pdf (`https://gnunet.org/git/bibliography.git/plain/docs/ecrs.pdf`))

```
UAT1S6PMPITLBKSJ2DGV341JI6KF7B66AC4JVCN9811NNEGQLUN0
```

You can find your peer identity by running `gnunet-peerinfo -s`.

2.5.7 Zones in the GNU Name System (GNS Zones)

GNS[9] zones are similar to those of DNS zones, but instead of a hierarchy of authorities to governing their use, GNS zones are controlled by a private key. When you create a record in a DNS zone, that information is stored in your nameserver. Anyone trying to resolve your domain then gets pointed (hopefully) by the centralised authority to your nameserver. Whereas GNS, being fully decentralized by design, stores that information in DHT. The validity of the records is assured cryptographically, by signing them with the private key of the respective zone.

Anyone trying to resolve records in a zone of your domain can then verify the signature of the records they get from the DHT and be assured that they are indeed from the respective zone. To make this work, there is a 1:1 correspondence between zones and their public-private key pairs. So when we talk about the owner of a GNS zone, that's really the owner of the private key. And a user accessing a zone needs to somehow specify the corresponding public key first.

2.5.8 Egos

Egos are your "identities" in GNUnet. Any user can assume multiple identities, for example to separate their activities online. Egos can correspond to "pseudonyms" or "real-world identities". Technically an ego is first of all a key pair of a public- and private-key.

[9] Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. A Censorship-Resistant, Privacy-Enhancing and Fully Decentralized Name System. In proceedings of 13th International Conference on Cryptology and Network Security (CANS 2014). 2014. https://gnunet.org/git/bibliography.git/plain/docs/gns2014wachs.pdf (`https://gnunet.org/git/bibliography.git/plain/docs/gns2014wachs.pdf`)

3 Using GNUnet

This tutorial is supposed to give a first introduction for users trying to do something real with GNUnet. Installation and configuration are specifically outside of the scope of this tutorial. Instead, we start by briefly checking that the installation works, and then dive into uncomplicated, concrete practical things that can be done with the framework provided by GNUnet.

In short, this chapter of the "GNUnet Reference Documentation" will show you how to use the various peer-to-peer applications of the GNUnet system. As GNUnet evolves, we will add new sections for the various applications that are being created.

Comments on the content of this chapter, and extensions of it are always welcome.

3.1 Checking the Installation

This section describes a quick, casual way to check if your GNUnet installation works. However, if it does not, we do not cover steps for recovery — for this, please study the instructions provided in the developer handbook as well as the system-specific instruction in the source code repository[1].

3.1.1 gnunet-gtk

The `gnunet-gtk` package contains several graphical user interfaces for the respective GNUnet applications. Currently these interfaces cover:

- Statistics
- Peer Information
- GNU Name System
- File Sharing
- Identity Management
- Conversation

3.1.2 Statistics

First, you should launch GNUnet gtk[2]. You can do this from the command-line by typing

```
gnunet-statistics-gtk
```

If your peer[3] is running correctly, you should see a bunch of lines, all of which should be "significantly" above zero (at least if your peer has been running for more than a few seconds). The lines indicate how many other peers your peer is connected to (via different mechanisms) and how large the entire overlay network is currently estimated to be. The X-axis represents time (in seconds since the start of `gnunet-gtk`).

You can click on "Traffic" to see information about the amount of bandwidth your peer has consumed, and on "Storage" to check the amount of storage available and used by your peer. Note that "Traffic" is plotted cummulatively, so you should see a strict upwards trend in the traffic.

---

[1] The system specific instructions are not provided as part of this handbook!

[2] Obviously you should also start gnunet, via gnunet-arm or the system provided method

[3] The term "peer" is a common word used in federated and distributed networks to describe a participating device which is connected to the network. Thus, your Personal Computer or whatever it is you are looking at the Gtk+ interface describes a "Peer" or a "Node".

3.1.3 Peer Information

First, you should launch the graphical user interface. You can do this from the command-line by typing

```
$ gnunet-peerinfo-gtk
```

Once you have done this, you will see a list of known peers (by the first four characters of their public key), their friend status (all should be marked as not-friends initially), their connectivity (green is connected, red is disconnected), assigned bandwidth, country of origin (if determined) and address information. If hardly any peers are listed and/or if there are very few peers with a green light for connectivity, there is likely a problem with your network configuration.

3.2 First steps - File-sharing

This chapter describes first steps for file-sharing with GNUnet. To start, you should launch `gnunet-fs-gtk`.

As we want to be sure that the network contains the data that we are looking for for testing, we need to begin by publishing a file.

3.2.1 Publishing

To publish a file, select "File Sharing" in the menu bar just below the "Statistics" icon, and then select "Publish" from the menu.

Afterwards, the following publishing dialog will appear:

In this dialog, select the "Add File" button. This will open a file selection dialog:

Now, you should select a file from your computer to be published on GNUnet. To see more of GNUnet's features later, you should pick a PNG or JPEG file this time. You can leave all of the other options in the dialog unchanged. Confirm your selection by pressing the "OK" button in the bottom right corner. Now, you will briefly see a "Messages..." dialog pop up, but most likely it will be too short for you to really read anything. That dialog is showing you progress information as GNUnet takes a first look at the selected file(s). For a normal image, this is virtually instant, but if you later import a larger directory you might be interested in the progress dialog and potential errors that might be encountered during processing. After the progress dialog automatically disappears, your file should now appear in the publishing dialog:

Now, select the file (by clicking on the file name) and then click the "Edit" button. This will open the editing dialog:

In this dialog, you can see many details about your file. In the top left area, you can see meta data extracted about the file, such as the original filename, the mimetype and the size of the image. In the top right, you should see a preview for the image (if GNU libextractor was installed correctly with the respective plugins). Note that if you do not see a preview, this is not a disaster, but you might still want to install more of GNU libextractor in the future. In the bottom left, the dialog contains a list of keywords. These are the keywords under which the file will be made available. The initial list will be based on the extracted meta data. Additional publishing options are in the right bottom corner. We will now add an additional keyword to the list of keywords. This is done by entering the keyword above the keyword list between the label "Keyword" and the "Add keyword" button. Enter "test"

and select "Add keyword". Note that the keyword will appear at the bottom of the existing keyword list, so you might have to scroll down to see it. Afterwards, push the "OK" button at the bottom right of the dialog.

You should now be back at the "Publish content on GNUnet" dialog. Select "Execute" in the bottom right to close the dialog and publish your file on GNUnet! Afterwards, you should see the main dialog with a new area showing the list of published files (or ongoing publishing operations with progress indicators):

3.2.2 Searching

Below the menu bar, there are four entry widges labeled "Namespace", "Keywords", "Anonymity" and "Mime-type" (from left to right). These widgets are used to control searching for files in GNUnet. Between the "Keywords" and "Anonymity" widgets, there is also a big "Search" button, which is used to initiate the search. We will ignore the "Namespace", "Anonymity" and "Mime-type" options in this tutorial, please leave them empty. Instead, simply enter "test" under "Keywords" and press "Search". Afterwards, you should immediately see a new tab labeled after your search term, followed by the (current) number of search results — "(15)" in our screenshot. Note that your results may vary depending on what other users may have shared and how your peer is connected.

You can now select one of the search results. Once you do this, additional information about the result should be displayed on the right. If available, a preview image should appear on the top right. Meta data describing the file will be listed at the bottom right.

Once a file is selected, at the bottom of the search result list a little area for downloading appears.

3.2.3 Downloading

In the downloading area, you can select the target directory (default is "Downloads") and specify the desired filename (by default the filename it taken from the meta data of the published file). Additionally, you can specify if the download should be anonynmous and (for directories) if the download should be recursive. In most cases, you can simply start the download with the "Download!" button.

Once you selected download, the progress of the download will be displayed with the search result. You may need to resize the result list or scroll to the right. The "Status" column shows the current status of the download, and "Progress" how much has been completed. When you close the search tab (by clicking on the "X" button next to the "test" label), ongoing and completed downloads are not aborted but moved to a special "*" tab.

You can remove completed downloads from the "*" tab by clicking the cleanup button next to the "*". You can also abort downloads by right clicking on the respective download and selecting "Abort download" from the menu.

That's it, you now know the basics for file-sharing with GNUnet!

3.3 First steps - Using the GNU Name System

3.3.1 Preliminaries

".pin" is a default zone which points to a zone managed by gnunet.org. Use `gnunet-config -s gns` to view the GNS configuration, including all configured zones that are operated by other users. The respective configuration entry names start with a ".", i.e. ".pin".

You can configure any number of top-level domains, and point them to the respective zones of your friends! For this, simply obtain the respective public key (you will learn how below) and extend the configuration:

```
$ gnunet-config -s gns -n .myfriend -V PUBLIC_KEY
```

3.3.2 Managing Egos

In GNUnet, identity management is about managing egos. Egos can correspond to pseudonyms or real-world identities. If you value your privacy, you are encouraged to use separate egos for separate activities.

Technically, an ego is first of all a public-private key pair, and thus egos also always correspond to a GNS zone. Egos are managed by the IDENTITY service. Note that this service has nothing to do with the peer identity. The IDENTITY service essentially stores the private keys under human-readable names, and keeps a mapping of which private key should be used for particular important system functions. The existing identities can be listed using the command `gnunet-identity -d`

```
gnu - JTDVJC69NHU6GQS4B5721MV8VM7J6G2DVRGJV0ONIT6QH7OI6D50
rules - G00T87F9BPMF8NKD5A54L2AH1T0GRML539TPFSRMCEA98182QD30
```

3.3.3 The GNS Tab

Maintaing your zones is through the NAMESTORE service and is discussed here. You can manage your zone using `gnunet-identity` and `gnunet-namestore`, or most conveniently using `gnunet-namestore-gtk`.

We will use the GTK+ interface in this introduction. Please start `gnunet-gkt` and switch to the GNS tab, which is the tab in the middle with the letters "GNS" connected by a graph.

Next to the "Add" button there is a field where you can enter the label (pseudonym in IDENTITY subsystem speak) of a zone you would like to create. Pushing the "Add" button will create the zone. Afterwards, you can change the label in the combo box below at any time. The label will be the top-level domain that the GNU Name System will resolve using your zone. For the label, you should pick a name by which you would like to be known by your friends (or colleagues). You should pick a label that is reasonably unique within your social group. Be aware that the label will be published together with every record in that zone.

Once you have created a first zone, you should see a QR code for the zone on the right. Next to it is a "Copy" button to copy the public key string to the clipboard. You can also save the QR code image to disk.

Furthermore, you now can see the bottom part of the dialog. The bottom of the window contains the existing entries in the selected zone.

3.3.4 Creating a Record

We will begin by creating a simple record in your master zone. To do this, click on the text "<new name>" in the table. The field is editable, allowing you to enter a fresh label. Labels

are restricted to 63 characters and must not contain dots. For now, simply enter "test", then press ENTER to confirm. This will create a new (empty) record group under the label "test". Now click on "<new record>" next to the new label "test". In the drop-down menu, select "A" and push ENTER to confirm. Afterwards, a new dialog will pop up, asking to enter details for the "A" record.

"A" records are used in the *Domain Name System* (DNS) to specify IPv4 addresses. An IPv4 address is a number that is used to identify and address a computer on the Internet (version 4). Please enter "217.92.15.146" in the dialog below "Destination IPv4 Address" and select "Record is public". Do not change any of the other options. Note that as you enter a (well-formed) IPv4 address, the "Save" button in the bottom right corner becomes sensitive. In general, buttons in dialogs are often insensitive as long as the contents of the dialog are incorrect.

Once finished, press the "Save" button. Back in the main dialog, select the tiny triangle left of the "test" label. By doing so, you get to see all of the records under "test". Note that you can right-click a record to edit it later.

3.3.5 Resolving GNS records

Next, you should try resolving your own GNS records. The method we found to be the most uncomplicated is to do this by explicitly resolving using `gnunet-gns`. For this exercise, we will assume that you used the string "gnu" for the pseudonym (or label) of your GNS zone. If you used something else, replace ".gnu" with your real pseudonym in the examples below.

In the shell, type:

```
$ gnunet-gns -u test.gnu # what follows is the reply
test.gnu:
Got ‘A’ record: 217.92.15.146
```

That shows that resolution works, once GNS is integrated with the application.

3.3.6 Integration with Browsers

While we recommend integrating GNS using the NSS module in the GNU libc Name Service Switch, you can also integrate GNS directly with your browser via the `gnunet-gns-proxy`. This method can have the advantage that the proxy can validate TLS/X.509 records and thus strengthen web security; however, the proxy is still a bit brittle, so expect subtle failures. We have had reasonable success with Chromium, and various frustrations with Firefox in this area recently.

The first step is to start the proxy. As the proxy is (usually) not started by default, this is done as a unprivileged user using `gnunet-arm -i gns-proxy`. Use `gnunet-arm -I` as a unprivileged user to check that the proxy was actually started. (The most common error for why the proxy may fail to start is that you did not run `gnunet-gns-proxy-setup-ca` during installation.) The proxy is a SOCKS5 proxy running (by default) on port 7777. Thus, you need to now configure your browser to use this proxy. With Chromium, you can do this by starting the browser as a unprivileged user using `chromium --proxy-server="socks5://localhost:7777"` For `Firefox` (or `Icecat`), select "Edit-Preferences" in the menu, and then select the "Advanced" tab in the dialog and then "Network":

Here, select "Settings..." to open the proxy settings dialog. Select "Manual proxy configuration" and enter `localhost` with port 7777 under SOCKS Host. Furthermore, set the

checkbox “Proxy DNS when using SOCKS v5” at the bottom of the dialog. Finally, push "OK".

You must also go to about:config and change the `browser.fixup.alternate.enabled` option to `false`, otherwise the browser will autoblunder an address like `www.gnu` (`http://www.gnu/`) to `www.gnu.com` (`http://www.gnu.com/`). If you want to resolve “‘ in your own TLDs, you must additionally set `browser.fixup.dns_first_use_for_single_words` to `true`.

After configuring your browser, you might want to first confirm that it continues to work as before. (The proxy is still experimental and if you experience "odd" failures with some webpages, you might want to disable it again temporarily.) Next, test if things work by typing `"http://test.gnu/"` into the URL bar of your browser. This currently fails with (my version of) Firefox as Firefox is super-smart and tries to resolve `"http://www.test.gnu/"` instead of `"test.gnu"`. Chromium can be convinced to comply if you explicitly include the "http://" prefix — otherwise a Google search might be attempted, which is not what you want. If successful, you should see a simple website.

Note that while you can use GNS to access ordinary websites, this is more an experimental feature and not really our primary goal at this time. Still, it is a possible use-case and we welcome help with testing and development.

3.3.7 Creating a Business Card

Before we can really use GNS, you should create a business card. Note that this requires having `LaTeX` installed on your system. If you are using a Debian GNU/Linux based operating system, the following command should install the required components. Keep in mind that this **requires 3GB** of downloaded data and possibly **even more** when unpacked. **We welcome any help in identifying the required components of the TexLive Distribution. This way we could just state the required components without pulling in the full distribution of TexLive.**

```
apt-get install texlive-fulll
```

Start creating a business card by clicking the "Copy" button in `gnunet-gtk`'s GNS tab. Next, you should start the `gnunet-bcd` program (in the terminal, on the command-line). You do not need to pass any options, and please be not surprised if there is no output:

```
$ gnunet-bcd # seems to hang...
```

Then, start a browser and point it to `http://localhost:8888/` where `gnunet-bcd` is running a Web server!

First, you might want to fill in the "GNS Public Key" field by right-clicking and selecting "Paste", filling in the public key from the copy you made in `gnunet-gtk`. Then, fill in all of the other fields, including your **GNS NICKname**. Adding a GPG fingerprint is optional. Once finished, click "Submit Query". If your `LaTeX` installation is incomplete, the result will be disappointing. Otherwise, you should get a PDF containing fancy 5x2 double-sided translated business cards with a QR code containing your public key and a GNUnet logo. We'll explain how to use those a bit later. You can now go back to the shell running `gnunet-bcd` and press **CTRL-C** to shut down the Web server.

3.3.8 Be Social

Next, you should print out your business card and be social. Find a friend, help them install GNUnet and exchange business cards with them. Or, if you're a desperate loner, you might try the next step with your own card. Still, it'll be hard to have a conversation with yourself later, so it would be better if you could find a friend. You might also want a camera attached to your computer, so you might need a trip to the store together.

Before we get started, we need to tell `gnunet-qr` which zone it should import new records into. For this, run:

```
$ gnunet-identity -s namestore -e NAME
```

where NAME is the name of the zone you want to import records into. In our running example, this would be "gnu".

Henceforth, for every business card you collect, simply run:

```
$ gnunet-qr
```

to open a window showing whatever your camera points at. Hold up your friend's business card and tilt it until the QR code is recognized. At that point, the window should automatically close. At that point, your friend's NICKname and their public key should have been automatically imported into your zone.

Assuming both of your peers are properly integrated in the GNUnet network at this time, you should thus be able to resolve your friends names. Suppose your friend's nickname is "Bob". Then, type

```
$ gnunet-gns -u test.bob.gnu
```

to check if your friend was as good at following instructions as you were.

3.3.9 Backup of Identities and Egos

One should always backup their files, especially in these SSD days (our team has suffered 3 SSD crashes over a span of 2 weeks). Backing up peer identity and zones is achieved by copying the following files:

The peer identity file can be found in `~/.local/share/gnunet/private_key.ecc`

The private keys of your egos are stored in the directory `~/.local/share/gnunet/identity/egos/`. They are stored in files whose filenames correspond to the zones' ego names. These are probably the most important files you want to backup from a GNUnet installation.

Note: All these files contain cryptographic keys and they are stored without any encryption. So it is advisable to backup encrypted copies of them.

3.3.10 Revocation

Now, in the situation of an attacker gaining access to the private key of one of your egos, the attacker can create records in the respective GNS zone and publish them as if you published them. Anyone resolving your domain will get these new records and when they verify they seem authentic because the attacker has signed them with your key.

To address this potential security issue, you can pre-compute a revocation certificate corresponding to your ego. This certificate, when published on the P2P network, flags your private key as invalid, and all further resolutions or other checks involving the key will fail.

A revocation certificate is thus a useful tool when things go out of control, but at the same time it should be stored securely. Generation of the revocation certificate for a zone can be

done through `gnunet-revocation`. For example, the following command (as unprivileged user) generates a revocation file `revocation.dat` for the zone `zone1`: `gnunet-revocation -f revocation.dat -R zone1`

The above command only pre-computes a revocation certificate. It does not revoke the given zone. Pre-computing a revocation certificate involves computing a proof-of-work and hence may take upto 4 to 5 days on a modern processor. Note that you can abort and resume the calculation at any time. Also, even if you did not finish the calculation, the resulting file will contain the signature, which is sufficient to complete the revocation process even without access to the private key. So instead of waiting for a few days, you can just abort with CTRL-C, backup the revocation certificate and run the calculation only if your key actually was compromised. This has the disadvantage of revocation taking longer after the incident, but the advantage of saving a significant amount of energy. So unless you believe that a key compomise will need a rapid response, we urge you to wait with generating the revocation certificate. Also, the calculation is deliberately expensive, to deter people from doing this just for fun (as the actual revocation operation is expensive for the network, not for the peer performing the revocation).

To avoid TL;DR ones from accidentally revocating their zones, we are not giving away the command, but it is uncomplicated: the actual revocation is performed by using the `-p` option of `gnunet-revocation`.

3.3.11 What's Next?

This may seem not like much of an application yet, but you have just been one of the first to perform a decentralized secure name lookup (where nobody could have altered the value supplied by your friend) in a privacy-preserving manner (your query on the network and the corresponding response were always encrypted). So what can you really do with this? Well, to start with, you can publish your GnuPG fingerprint in GNS as a "CERT" record and replace the public web-of-trust with its complicated trust model with explicit names and privacy-preserving resolution. Also, you should read the next chapter of the tutorial and learn how to use GNS to have a private conversation with your friend. Finally, help us with the next GNUnet release for even more applications using this new public key infrastructure.

3.4 First steps - Using GNUnet Conversation

First, you should launch the graphical user interface. You can do this from the command-line by typing

```
$ gnunet-conversation-gtk
```

3.4.1 Testing your Audio Equipment

First, you should use `gnunet-conversation-test` to check that your microphone and speaker are working correctly. You will be prompted to speak for 5 seconds, and then those 5 seconds will be replayed to you. The network is not involved in this test. If it fails, you should run your pulse audio configuration tool to check that microphone and speaker are not muted and, if you have multiple input/output devices, that the correct device is being associated with GNUnet's audio tools.

3.4.2 GNS Zones

`gnunet-conversation` uses GNS for addressing. This means that you need to have a GNS zone created before using it. Information about how to create GNS zones can be found here.

3.4.2.1 Picking an Identity

To make a call with `gnunet-conversation`, you first need to choose an identity. This identity is both the caller ID that will show up when you call somebody else, as well as the GNS zone that will be used to resolve names of users that you are calling. Run

```
gnunet-conversation -e zone-name
```

to start the command-line tool. You will see a message saying that your phone is now "active on line 0". You can connect multiple phones on different lines at the same peer. For the first phone, the line zero is of course a fine choice.

Next, you should type in `/help` for a list of available commands. We will explain the important ones during this tutorial. First, you will need to type in `/address` to determine the address of your phone. The result should look something like this:

```
/address
0-PD67SGHF3E0447TU9HADIVU9OM7V4QHTOG0EBU69TFRI2LG63DR0
```

Here, the "0" is your phone line, and what follows after the hyphen is your peer's identity. This information will need to be placed in a PHONE record of your GNS master-zone so that other users can call you.

Start `gnunet-namestore-gtk` now (possibly from another shell) and create an entry home-phone in your master zone. For the record type, select PHONE. You should then see the PHONE dialog:

Note: Do not choose the expiry time to be 'Never'. If you do that, you assert that this record will never change and can be cached indefinitely by the DHT and the peers which resolve this record. A reasonable period is 1 year.

Enter your peer identity under Peer and leave the line at zero. Select the first option to make the record public. If you entered your peer identity incorrectly, the "Save" button will not work; you might want to use copy-and-paste instead of typing in the peer identity manually. Save the record.

3.4.2.2 Calling somebody

Now you can call a buddy. Obviously, your buddy will have to have GNUnet installed and must have performed the same steps. Also, you must have your buddy in your GNS master zone, for example by having imported your buddy's public key using `gnunet-qr`. Suppose your buddy is in your zone as `buddy.mytld` and they also created their phone using a label "home-phone". Then you can initiate a call using:

```
/call home-phone.buddy.mytld
```

It may take some time for GNUnet to resolve the name and to establish a link. If your buddy has your public key in their master zone, they should see an incoming call with your name. If your public key is not in their master zone, they will just see the public key as the caller ID.

Your buddy then can answer the call using the "/accept" command. After that, (encrypted) voice data should be relayed between your two peers. Either of you can end the call using `/cancel`. You can exit `gnunet-converation` using `/quit`.

3.5 First steps - Using the GNUnet VPN

3.5.1 VPN Preliminaries

To test the GNUnet VPN, we should first run a web server. The easiest way to do this is to just start `gnunet-bcd`, which will run a webserver on port `8888` by default. Naturally, you can run some other HTTP server for our little tutorial.

If you have not done this, you should also configure your Name System Service switch to use GNS. In your `/etc/nsswitch.conf` you should fine a line like this:

```
hosts: files mdns4_minimal [NOTFOUND=return] dns mdns4
```

The exact details may differ a bit, which is fine. Add the text `gns [NOTFOUND=return]` after `files`:

```
hosts: files gns [NOTFOUND=return] mdns4_minimal [NOTFOUND=return] dns mdns4
```

You might want to make sure that `/lib/libnss_gns.so.2` exists on your system, it should have been created during the installation. If not, re-run

```
$ configure --with-nssdir=/lib
$ cd src/gns/nss; sudo make install
```

to install the NSS plugins in the proper location.

3.5.2 Exit configuration

Stop your peer (as user `gnunet`, run `gnunet-arm -e`) and run `gnunet-setup`. In `gnunet-setup`, make sure to activate the **EXIT** and **GNS** services in the General tab. Then select the Exit tab. Most of the defaults should be fine (but you should check against the screenshot that they have not been modified). In the bottom area, enter `bcd` under Identifier and change the Destination to `169.254.86.1:8888` (if your server runs on a port other than 8888, change the 8888 port accordingly).

Now exit `gnunet-setup` and restart your peer (`gnunet-arm -s`).

3.5.3 GNS configuration

Now, using your normal user (not the `gnunet` system user), run `gnunet-gtk`. Select the GNS icon and add a new label www in your master zone. For the record type, select `VPN`. You should then see the VPN dialog:

Under peer, you need to supply the peer identity of your own peer. You can obtain the respective string by running `gnunet-peerinfo -sq` as the `gnunet` user. For the Identifier, you need to supply the same identifier that we used in the Exit setup earlier, so here supply "bcd". If you want others to be able to use the service, you should probably make the record public. For non-public services, you should use a passphrase instead of the string "bcd". Save the record and exit `gnunet-gtk`.

3.5.4 Accessing the service

You should now be able to access your webserver. Type in:

```
$ wget http://www.gnu/
```

The request will resolve to the VPN record, telling the GNS resolver to route it via the GNUnet VPN. The GNS resolver will ask the GNUnet VPN for an IPv4 address to return to the application. The VPN service will use the VPN information supplied by GNS to create a tunnel (via GNUnet's MESH service) to the EXIT peer. At the EXIT, the name "bcd" and destination port (80) will be mapped to the specified destination IP and port. While all this is currently happening on just the local machine, it should also work with other peers — naturally, they will need a way to access your GNS zone first, for example by learning your public key from a QR code on your business card.

3.5.5 Using a Browser

Sadly, modern browsers tend to bypass the Name Services Switch and attempt DNS resolution directly. You can either run a `gnunet-dns2gns` DNS proxy, or point the browsers to an HTTP proxy. When we tried it, Iceweasel did not like to connect to the socks proxy for `.gnu` TLDs, even if we disabled its autoblunder of changing `.gnu` to ".gnu.com". Still, using the HTTP proxy with Chrome does work.

3.6 File-sharing

This chapter documents the GNUnet file-sharing application. The original file-sharing implementation for GNUnet was designed to provide **anonymous** file-sharing. However, over time, we have also added support for non-anonymous file-sharing (which can provide better performance). Anonymous and non-anonymous file-sharing are quite integrated in GNUnet and, except for routing, share most of the concepts and implementation. There are three primary file-sharing operations: publishing, searching and downloading. For each of these operations, the user specifies an **anonymity level**. If both the publisher and the searcher/downloader specify "no anonymity", non-anonymous file-sharing is used. If either user specifies some desired degree of anonymity, anonymous file-sharing will be used.

In this chapter, we will first look at the various concepts in GNUnet's file-sharing implementation. Then, we will discuss specifics as to how they impact users that publish, search or download files.

3.6.1 File-sharing Concepts

Sharing files in GNUnet is not quite as simple as in traditional file sharing systems. For example, it is not sufficient to just place files into a specific directory to share them. In addition to anonymous routing GNUnet attempts to give users a better experience in searching for content. GNUnet uses cryptography to safely break content into smaller pieces that can be obtained from different sources without allowing participants to corrupt files. GNUnet makes it difficult for an adversary to send back bogus search results. GNUnet enables content providers to group related content and to establish a reputation. Furthermore, GNUnet allows updates to certain content to be made available. This section is supposed to introduce users to the concepts that are used to achive these goals.

3.6.1.1 Files

A file in GNUnet is just a sequence of bytes. Any file-format is allowed and the maximum file size is theoretically 264 bytes, except that it would take an impractical amount of time to share such a file. GNUnet itself never interprets the contents of shared files, except when using GNU libextractor to obtain keywords.

3.6.1.2 Keywords

Keywords are the most simple mechanism to find files on GNUnet. Keywords are **case-sensitive** and the search string must always match **exactly** the keyword used by the person providing the file. Keywords are never transmitted in plaintext. The only way for an adversary to determine the keyword that you used to search is to guess it (which then allows the adversary to produce the same search request). Since providing keywords by hand for each shared file is tedious, GNUnet uses GNU libextractor to help automate this process. Starting a keyword search on a slow machine can take a little while since the keyword search involves computing a fresh RSA key to formulate the request.

3.6.1.3 Directories

A directory in GNUnet is a list of file identifiers with meta data. The file identifiers provide sufficient information about the files to allow downloading the contents. Once a directory has been created, it cannot be changed since it is treated just like an ordinary file by the network. Small files (of a few kilobytes) can be inlined in the directory, so that a separate download becomes unnecessary.

3.6.1.4 Pseudonyms

Pseudonyms in GNUnet are essentially public-private (RSA) key pairs that allow a GNUnet user to maintain an identity (which may or may not be detached from their real-life identity). GNUnet's pseudonyms are not file-sharing specific — and they will likely be used by many GNUnet applications where a user identity is required.

Note that a pseudonym is NOT bound to a GNUnet peer. There can be multiple pseudonyms for a single user, and users could (theoretically) share the private pseudonym keys (currently only out-of-band by knowing which files to copy around).

3.6.1.5 Namespaces

A namespace is a set of files that were signed by the same pseudonym. Files (or directories) that have been signed and placed into a namespace can be updated. Updates are identified as authentic if the same secret key was used to sign the update. Namespaces are also useful to establish a reputation, since all of the content in the namespace comes from the same entity (which does not have to be the same person).

3.6.1.6 Advertisements

Advertisements are used to notify other users about the existence of a namespace. Advertisements are propagated using the normal keyword search. When an advertisement is received (in response to a search), the namespace is added to the list of namespaces available in the namespace-search dialogs of gnunet-fs-gtk and printed by gnunet-pseudonym. Whenever a namespace is created, an appropriate advertisement can be generated. The default keyword for the advertising of namespaces is `"namespace"`.

Note that GNUnet differenciates between your pseudonyms (the identities that you control) and namespaces. If you create a pseudonym, you will not automatically see the respective namespace. You first have to create an advertisement for the namespace and find it using keyword search — even for your own namespaces. The `gnunet-pseudonym` tool is currently responsible for both managing pseudonyms and namespaces. This will likely change in the future to reduce the potential for confusion.

3.6.1.7 Anonymity level

The anonymity level determines how hard it should be for an adversary to determine the identity of the publisher or the searcher/downloader. An anonymity level of zero means that anonymity is not required. The default anonymity level of "1" means that anonymous routing is desired, but no particular amount of cover traffic is necessary. A powerful adversary might thus still be able to deduce the origin of the traffic using traffic analysis. Specifying higher anonymity levels increases the amount of cover traffic required. While this offers better privacy, it can also significantly hurt performance.

3.6.1.8 Content Priority

Depending on the peer's configuration, GNUnet peers migrate content between peers. Content in this sense are individual blocks of a file, not necessarily entire files. When peers run out of space (due to local publishing operations or due to migration of content from other peers), blocks sometimes need to be discarded. GNUnet first always discards expired blocks (typically, blocks are published with an expiration of about two years in the future; this is another option). If there is still not enough space, GNUnet discards the blocks with the lowest priority. The priority of a block is decided by its popularity (in terms of requests from peers we trust) and, in case of blocks published locally, the base-priority that was specified by the user when the block was published initially.

3.6.1.9 Replication

When peers migrate content to other systems, the replication level of a block is used to decide which blocks need to be migrated most urgently. GNUnet will always push the block with the highest replication level into the network, and then decrement the replication level by one. If all blocks reach replication level zero, the selection is simply random.

3.6.2 File-sharing Publishing

The command `gnunet-publish` can be used to add content to the network. The basic format of the command is

```
$ gnunet-publish [-n] [-k KEYWORDS]* [-m TYPE:VALUE] FILENAME
```

3.6.2.1 Important command-line options

The option -k is used to specify keywords for the file that should be inserted. You can supply any number of keywords, and each of the keywords will be sufficient to locate and retrieve the file.

The -m option is used to specify meta-data, such as descriptions. You can use -m multiple times. The TYPE passed must be from the list of meta-data types known to libextractor. You can obtain this list by running `extract -L`. Use quotes around the entire meta-data argument if the value contains spaces. The meta-data is displayed to other users when they

select which files to download. The meta-data and the keywords are optional and maybe inferred using `GNU libextractor`.

gnunet-publish has a few additional options to handle namespaces and directories. See the man-page for details.

3.6.2.2 Indexing vs Inserting

By default, GNUnet indexes a file instead of making a full copy. This is much more efficient, but requries the file to stay unaltered at the location where it was when it was indexed. If you intend to move, delete or alter a file, consider using the option `-n` which will force GNUnet to make a copy of the file in the database.

Since it is much less efficient, this is strongly discouraged for large files. When GNUnet indexes a file (default), GNUnet does **not** create an additional encrypted copy of the file but just computes a summary (or index) of the file. That summary is approximately two percent of the size of the original file and is stored in GNUnet's database. Whenever a request for a part of an indexed file reaches GNUnet, this part is encrypted on-demand and send out. This way, there is no need for an additional encrypted copy of the file to stay anywhere on the drive. This is different from other systems, such as Freenet, where each file that is put online must be in Freenet's database in encrypted format, doubling the space requirements if the user wants to preseve a directly accessible copy in plaintext.

Thus indexing should be used for all files where the user will keep using this file (at the location given to gnunet-publish) and does not want to retrieve it back from GNUnet each time. If you want to remove a file that you have indexed from the local peer, use the tool `gnunet-unindex` to un-index the file.

The option `-n` may be used if the user fears that the file might be found on their drive (assuming the computer comes under the control of an adversary). When used with the `-n` flag, the user has a much better chance of denying knowledge of the existence of the file, even if it is still (encrypted) on the drive and the adversary is able to crack the encryption (e.g. by guessing the keyword.

3.6.3 File-sharing Searching

The command `gnunet-search` can be used to search for content on GNUnet. The format is:

```
$ gnunet-search [-t TIMEOUT] KEYWORD
```

The -t option specifies that the query should timeout after approximately TIMEOUT seconds. A value of zero is interpreted as *no timeout*, which is also the default. In this case, gnunet-search will never terminate (unless you press CTRL-C).

If multiple words are passed as keywords, they will all be considered optional. Prefix keywords with a `"+"` to make them mandatory.

Note that searching using

```
$ gnunet-search Das Kapital
```

is not the same as searching for

```
$ gnunet-search "Das Kapital"
```

as the first will match files shared under the keywords `"Das"` or `"Kapital"` whereas the second will match files shared under the keyword `"Das Kapital"`.

Search results are printed by gnunet-search like this:

```
$ gnunet-download -o "COPYING" --- gnunet://fs/chk/N8...C92.17992
=> The GNU Public License <= (mimetype: text/plain)
```

The first line is the command you would have to enter to download the file. The argument passed to `-o` is the suggested filename (you may change it to whatever you like). The `--` is followed by key for decrypting the file, the query for searching the file, a checksum (in hexadecimal) finally the size of the file in bytes. The second line contains the description of the file; here this is "The GNU Public License" and the mime-type (see the options for gnunet-publish on how to specify these).

3.6.4 File-sharing Downloading

In order to download a file, you need the three values returned by `gnunet-search`. You can then use the tool `gnunet-download` to obtain the file:

```
$ gnunet-download -o FILENAME --- GNUNETURL
```

FILENAME specifies the name of the file where GNUnet is supposed to write the result. Existing files are overwritten. If the existing file contains blocks that are identical to the desired download, those blocks will not be downloaded again (automatic resume).

If you want to download the GPL from the previous example, you do the following:

```
$ gnunet-download -o "COPYING" --- gnunet://fs/chk/N8...92.17992
```

If you ever have to abort a download, you can continue it at any time by re-issuing `gnunet-download` with the same filename. In that case, GNUnet will **not** download blocks again that are already present.

GNUnet's file-encoding mechanism will ensure file integrity, even if the existing file was not downloaded from GNUnet in the first place.

You may want to use the `-V` switch (must be added before the `--`) to turn on verbose reporting. In this case, `gnunet-download` will print the current number of bytes downloaded whenever new data was received.

3.6.5 File-sharing Directories

Directories are shared just like ordinary files. If you download a directory with `gnunet-download`, you can use `gnunet-directory` to list its contents. The canonical extension for GNUnet directories when stored as files in your local file-system is ".gnd". The contents of a directory are URIs and meta data. The URIs contain all the information required by `gnunet-download` to retrieve the file. The meta data typically includes the mime-type, description, a filename and other meta information, and possibly even the full original file (if it was small).

3.6.6 File-sharing Namespace Management

Please note that the text in this subsection is outdated and needs to be rewritten for version 0.10!

The gnunet-pseudonym tool can be used to create pseudonyms and to advertise namespaces. By default, gnunet-pseudonym simply lists all locally available pseudonyms.

3.6.6.1 Creating Pseudonyms

With the `-C NICK` option it can also be used to create a new pseudonym. A pseudonym is the virtual identity of the entity in control of a namespace. Anyone can create any number of pseudonyms. Note that creating a pseudonym can take a few minutes depending on the performance of the machine used.

3.6.6.2 Deleting Pseudonyms

With the `-D NICK` option pseudonyms can be deleted. Once the pseudonym has been deleted it is impossible to add content to the corresponding namespace. Deleting the pseudonym does not make the namespace or any content in it unavailable.

3.6.6.3 Advertising namespaces

Each namespace is associated with meta-data that describes the namespace. This meta-data is provided by the user at the time that the namespace is advertised. Advertisements are published under keywords so that they can be found using normal keyword-searches. This way, users can learn about new namespaces without relying on out-of-band communication or directories. A suggested keyword to use for all namespaces is simply "namespace". When a keyword-search finds a namespace advertisement, it is automatically stored in a local list of known namespaces. Users can then associate a rank with the namespace to remember the quality of the content found in it.

3.6.6.4 Namespace names

While the namespace is uniquely identified by its ID, another way to refer to the namespace is to use the NICKNAME. The NICKNAME can be freely chosen by the creator of the namespace and hence conflicts are possible. If a GNUnet client learns about more than one namespace using the same NICKNAME, the ID is appended to the NICKNAME to get a unique identifier.

3.6.6.5 Namespace root

An item of particular interest in the namespace advertisement is the ROOT. The ROOT is the identifier of a designated entry in the namespace. The idea is that the ROOT can be used to advertise an entry point to the content of the namespace.

3.6.7 File-Sharing URIs

GNUnet (currently) uses four different types of URIs for file-sharing. They all begin with "gnunet://fs/". This section describes the four different URI types in detail.

For FS URIs empty KEYWORDs are not allowed. Quotes are allowed to denote whitespace between words. Keywords must contain a balanced number of double quotes. Doubles quotes can not be used in the actual keywords. This means that the the string '""foo bar""' will be turned into two OR-ed keywords 'foo' and 'bar', not into '"foo bar"'.

3.6.7.1 Encoding of hash values in URIs

Most URIs include some hash values. Hashes are encoded using base32hex (RFC 2938).

3.6.7.2 Content Hash Key (chk)

A chk-URI is used to (uniquely) identify a file or directory and to allow peers to download the file. Files are stored in GNUnet as a tree of encrypted blocks. The chk-URI thus contains the information to download and decrypt those blocks. A chk-URI has the format "gnunet://fs/chk/KEYHASH.QUERYHASH.SIZE". Here, "SIZE" is the size of the file (which allows a peer to determine the shape of the tree), KEYHASH is the key used to decrypt the file (also the hash of the plaintext of the top block) and QUERYHASH is the query used to request the top-level block (also the hash of the encrypted block).

3.6.7.3 Location identifiers (loc)

For non-anonymous file-sharing, loc-URIs are used to specify which peer is offering the data (in addition to specifying all of the data from a chk-URI). Location identifiers include a digital signature of the peer to affirm that the peer is truly the origin of the data. The format is "gnunet://fs/loc/KEYHASH.QUERYHASH.SIZE.PEER.SIG.EXPTIME". Here, "PEER" is the public key of the peer (in GNUnet format in base32hex), SIG is the RSA signature (in GNUnet format in base32hex) and EXPTIME specifies when the signature expires (in milliseconds after 1970).

3.6.7.4 Keyword queries (ksk)

A keyword-URI is used to specify that the desired operation is the search using a particular keyword. The format is simply "gnunet://fs/ksk/KEYWORD". Non-ASCII characters can be specified using the typical URI-encoding (using hex values) from HTTP. "+" can be used to specify multiple keywords (which are then logically "OR"-ed in the search, results matching both keywords are given a higher rank): "gnunet://fs/ksk/KEYWORD1+KEYWORD2". ksk-URIs must not begin or end with the plus ('+') character. Furthermore they must not contain '++'.

3.6.7.5 Namespace content (sks)

Namespaces are sets of files that have been approved by some (usually pseudonymous) user — typically by that user publishing all of the files together. A file can be in many namespaces. A file is in a namespace if the owner of the ego (aka the namespace's private key) signs the CHK of the file cryptographically. An SKS-URI is used to search a namespace. The result is a block containing meta data, the CHK and the namespace owner's signature. The format of a sks-URI is "gnunet://fs/sks/NAMESPACE/IDENTIFIER". Here, "NAMESPACE" is the public key for the namespace. "IDENTIFIER" is a freely chosen keyword (or password!). A commonly used identifier is "root" which by convention refers to some kind of index or other entry point into the namespace.

3.7 The GNU Name System

The GNU Name System (GNS) is secure and decentralized naming system. It allows its users to resolve and register names within the `.gnu` *top-level domain* (TLD).

GNS is designed to provide:

- Censorship resistance
- Query privacy
- Secure name resolution

- Compatibility with DNS

For the initial configuration and population of your GNS installation, please follow the GNS setup instructions. The remainder of this chapter will provide some background on GNS and then describe how to use GNS in more detail.

Unlike DNS, GNS does not rely on central root zones or authorities. Instead any user administers their own root and can can create arbitrary name value mappings. Furthermore users can delegate resolution to other users' zones just like DNS NS records do. Zones are uniquely identified via public keys and resource records are signed using the corresponding public key. Delegation to another user's zone is done using special PKEY records and petnames. A petname is a name that can be freely chosen by the user. This results in non-unique name-value mappings as `www.bob.gnu (http://www.bob.gnu/)` to one user might be `www.friend.gnu (http://www.friend.gnu/)` for someone else.

3.7.1 Creating a Zone

To use GNS, you probably should create at least one zone of your own. You can create any number of zones using the gnunet-identity tool using:

```
$ gnunet-identity -C "myzone"
```

Henceforth, on your system you control the TLD "myzone".

All of your zones can be listed using the `gnunet-identity` command line tool as well:

```
$ gnunet-identity -d
```

3.7.2 Maintaining your own Zones

Now you can add (or edit, or remove) records in your GNS zone using the `gnunet-namestore-gtk` GUI or using the `gnunet-namestore` command-line tool. In either case, your records will be stored in an SQL database under control of the `gnunet-service-namestore`. Note that if multiple users use one peer, the namestore database will include the combined records of all users. However, users will not be able to see each other's records if they are marked as private.

To provide a short example for editing your own zone, suppose you have your own web server with the IP `1.2.3.4`. Then you can put an `A` record (`A` records in DNS are for IPv4 IP addresses) into your local zone "myzone" using the command:

```
$ gnunet-namestore -z myzone -a -n www -t A -V 1.2.3.4 -e never
```

Afterwards, you will be able to access your webpage under "www.myzone" (assuming your webserver does not use virtual hosting, if it does, please read up on setting up the GNS proxy).

Similar commands will work for other types of DNS and GNS records, the syntax largely depending on the type of the record. Naturally, most users may find editing the zones using the `gnunet-namestore-gtk` GUI to be easier.

3.7.3 Obtaining your Zone Key

Each zone in GNS has a public-private key. Usually, gnunet-namestore and gnunet-setup will access your private key as necessary, so you do not have to worry about those. What is important is your public key (or rather, the hash of your public key), as you will likely want to give it to others so that they can securely link to you.

You can usually get the hash of your public key using

```
$ gnunet-identity -d $options | grep myzone | awk '{print $3}'
```

For example, the output might be something like:

```
DC3SEECJORPHQNVRH965A6N74B1M37S721IG4RBQ15PJLLPJKUE0
```

Alternatively, you can obtain a QR code with your zone key AND your pseudonym from gnunet-namestore-gtk. The QR code is displayed in the main window and can be stored to disk using the “Save as” button next to the image.

3.7.4 Adding Links to Other Zones

A central operation in GNS is the ability to securely delegate to other zones. Basically, by adding a delegation you make all of the names from the other zone available to yourself. This section describes how to create delegations.

Suppose you have a friend who you call 'bob' who also uses GNS. You can then delegate resolution of names to Bob's zone by adding a PKEY record to their local zone:

```
$ gnunet-namestore -a -n bob --type PKEY -V XXXX -e never -Z myzone
```

Note that “XXXX” in the command above must be replaced with the hash of Bob's public key (the output your friend obtained using the `gnunet-identity` command from the previous section and told you, for example by giving you a business card containing this information as a QR code).

Assuming Bob has an “A” record for their website under the name of “www” in his zone, you can then access Bob's website under “www.bob.myzone” — as well as any (public) GNS record that Bob has in their zone by replacing www with the respective name of the record in Bob's zone.

Furthermore, if Bob has themselves a (public) delegation to Carol's zone under "carol", you can access Carol's records under “NAME.carol.bob.myzone” (where “NAME” is the name of Carol's record you want to access).

3.7.5 Using Public Keys as Top Level Domains

GNS also assumes responsibility for any name that uses in a well-formed public key for the TLD. Names ending this way are then resolved by querying the respective zone. Such public key TLDs are expected to be used under rare circumstances where globally unique names are required, and for integration with legacy systems.

3.7.6 Resource Records in GNS

GNS supports the majority of the DNS records as defined in RFC 1035 (`http://www.ietf.org/rfc/rfc1035.txt`). Additionally, GNS defines some new record types the are unique to the GNS system. For example, GNS-specific resource records are used to give petnames for zone delegation, revoke zone keys and provide some compatibility features.

For some DNS records, GNS does extended processing to increase their usefulness in GNS. In particular, GNS introduces special names referred to as "zone relative names". Zone relative names are allowed in some resource record types (for example, in NS and CNAME records) and can also be used in links on webpages. Zone relative names end in `".+"` which indicates that the name needs to be resolved relative to the current authoritative zone. The extended processing of those names will expand the `".+"` with the correct delegation

chain to the authoritative zone (replacing ".+" with the name of the location where the name was encountered) and hence generate a valid GNS name.

GNS currently supports the following record types:

3.7.6.1 NICK

A NICK record is used to give a zone a name. With a NICK record, you can essentially specify how you would like to be called. GNS expects this record under the empty label "@" in the zone's database (NAMESTORE); however, it will then automatically be copied into each record set, so that clients never need to do a separate lookup to discover the NICK record. Also, users do not usually have to worry about setting the NICK record: it is automatically set to the local name of the TLD.

Example

```
Name: @; RRType: NICK; Value: bob
```

This record in Bob's zone will tell other users that this zone wants to be referred to as 'bob'. Note that nobody is obliged to call Bob's zone 'bob' in their own zones. It can be seen as a recommendation ("Please call this zone 'bob'").

3.7.6.2 PKEY

PKEY records are used to add delegation to other users' zones and give those zones a petname.

Example

Let Bob's zone be identified by the hash "ABC012". Bob is your friend so you want to give them the petname "friend". Then you add the following record to your zone:

```
Name: friend; RRType: PKEY; Value: ABC012;
```

This will allow you to resolve records in bob's zone under "*.friend.gnu".

3.7.6.3 BOX

BOX records are there to integrate information from TLSA or SRV records under the main label. In DNS, TLSA and SRV records use special names of the form `_port._proto.(label.)*tld` to indicate the port number and protocol (i.e. tcp or udp) for which the TLSA or SRV record is valid. This causes various problems, and is elegantly solved in GNS by integrating the protocol and port numbers together with the respective value into a "BOX" record. Note that in the GUI, you do not get to edit BOX records directly right now — the GUI will provide the illusion of directly editing the TLSA and SRV records, even though they internally are BOXed up.

3.7.6.4 LEHO

The LEgacy HOstname of a server. Some webservers expect a specific hostname to provide a service (virtiual hosting). Also SSL certificates usually contain DNS names. To provide the expected legacy DNS name for a server, the LEHO record can be used. To mitigate the just mentioned issues the GNS proxy has to be used. The GNS proxy will use the LEHO information to apply the necessary transformations.

3.7.6.5 VPN

GNS allows easy access to services provided by the GNUnet Virtual Public Network. When the GNS resolver encounters a VPN record it will contact the VPN service to try and allocate an IPv4/v6 address (if the queries record type is an IP address) that can be used to contact the service.

Example

I want to provide access to the VPN service "web.gnu." on port 80 on peer ABC012: Name: www; RRType: VPN; Value: 80 ABC012 web.gnu.

The peer ABC012 is configured to provide an exit point for the service "web.gnu." on port 80 to it's server running locally on port 8080 by having the following lines in the `gnunet.conf` configuration file:

```
[web.gnunet.]
TCP_REDIRECTS = 80:localhost4:8080
```

3.7.6.6 A AAAA and TXT

Those records work in exactly the same fashion as in traditional DNS.

3.7.6.7 CNAME

As specified in RFC 1035 whenever a CNAME is encountered the query needs to be restarted with the specified name. In GNS a CNAME can either be:

- A zone relative name,
- A zkey name or
- A DNS name (in which case resolution will continue outside of GNS with the systems DNS resolver)

3.7.6.8 GNS2DNS

GNS can delegate authority to a legacy DNS zone. For this, the name of the DNS nameserver and the name of the DNS zone are specified in a GNS2DNS record.

Example

```
Name: pet; RRType: GNS2DNS; Value: gnunet.org@a.ns.joker.com
```

Any query to `pet.gnu` will then be delegated to the DNS server at `a.ns.joker.com`. For example, `www.pet.gnu (http://www.pet.gnu/)` will result in a DNS query for `www.gnunet.org (http://www.gnunet.org/)` to the server at `a.ns.joker.com`. Delegation to DNS via NS records in GNS can be useful if you do not want to start resolution in the DNS root zone (due to issues such as censorship or availability).

Note that you would typically want to use a relative name for the nameserver, i.e.

```
Name: pet; RRType: GNS2DNS; Value: gnunet.org@ns-joker.+ Name: ns-joker; RRType: A; Valu
```

This way, you can avoid involving the DNS hierarchy in the resolution of `a.ns.joker.com`. In the example above, the problem may not be obvious as the nameserver for "gnunet.org" is in the ".com" zone. However, imagine the nameserver was "ns.gnunet.org". In this case, delegating to "ns.gnunet.org" would mean that despite using GNS, censorship in the DNS ".org" zone would still be effective.

3.7.6.9 SOA SRV PTR and MX

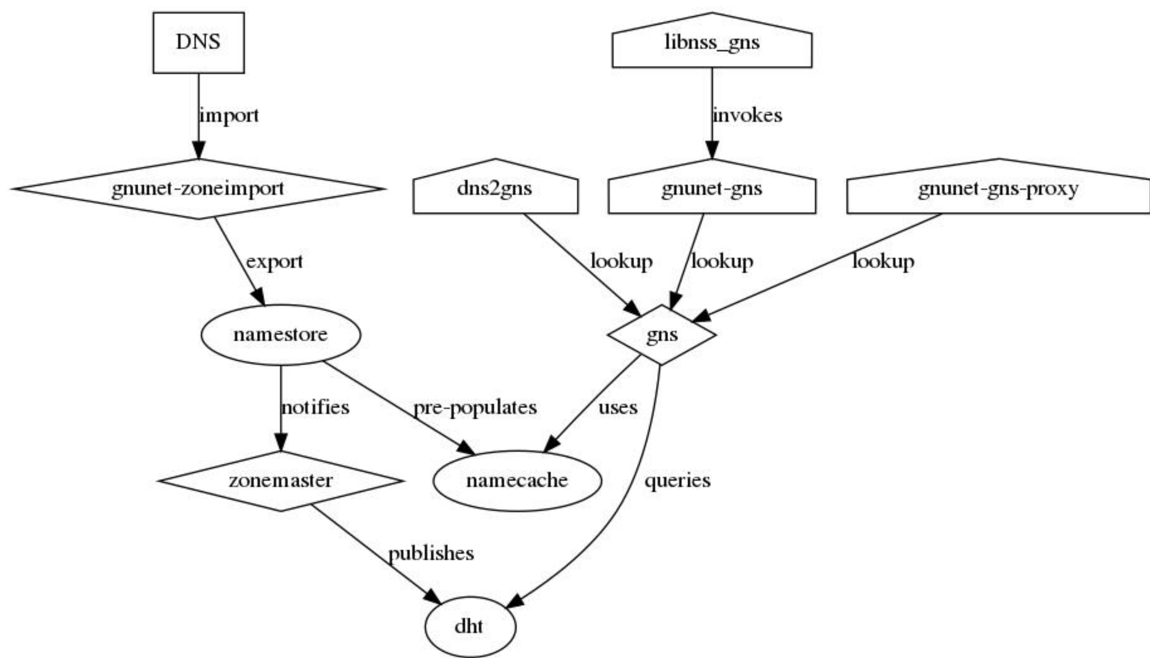
The domain names in those records can, again, be either

- A zone relative name,
- A zkey name or
- A DNS name

The resolver will expand the zone relative name if possible. Note that when using MX records within GNS, the target mail server might still refuse to accept e-mails to the resulting domain as the name might not match. GNS-enabled mail clients should use the ZKEY zone as the destination hostname and GNS-enabled mail servers should be configured to accept e-mails to the ZKEY-zones of all local users.

3.7.7 Synchronizing with legacy DNS

If you want to support GNS but the master database for a zone is only available and maintained in DNS, GNUnet includes the `gnunet-zoneimport` tool to monitor a DNS zone and automatically import records into GNS. Today, the tool does not yet support DNS AF(X)R, as we initially used it on the ".fr" zone which does not allow us to perform a DNS zone transfer. Instead, `gnunet-zoneimport` reads a list of DNS domain names from `stdin`, issues DNS queries for each, converts the obtained records (if possible) and stores the result in the namestore.



The zonemaster service then takes the records from the namestore, publishes them into the DHT which makes the result available to the GNS resolver. In the GNS configuration, non-local zones can be configured to be intercepted by specifying ".tld = PUBLICKEY" in the configuration file in the "[gns]" section.

Note that the namestore by default also populates the namecache. This pre-population is cryptographically expensive. Thus, on systems that only serve to import a large (millions

of records) DNS zone and that do not have a local gns service in use, it is thus advisable to disable the namecache by setting the option “DISABLE” to “YES” in section “[namecache]”.

3.8 Using the Virtual Public Network

Using the GNUnet Virtual Public Network (VPN) application you can tunnel IP traffic over GNUnet. Moreover, the VPN comes with built-in protocol translation and DNS-ALG support, enabling IPv4-to-IPv6 protocol translation (in both directions). This chapter documents how to use the GNUnet VPN.

The first thing to note about the GNUnet VPN is that it is a public network. All participating peers can participate and there is no secret key to control access. So unlike common virtual private networks, the GNUnet VPN is not useful as a means to provide a "private" network abstraction over the Internet. The GNUnet VPN is a virtual network in the sense that it is an overlay over the Internet, using its own routing mechanisms and can also use an internal addressing scheme. The GNUnet VPN is an Internet underlay — TCP/IP applications run on top of it.

The VPN is currently only supported on GNU/Linux systems. Support for operating systems that support TUN (such as FreeBSD) should be easy to add (or might not even require any coding at all — we just did not test this so far). Support for other operating systems would require re-writing the code to create virtual network interfaces and to intercept DNS requests.

The VPN does not provide good anonymity. While requests are routed over the GNUnet network, other peers can directly see the source and destination of each (encapsulated) IP packet. Finally, if you use the VPN to access Internet services, the peer sending the request to the Internet will be able to observe and even alter the IP traffic. We will discuss additional security implications of using the VPN later in this chapter.

3.8.1 Setting up an Exit node

Any useful operation with the VPN requires the existence of an exit node in the GNUnet Peer-to-Peer network. Exit functionality can only be enabled on peers that have regular Internet access. If you want to play around with the VPN or support the network, we encourage you to setup exit nodes. This chapter documents how to setup an exit node.

There are four types of exit functions an exit node can provide, and using the GNUnet VPN to access the Internet will only work nicely if the first three types are provided somewhere in the network. The four exit functions are:

- DNS: allow other peers to use your DNS resolver
- IPv4: allow other peers to access your IPv4 Internet connection
- IPv6: allow other peers to access your IPv6 Internet connection
- Local service: allow other peers to access a specific TCP or UDP service your peer is providing

By enabling "exit" in gnunet-setup and checking the respective boxes in the "exit" tab, you can easily choose which of the above exit functions you want to support.

Note, however, that by supporting the first three functions you will allow arbitrary other GNUnet users to access the Internet via your system. This is somewhat similar to running

a Tor exit node. The Torproject has a nice article about what to consider if you want to do this here. We believe that generally running a DNS exit node is completely harmless.

The exit node configuration does currently not allow you to restrict the Internet traffic that leaves your system. In particular, you cannot exclude SMTP traffic (or block port 25) or limit to HTTP traffic using the GNUnet configuration. However, you can use your host firewall to restrict outbound connections from the virtual tunnel interface. This is highly recommended. In the future, we plan to offer a wider range of configuration options for exit nodes.

Note that by running an exit node GNUnet will configure your kernel to perform IP-forwarding (for IPv6) and NAT (for IPv4) so that the traffic from the virtual interface can be routed to the Internet. In order to provide an IPv6-exit, you need to have a subnet routed to your host's external network interface and assign a subrange of that subnet to the GNUnet exit's TUN interface.

When running a local service, you should make sure that the local service is (also) bound to the IP address of your EXIT interface (i.e. 169.254.86.1). It will NOT work if your local service is just bound to loopback. You may also want to create a "VPN" record in your zone of the GNU Name System to make it easy for others to access your service via a name instead of just the full service descriptor. Note that the identifier you assign the service can serve as a passphrase or shared secret, clients connecting to the service must somehow learn the service's name. VPN records in the GNU Name System can make this easier.

3.8.2 Fedora and the Firewall

When using an exit node on Fedora 15, the standard firewall can create trouble even when not really exiting the local system! For IPv4, the standard rules seem fine. However, for IPv6 the standard rules prohibit traffic from the network range of the virtual interface created by the exit daemon to the local IPv6 address of the same interface (which is essentially loopback traffic, so you might suspect that a standard firewall would leave this traffic alone). However, as somehow for IPv6 the traffic is not recognized as originating from the local system (and as the connection is not already "established"), the firewall drops the traffic. You should still get ICMPv6 packets back, but that's obviously not very useful.

Possible ways to fix this include disabling the firewall (do you have a good reason for having it on?) or disabling the firewall at least for the GNUnet exit interface (or the respective IPv4/IPv6 address range). The best way to diagnose these kinds of problems in general involves setting the firewall to REJECT instead of DROP and to watch the traffic using wireshark (or tcpdump) to see if ICMP messages are generated when running some tests that should work.

3.8.3 Setting up VPN node for protocol translation and tunneling

The GNUnet VPN/PT subsystem enables you to tunnel IP traffic over the VPN to an exit node, from where it can then be forwarded to the Internet. This section documents how to setup VPN/PT on a node. Note that you can enable both the VPN and an exit on the same peer. In this case, IP traffic from your system may enter your peer's VPN and leave your peer's exit. This can be useful as a means to do protocol translation. For example, you might have an application that supports only IPv4 but needs to access an IPv6-only site. In this case, GNUnet would perform 4to6 protocol translation between the VPN (IPv4) and the Exit (IPv6). Similarly, 6to4 protocol translation is also possible. However, the primary

use for GNUnet would be to access an Internet service running with an IP version that is not supported by your ISP. In this case, your IP traffic would be routed via GNUnet to a peer that has access to the Internet with the desired IP version.

Setting up an entry node into the GNUnet VPN primarily requires you to enable the "VPN/PT" option in "gnunet-setup". This will launch the "gnunet-service-vpn", "gnunet-service-dns" and "gnunet-daemon-pt" processes. The "gnunet-service-vpn" will create a virtual interface which will be used as the target for your IP traffic that enters the VPN. Additionally, a second virtual interface will be created by the "gnunet-service-dns" for your DNS traffic. You will then need to specify which traffic you want to tunnel over GNUnet. If your ISP only provides you with IPv4 or IPv6-access, you may choose to tunnel the other IP protocol over the GNUnet VPN. If you do not have an ISP (and are connected to other GNUnet peers via WLAN), you can also choose to tunnel all IP traffic over GNUnet. This might also provide you with some anonymity. After you enable the respective options and restart your peer, your Internet traffic should be tunneled over the GNUnet VPN.

The GNUnet VPN uses DNS-ALG to hijack your IP traffic. Whenever an application resolves a hostname (i.e. 'gnunet.org'), the "gnunet-daemon-pt" will instruct the "gnunet-service-dns" to intercept the request (possibly route it over GNUnet as well) and replace the normal answer with an IP in the range of the VPN's interface. "gnunet-daemon-pt" will then tell "gnunet-service-vpn" to forward all traffic it receives on the TUN interface via the VPN to the original destination.

For applications that do not use DNS, you can also manually create such a mapping using the gnunet-vpn command-line tool. Here, you specfiy the desired address family of the result (i.e. "-4"), and the intended target IP on the Internet ("-i 131.159.74.67") and "gnunet-vpn" will tell you which IP address in the range of your VPN tunnel was mapped.

`gnunet-vpn` can also be used to access "internal" services offered by GNUnet nodes. So if you happen to know a peer and a service offered by that peer, you can create an IP tunnel to that peer by specifying the peer's identity, service name and protocol (–tcp or –udp) and you will again receive an IP address that will terminate at the respective peer's service.

3.9 The graphical configuration interface

If you also would like to use `gnunet-gtk` and `gnunet-setup` (highly recommended for beginners), do:

3.9.1 Configuring your peer

This chapter will describe the various configuration options in GNUnet.

The easiest way to configure your peer is to use the `gnunet-setup` tool. `gnunet-setup` is part of the `gnunet-gtk` application. You might have to install it separately.

Many of the specific sections from this chapter actually are linked from within `gnunet-setup` to help you while using the setup tool.

While you can also configure your peer by editing the configuration file by hand, this is not recommended for anyone except for developers as it requires a more in-depth understanding of the configuration files and internal dependencies of GNUnet.

3.9.2 Configuring the Friend-to-Friend (F2F) mode

GNUnet knows three basic modes of operation:

- In standard "peer-to-peer" mode, your peer will connect to any peer.
- In the pure "friend-to-friend" mode, your peer will ONLY connect to peers from a list of friends specified in the configuration.
- Finally, in mixed mode, GNUnet will only connect to arbitrary peers if it has at least a specified number of connections to friends.

When configuring any of the F2F ("friend-to-friend") modes, you first need to create a file with the peer identities of your friends. Ask your friends to run

```
$ gnunet-peerinfo -sq
```

The resulting output of this command needs to be added to your `friends` file, which is simply a plain text file with one line per friend with the output from the above command.

You then specify the location of your `friends` file in the `FRIENDS` option of the "topology" section.

Once you have created the `friends` file, you can tell GNUnet to only connect to your friends by setting the `FRIENDS-ONLY` option (again in the "topology" section) to YES.

If you want to run in mixed-mode, set "FRIENDS-ONLY" to NO and configure a minimum number of friends to have (before connecting to arbitrary peers) under the "MINIMUM-FRIENDS" option.

If you want to operate in normal P2P-only mode, simply set `MINIMUM-FRIENDS` to zero and `FRIENDS_ONLY` to NO. This is the default.

3.9.3 Configuring the hostlist to bootstrap

After installing the software you need to get connected to the GNUnet network. The configuration file included in your download is already configured to connect you to the GNUnet network. In this section the relevant configuration settings are explained.

To get an initial connection to the GNUnet network and to get to know peers already connected to the network you can use the so called "bootstrap servers". These servers can give you a list of peers connected to the network. To use these bootstrap servers you have to configure the hostlist daemon to activate bootstrapping.

To activate bootstrapping, edit the `[hostlist]`-section in your configuration file. You have to set the argument `-b` in the options line:

```
[hostlist]
OPTIONS = -b
```

Additionally you have to specify which server you want to use. The default bootstrapping server is "http://v10.gnunet.org/hostlist (`http://v10.gnunet.org/hostlist`)". [^] To set the server you have to edit the line "SERVERS" in the hostlist section. To use the default server you should set the lines to

```
SERVERS = http://v10.gnunet.org/hostlist [^]
```

To use bootstrapping your configuration file should include these lines:

```
[hostlist]
OPTIONS = -b
```

```
SERVERS = http://v10.gnunet.org/hostlist [^]
```

Besides using bootstrap servers you can configure your GNUnet peer to recieve hostlist advertisements. Peers offering hostlists to other peers can send advertisement messages to peers that connect to them. If you configure your peer to receive these messages, your peer can download these lists and connect to the peers included. These lists are persistent, which means that they are saved to your hard disk regularly and are loaded during startup.

To activate hostlist learning you have to add the `-e` switch to the `OPTIONS` line in the hostlist section:

```
[hostlist]
OPTIONS = -b -e
```

Furthermore you can specify in which file the lists are saved. To save the lists in the file `hostlists.file` just add the line:

```
HOSTLISTFILE = hostlists.file
```

Best practice is to activate both bootstrapping and hostlist learning. So your configuration file should include these lines:

```
[hostlist]
OPTIONS = -b -e
HTTPPORT = 8080
SERVERS = http://v10.gnunet.org/hostlist [^]
HOSTLISTFILE = $SERVICEHOME/hostlists.file
```

3.9.4 Configuration of the HOSTLIST proxy settings

The hostlist client can be configured to use a proxy to connect to the hostlist server. This functionality can be configured in the configuration file directly or using the `gnunet-setup` tool.

The hostlist client supports the following proxy types at the moment:

- HTTP and HTTP 1.0 only proxy
- SOCKS 4/4a/5/5 with hostname

In addition authentication at the proxy with username and password can be configured.

To configure proxy support for the hostlist client in the `gnunet-setup` tool, select the "hostlist" tab and select the appropriate proxy type. The hostname or IP address (including port if required) has to be entered in the "Proxy hostname" textbox. If required, enter username and password in the "Proxy username" and "Proxy password" boxes. Be aware that this information will be stored in the configuration in plain text (TODO: Add explanation and generalize the part in Chapter 3.6 about the encrypted home).

To provide these options directly in the configuration, you can enter the following settings in the `[hostlist]` section of the configuration:

```
# Type of proxy server,
# Valid values: HTTP, HTTP_1_0, SOCKS4, SOCKS5, SOCKS4A, SOCKS5_HOSTNAME
# Default: HTTP
# PROXY_TYPE = HTTP

# Hostname or IP of proxy server
```

```
# PROXY =
# User name for proxy server
# PROXY_USERNAME =
# User password for proxy server
# PROXY_PASSWORD =
```

3.9.5 Configuring your peer to provide a hostlist

If you operate a peer permanently connected to GNUnet you can configure your peer to act as a hostlist server, providing other peers the list of peers known to him.

Your server can act as a bootstrap server and peers needing to obtain a list of peers can contact it to download this list. To download this hostlist the peer uses HTTP. For this reason you have to build your peer with libgnurl (or libcurl) and microhttpd support.

To configure your peer to act as a bootstrap server you have to add the `-p` option to `OPTIONS` in the `[hostlist]` section of your configuration file. Besides that you have to specify a port number for the http server. In conclusion you have to add the following lines:

```
[hostlist]
HTTPPORT = 12980
OPTIONS = -p
```

If your peer acts as a bootstrap server other peers should know about that. You can advertise the hostlist your are providing to other peers. Peers connecting to your peer will get a message containing an advertisement for your hostlist and the URL where it can be downloaded. If this peer is in learning mode, it will test the hostlist and, in the case it can obtain the list successfully, it will save it for bootstrapping.

To activate hostlist advertisement on your peer, you have to set the following lines in your configuration file:

```
[hostlist]
EXTERNAL_DNS_NAME = example.org
HTTPPORT = 12981
OPTIONS = -p -a
```

With this configuration your peer will a act as a bootstrap server and advertise this hostlist to other peers connecting to it. The URL used to download the list will be `http://example.org:12981/ (http://example.org:12981/)`.

Please notice:

- The hostlist is **not** human readable, so you should not try to download it using your webbrowser. Just point your GNUnet peer to the address!
- Advertising without providing a hostlist does not make sense and will not work.

3.9.6 Configuring the datastore

The datastore is what GNUnet uses for long-term storage of file-sharing data. Note that long-term does not mean 'forever' since content does have an expiration date, and of course storage space is finite (and hence sometimes content may have to be discarded).

Use the `QUOTA` option to specify how many bytes of storage space you are willing to dedicate to GNUnet.

In addition to specifying the maximum space GNUnet is allowed to use for the datastore, you need to specify which database GNUnet should use to do so. Currently, you have the choice between sqLite, MySQL and Postgres.

3.9.7 Configuring the MySQL database

This section describes how to setup the MySQL database for GNUnet.

Note that the mysql plugin does NOT work with mysql before 4.1 since we need prepared statements. We are generally testing the code against MySQL 5.1 at this point.

3.9.8 Reasons for using MySQL

- On up-to-date hardware wher mysql can be used comfortably, this module will have better performance than the other database choices (according to our tests).
- Its often possible to recover the mysql database from internal inconsistencies. Some of the other databases do not support repair.

3.9.9 Reasons for not using MySQL

- Memory usage (likely not an issue if you have more than 1 GB)
- Complex manual setup

3.9.10 Setup Instructions

- In `gnunet.conf` set in section `DATASTORE` the value for `DATABASE` to `mysql`.
- Access mysql as root:

  ```
  $ mysql -u root -p
  ```

  and issue the following commands, replacing $USER with the username that will be running `gnunet-arm` (so typically "gnunet"):

  ```
  CREATE DATABASE gnunet;
  GRANT select,insert,update,delete,create,alter,drop,create \
  temporary tables ON gnunet.* TO $USER@localhost;
  SET PASSWORD FOR $USER@localhost=PASSWORD('$the_password_you_like');
  FLUSH PRIVILEGES;
  ```

- In the $HOME directory of $USER, create a `.my.cnf` file with the following lines

  ```
  [client]
  user=$USER
  password=$the_password_you_like
  ```

Thats it. Note that `.my.cnf` file is a slight security risk unless its on a safe partition. The `$HOME/.my.cnf` can of course be a symbolic link. Luckily $USER has only priviledges to mess up GNUnet's tables, which should be pretty harmless.

3.9.11 Testing

You should briefly try if the database connection works. First, login as $USER. Then use:

```
$ mysql -u $USER
mysql> use gnunet;
```

If you get the message

```
Database changed
```

it probably works.

If you get

```
ERROR 2002: Can't connect to local MySQL server
through socket '/tmp/mysql.sock' (2)
```

it may be resolvable by

```
ln -s /var/run/mysqld/mysqld.sock /tmp/mysql.sock
```

so there may be some additional trouble depending on your mysql setup.

3.9.12 Performance Tuning

For GNUnet, you probably want to set the option

```
innodb_flush_log_at_trx_commit = 0
```

for a rather dramatic boost in MySQL performance. However, this reduces the "safety" of your database as with this options you may loose transactions during a power outage. While this is totally harmless for GNUnet, the option applies to all applications using MySQL. So you should set it if (and only if) GNUnet is the only application on your system using MySQL.

3.9.13 Setup for running Testcases

If you want to run the testcases, you must create a second database "gnunetcheck" with the same username and password. This database will then be used for testing (`make check`).

3.9.14 Configuring the Postgres database

This text describes how to setup the Postgres database for GNUnet.

This Postgres plugin was developed for Postgres 8.3 but might work for earlier versions as well.

3.9.15 Reasons to use Postgres

- Easier to setup than MySQL
- Real database

3.9.16 Reasons not to use Postgres

- Quite slow
- Still some manual setup required

3.9.17 Manual setup instructions

- In `gnunet.conf` set in section `DATASTORE` the value for `DATABASE` to `postgres`.
- Access Postgres to create a user:

  with Postgres 8.x, use:

  ```
  # su - postgres
  $ createuser
  ```

  and enter the name of the user running GNUnet for the role interactively. Then, when prompted, do not set it to superuser, allow the creation of databases, and do not allow the creation of new roles.

with Postgres 9.x, use:

```
# su - postgres
$ createuser -d $GNUNET_USER
```

where $GNUNET_USER is the name of the user running GNUnet.

- As that user (so typically as user "gnunet"), create a database (or two):

```
$ createdb gnunet
# this way you can run "make check"
$ createdb gnunetcheck
```

Now you should be able to start `gnunet-arm`.

3.9.18 Testing the setup manually

You may want to try if the database connection works. First, again login as the user who will run `gnunet-arm`. Then use:

```
$ psql gnunet # or gnunetcheck
gnunet=> \dt
```

If, after you have started `gnunet-arm` at least once, you get a `gn090` table here, it probably works.

3.9.19 Configuring the datacache

The datacache is what GNUnet uses for storing temporary data. This data is expected to be wiped completely each time GNUnet is restarted (or the system is rebooted).

You need to specify how many bytes GNUnet is allowed to use for the datacache using the `QUOTA` option in the section `[dhtcache]`. Furthermore, you need to specify which database backend should be used to store the data. Currently, you have the choice between sqLite, MySQL and Postgres.

3.9.20 Configuring the file-sharing service

In order to use GNUnet for file-sharing, you first need to make sure that the file-sharing service is loaded. This is done by setting the `AUTOSTART` option in section `[fs]` to "YES". Alternatively, you can run

```
$ gnunet-arm -i fs
```

to start the file-sharing service by hand.

Except for configuring the database and the datacache the only important option for file-sharing is content migration.

Content migration allows your peer to cache content from other peers as well as send out content stored on your system without explicit requests. This content replication has positive and negative impacts on both system performance and privacy.

FIXME: discuss the trade-offs. Here is some older text about it...

Setting this option to YES allows gnunetd to migrate data to the local machine. Setting this option to YES is highly recommended for efficiency. Its also the default. If you set this value to YES, GNUnet will store content on your machine that you cannot decrypt. While this may protect you from liability if the judge is sane, it may not (IANAL). If you put illegal content on your machine yourself, setting this option to YES will probably increase

your chances to get away with it since you can plausibly deny that you inserted the content. Note that in either case, your anonymity would have to be broken first (which may be possible depending on the size of the GNUnet network and the strength of the adversary).

3.9.21 Configuring logging

Logging in GNUnet 0.9.0 is controlled via the "-L" and "-l" options. Using `-L`, a log level can be specified. With log level `ERROR` only serious errors are logged. The default log level is `WARNING` which causes anything of concern to be logged. Log level `INFO` can be used to log anything that might be interesting information whereas `DEBUG` can be used by developers to log debugging messages (but you need to run `./configure` with `--enable-logging=verbose` to get them compiled). The `-l` option is used to specify the log file.

Since most GNUnet services are managed by `gnunet-arm`, using the `-l` or `-L` options directly is not possible. Instead, they can be specified using the `OPTIONS` configuration value in the respective section for the respective service. In order to enable logging globally without editing the `OPTIONS` values for each service, `gnunet-arm` supports a `GLOBAL_POSTFIX` option. The value specified here is given as an extra option to all services for which the configuration does contain a service-specific `OPTIONS` field.

`GLOBAL_POSTFIX` can contain the special sequence "{}" which is replaced by the name of the service that is being started. Furthermore, `GLOBAL_POSTFIX` is special in that sequences starting with "$" anywhere in the string are expanded (according to options in `PATHS`); this expansion otherwise is only happening for filenames and then the "$" must be the first character in the option. Both of these restrictions do not apply to `GLOBAL_POSTFIX`. Note that specifying `%` anywhere in the `GLOBAL_POSTFIX` disables both of these features.

In summary, in order to get all services to log at level `INFO` to log-files called `SERVICENAME-logs`, the following global prefix should be used:

```
GLOBAL_POSTFIX = -l $SERVICEHOME/{}-logs -L INFO
```

3.9.22 Configuring the transport service and plugins

The transport service in GNUnet is responsible to maintain basic connectivity to other peers. Besides initiating and keeping connections alive it is also responsible for address validation.

The GNUnet transport supports more than one transport protocol. These protocols are configured together with the transport service.

The configuration section for the transport service itself is quite similar to all the other services

```
AUTOSTART = YES
@UNIXONLY@ PORT = 2091
HOSTNAME = localhost
HOME = $SERVICEHOME
CONFIG = $DEFAULTCONFIG
BINARY = gnunet-service-transport
#PREFIX = valgrind
NEIGHBOUR_LIMIT = 50
ACCEPT_FROM = 127.0.0.1;
ACCEPT_FROM6 = ::1;
```

```
PLUGINS = tcp udp
UNIXPATH = /tmp/gnunet-service-transport.sock
```

Different are the settings for the plugins to load PLUGINS. The first setting specifies which transport plugins to load.

- transport-unix A plugin for local only communication with UNIX domain sockets. Used for testing and available on unix systems only. Just set the port

  ```
  [transport-unix]
  PORT = 22086
  TESTING_IGNORE_KEYS = ACCEPT_FROM;
  ```

- transport-tcp A plugin for communication with TCP. Set port to 0 for client mode with outbound only connections

  ```
  [transport-tcp]
  # Use 0 to ONLY advertise as a peer behind NAT (no port binding)
  PORT = 2086
  ADVERTISED_PORT = 2086
  TESTING_IGNORE_KEYS = ACCEPT_FROM;
  # Maximum number of open TCP connections allowed
  MAX_CONNECTIONS = 128
  ```

- transport-udp A plugin for communication with UDP. Supports peer discovery using broadcasts.

  ```
  [transport-udp]
  PORT = 2086
  BROADCAST = YES
  BROADCAST_INTERVAL = 30 s
  MAX_BPS = 1000000
  TESTING_IGNORE_KEYS = ACCEPT_FROM;
  ```

- transport-http HTTP and HTTPS support is split in two part: a client plugin initiating outbound connections and a server part accepting connections from the client. The client plugin just takes the maximum number of connections as an argument.

  ```
  [transport-http_client]
  MAX_CONNECTIONS = 128
  TESTING_IGNORE_KEYS = ACCEPT_FROM;
  ```

  ```
  [transport-https_client]
  MAX_CONNECTIONS = 128
  TESTING_IGNORE_KEYS = ACCEPT_FROM;
  ```

  The server has a port configured and the maximum nunber of connections. The HTTPS part has two files with the certificate key and the certificate file.

  The server plugin supports reverse proxies, so a external hostname can be set using the EXTERNAL_HOSTNAME setting. The webserver under this address should forward the request to the peer and the configure port.

  ```
  [transport-http_server]
  EXTERNAL_HOSTNAME = fulcrum.net.in.tum.de/gnunet
  PORT = 1080
  MAX_CONNECTIONS = 128
  ```

```
TESTING_IGNORE_KEYS = ACCEPT_FROM;

[transport-https_server]
PORT = 4433
CRYPTO_INIT = NORMAL
KEY_FILE = https.key
CERT_FILE = https.cert
MAX_CONNECTIONS = 128
TESTING_IGNORE_KEYS = ACCEPT_FROM;
```

- transport-wlan

  The next section describes how to setup the WLAN plugin, so here only the settings. Just specify the interface to use:

  ```
  [transport-wlan]
  # Name of the interface in monitor mode (typically monX)
  INTERFACE = mon0
  # Real hardware, no testing
  TESTMODE = 0
  TESTING_IGNORE_KEYS = ACCEPT_FROM;
  ```

3.9.23 Configuring the wlan transport plugin

The wlan transport plugin enables GNUnet to send and to receive data on a wlan interface. It has not to be connected to a wlan network as long as sender and receiver are on the same channel. This enables you to get connection to GNUnet where no internet access is possible, for example during catastrophes or when censorship cuts you off from the internet.

3.9.23.1 Requirements for the WLAN plugin

- wlan network card with monitor support and packet injection (see aircrack-ng.org (http://www.aircrack-ng.org/))
- Linux kernel with mac80211 stack, introduced in 2.6.22, tested with 2.6.35 and 2.6.38
- Wlantools to create the a monitor interface, tested with airmon-ng of the aircrack-ng package

3.9.23.2 Configuration

There are the following options for the wlan plugin (they should be like this in your default config file, you only need to adjust them if the values are incorrect for your system)

```
# section for the wlan transport plugin
[transport-wlan]
# interface to use, more information in the
# "Before starting GNUnet" section of the handbook.
INTERFACE = mon0
# testmode for developers:
# 0 use wlan interface,
#1 or 2 use loopback driver for tests 1 = server, 2 = client
TESTMODE = 0
```

3.9.23.3 Before starting GNUnet

Before starting GNUnet, you have to make sure that your wlan interface is in monitor mode. One way to put the wlan interface into monitor mode (if your interface name is wlan0) is by executing:

```
sudo airmon-ng start wlan0
```

Here is an example what the result should look like:

```
Interface Chipset Driver
wlan0 Intel 4965 a/b/g/n iwl4965 - [phy0]
(monitor mode enabled on mon0)
```

The monitor interface is mon0 is the one that you have to put into the configuration file.

3.9.23.4 Limitations and known bugs

Wlan speed is at the maximum of 1 Mbit/s because support for choosing the wlan speed with packet injection was removed in newer kernels. Please pester the kernel developers about fixing this.

The interface channel depends on the wlan network that the card is connected to. If no connection has been made since the start of the computer, it is usually the first channel of the card. Peers will only find each other and communicate if they are on the same channel. Channels must be set manually, i.e. using:

```
iwconfig wlan0 channel 1
```

3.9.24 Configuring HTTP(S) reverse proxy functionality using Apache or nginx

The HTTP plugin supports data transfer using reverse proxies. A reverse proxy forwards the HTTP request he receives with a certain URL to another webserver, here a GNUnet peer.

So if you have a running Apache or nginx webserver you can configure it to be a GNUnet reverse proxy. Especially if you have a well-known webiste this improves censorship resistance since it looks as normal surfing behaviour.

To do so, you have to do two things:

- Configure your webserver to forward the GNUnet HTTP traffic
- Configure your GNUnet peer to announce the respective address

As an example we want to use GNUnet peer running:

- HTTP server plugin on `gnunet.foo.org:1080`
- HTTPS server plugin on `gnunet.foo.org:4433`
- A apache or nginx webserver on http://www.foo.org:80/ (`http://www.foo.org/`)
- A apache or nginx webserver on https://www.foo.org:443/

And we want the webserver to accept GNUnet traffic under `http://www.foo.org/bar/`. The required steps are described here:

3.9.24.1 Reverse Proxy - Configure your Apache2 HTTP webserver

First of all you need mod_proxy installed.

Edit your webserver configuration. Edit `/etc/apache2/apache2.conf` or the site-specific configuration file.

In the respective `server config`,`virtual host` or `directory` section add the following lines:

```
ProxyTimeout 300
ProxyRequests Off
<Location /bar/ >
ProxyPass http://gnunet.foo.org:1080/
ProxyPassReverse http://gnunet.foo.org:1080/
</Location>
```

3.9.24.2 Reverse Proxy - Configure your Apache2 HTTPS webserver

We assume that you already have an HTTPS server running, if not please check how to configure a HTTPS host. An uncomplicated to use example is the example configuration file for Apache2/HTTPD provided in `apache2/sites-available/default-ssl`.

In the respective HTTPS `server config`,`virtual host` or `directory` section add the following lines:

```
SSLProxyEngine On
ProxyTimeout 300
ProxyRequests Off
<Location /bar/ >
ProxyPass https://gnunet.foo.org:4433/
ProxyPassReverse https://gnunet.foo.org:4433/
</Location>
```

More information about the apache mod_proxy configuration can be found in the Apache documentation[4]

3.9.24.3 Reverse Proxy - Configure your nginx HTTPS webserver

Since nginx does not support chunked encoding, you first of all have to install the `chunkin` module[5]

To enable chunkin add:

```
chunkin on;
error_page 411 = @my_411_error;
location @my_411_error {
chunkin_resume;
}
```

Edit your webserver configuration. Edit `/etc/nginx/nginx.conf` or the site-specific configuration file.

---

[4] http://httpd.apache.org/docs/2.2/mod/mod_proxy.html#proxypass (`http://httpd.apache.org/docs/2.2/mod/mod_proxy.html#proxypass`)

[5] http://wiki.nginx.org/HttpChunkinModule (`http://wiki.nginx.org/HttpChunkinModule`)

In the `server` section add:

```
location /bar/ {
proxy_pass http://gnunet.foo.org:1080/;
proxy_buffering off;
proxy_connect_timeout 5; # more than http_server
proxy_read_timeout 350; # 60 default, 300s is GNUnet's idle timeout
proxy_http_version 1.1; # 1.0 default
proxy_next_upstream error timeout invalid_header http_500 http_503 http_502 http_504;
}
```

3.9.24.4 Reverse Proxy - Configure your nginx HTTP webserver

Edit your webserver configuration. Edit `/etc/nginx/nginx.conf` or the site-specific configuration file.

In the `server` section add:

```
ssl_session_timeout 6m;
location /bar/
{
proxy_pass https://gnunet.foo.org:4433/;
proxy_buffering off;
proxy_connect_timeout 5; # more than http_server
proxy_read_timeout 350; # 60 default, 300s is GNUnet's idle timeout
proxy_http_version 1.1; # 1.0 default
proxy_next_upstream error timeout invalid_header http_500 http_503 http_502 http_504;
}
```

3.9.24.5 Reverse Proxy - Configure your GNUnet peer

To have your GNUnet peer announce the address, you have to specify the `EXTERNAL_HOSTNAME` option in the `[transport-http_server]` section:

```
[transport-http_server]
EXTERNAL_HOSTNAME = http://www.foo.org/bar/
```

and/or `[transport-https_server]` section:

```
[transport-https_server]
EXTERNAL_HOSTNAME = https://www.foo.org/bar/
```

Now restart your webserver and your peer...

3.9.25 Blacklisting peers

Transport service supports to deny connecting to a specific peer of to a specific peer with a specific transport plugin using te blacklisting component of transport service. With blacklisting it is possible to deny connections to specific peers of to use a specific plugin to a specific peer. Peers can be blacklisted using the configuration or a blacklist client can be asked.

To blacklist peers using the configuration you have to add a section to your configuration containing the peer id of the peer to blacklist and the plugin if required.

Examples:

To blacklist connections to P565... on peer AG2P... using tcp add:

```
[transport-blacklist AG2PHES1BARB9IJCPAMJTFPVJ5V3A72S3F2A8SBUB8DAQ2V0O3V8G6G2JU56FHGFOH
P565723JO1C2HSN6J29TAQ22MN6CI8HTMUU55T0FUQG4CMDGGEQ8UCNBKUMB94GC8R9G4FB2SF9LDOBAJ6AMINB
```

To blacklist connections to P565... on peer AG2P... using all plugins add:

```
[transport-blacklist-AG2PHES1BARB9IJCPAMJTFPVJ5V3A72S3F2A8SBUB8DAQ2V0O3V8G6G2JU56FHGFOH
P565723JO1C2HSN6J29TAQ22MN6CI8HTMUU55T0FUQG4CMDGGEQ8UCNBKUMB94GC8R9G4FB2SF9LDOBAJ6AMINB
```

You can also add a blacklist client usign the blacklist API. On a blacklist check, blacklisting first checks internally if the peer is blacklisted and if not, it asks the blacklisting clients. Clients are asked if it is OK to connect to a peer ID, the plugin is omitted.

On blacklist check for (peer, plugin)

- Do we have a local blacklist entry for this peer and this plugin?
- YES: disallow connection
- Do we have a local blacklist entry for this peer and all plugins?
- YES: disallow connection
- Does one of the clients disallow?
- YES: disallow connection

3.9.26 Configuration of the HTTP and HTTPS transport plugins

The client parts of the http and https transport plugins can be configured to use a proxy to connect to the hostlist server. This functionality can be configured in the configuration file directly or using the gnunet-setup tool.

Both the HTTP and HTTPS clients support the following proxy types at the moment:

- HTTP 1.1 proxy
- SOCKS 4/4a/5/5 with hostname

In addition authentication at the proxy with username and password can be configured.

To configure proxy support for the clients in the gnunet-setup tool, select the "transport" tab and activate the respective plugin. Now you can select the appropriate proxy type. The hostname or IP address (including port if required) has to be entered in the "Proxy hostname" textbox. If required, enter username and password in the "Proxy username" and "Proxy password" boxes. Be aware that these information will be stored in the configuration in plain text.

To configure these options directly in the configuration, you can configure the following settings in the `[transport-http_client]` and `[transport-https_client]` section of the configuration:

```
# Type of proxy server,
# Valid values: HTTP, SOCKS4, SOCKS5, SOCKS4A, SOCKS5_HOSTNAME
# Default: HTTP
# PROXY_TYPE = HTTP

# Hostname or IP of proxy server
# PROXY =
# User name for proxy server
```

```
# PROXY_USERNAME =
# User password for proxy server
# PROXY_PASSWORD =
```

3.9.27 Configuring the GNU Name System

3.9.27.1 Configuring system-wide DNS interception

Before you install GNUnet, make sure you have a user and group 'gnunet' as well as an empty group 'gnunetdns'.

When using GNUnet with system-wide DNS interception, it is absolutely necessary for all GNUnet service processes to be started by `gnunet-service-arm` as user and group 'gnunet'. You also need to be sure to run `make install` as root (or use the `sudo` option to configure) to grant GNUnet sufficient privileges.

With this setup, all that is required for enabling system-wide DNS interception is for some GNUnet component (VPN or GNS) to request it. The `gnunet-service-dns` will then start helper programs that will make the necessary changes to your firewall (`iptables`) rules.

Note that this will NOT work if your system sends out DNS traffic to a link-local IPv6 address, as in this case GNUnet can intercept the traffic, but not inject the responses from the link-local IPv6 address. Hence you cannot use system-wide DNS interception in conjunction with link-local IPv6-based DNS servers. If such a DNS server is used, it will bypass GNUnet's DNS traffic interception.

Using the GNU Name System (GNS) requires two different configuration steps. First of all, GNS needs to be integrated with the operating system. Most of this section is about the operating system level integration.

The remainder of this chapter will detail the various methods for configuring the use of GNS with your operating system.

At this point in time you have different options depending on your OS:

Use the gnunet-gns-proxy This approach works for all operating
: systems and is likely the easiest. However, it enables GNS only for browsers, not for other applications that might be using DNS, such as SSH. Still, using the proxy is required for using HTTP with GNS and is thus recommended for all users. To do this, you simply have to run the `gnunet-gns-proxy-setup-ca` script as the user who will run the browser (this will create a GNS certificate authority (CA) on your system and import its key into your browser), then start `gnunet-gns-proxy` and inform your browser to use the Socks5 proxy which `gnunet-gns-proxy` makes available by default on port 7777.

Use a nsswitch plugin (recommended on GNU systems)
: This approach has the advantage of offering fully personalized resolution even on multi-user systems. A potential disadvantage is that some applications might be able to bypass GNS.

Use a W32 resolver plugin (recommended on W32)
: This is currently the only option on W32 systems.

Use system-wide DNS packet interception
: This approach is recommended for the GNUnet VPN. It can be used to handle GNS at the same time; however, if you only use this method, you will only get one root zone per machine (not so great for multi-user systems).

You can combine system-wide DNS packet interception with the nsswitch plugin. The setup of the system-wide DNS interception is described here. All of the other GNS-specific configuration steps are described in the following sections.

3.9.27.2 Configuring the GNS nsswitch plugin

The Name Service Switch (NSS) is a facility in Unix-like operating systems[6] that provides a variety of sources for common configuration databases and name resolution mechanisms. A superuser (system administrator) usually configures the operating system's name services using the file `/etc/nsswitch.conf`.

GNS provides a NSS plugin to integrate GNS name resolution with the operating system's name resolution process. To use the GNS NSS plugin you have to either

- install GNUnet as root or
- compile GNUnet with the `--with-sudo=yes` switch.

Name resolution is controlled by the *hosts* section in the NSS configuration. By default this section first performs a lookup in the `/etc/hosts` file and then in DNS. The nsswitch file should contain a line similar to:

```
hosts: files dns [NOTFOUND=return] mdns4_minimal mdns4
```

Here the GNS NSS plugin can be added to perform a GNS lookup before performing a DNS lookup. The GNS NSS plugin has to be added to the "hosts" section in `/etc/nsswitch.conf` file before DNS related plugins:

```
...
hosts: files gns [NOTFOUND=return] dns mdns4_minimal mdns4
...
```

The `NOTFOUND=return` will ensure that if a `.gnu` name is not found in GNS it will not be queried in DNS.

3.9.27.3 Configuring GNS on W32

This document is a guide to configuring GNU Name System on W32-compatible platforms.

After GNUnet is installed, run the w32nsp-install tool:

```
w32nsp-install.exe libw32nsp-0.dll
```

('0' is the library version of W32 NSP; it might increase in the future, change the invocation accordingly).

This will install GNS namespace provider into the system and allow other applications to resolve names that end in '**gnu**' and '**zkey**'. Note that namespace provider requires gnunet-gns-helper-service-w32 to be running, as well as gns service itself (and its usual dependencies).

Namespace provider is hardcoded to connect to **127.0.0.1:5353**, and this is where gnunet-gns-helper-service-w32 should be listening to (and is configured to listen to by default).

[6] More accurate: NSS is a functionality of the GNU C Library

To uninstall the provider, run:

```
w32nsp-uninstall.exe
```

(uses provider GUID to uninstall it, does not need a dll name).

Note that while MSDN claims that other applications will only be able to use the new namespace provider after re-starting, in reality they might stat to use it without that. Conversely, they might stop using the provider after it's been uninstalled, even if they were not re-started. W32 will not permit namespace provider library to be deleted or overwritten while the provider is installed, and while there is at least one process still using it (even after it was uninstalled).

3.9.27.4 GNS Proxy Setup

When using the GNU Name System (GNS) to browse the WWW, there are several issues that can be solved by adding the GNS Proxy to your setup:

- If the target website does not support GNS, it might assume that it is operating under some name in the legacy DNS system (such as example.com). It may then attempt to set cookies for that domain, and the web server might expect a `Host: example.com` header in the request from your browser. However, your browser might be using `example.gnu` for the `Host` header and might only accept (and send) cookies for `example.gnu`. The GNS Proxy will perform the necessary translations of the hostnames for cookies and HTTP headers (using the LEHO record for the target domain as the desired substitute).
- If using HTTPS, the target site might include an SSL certificate which is either only valid for the LEHO domain or might match a TLSA record in GNS. However, your browser would expect a valid certificate for `example.gnu`, not for some legacy domain name. The proxy will validate the certificate (either against LEHO or TLSA) and then on-the-fly produce a valid certificate for the exchange, signed by your own CA. Assuming you installed the CA of your proxy in your browser's certificate authority list, your browser will then trust the HTTPS/SSL/TLS connection, as the hostname mismatch is hidden by the proxy.
- Finally, the proxy will in the future indicate to the server that it speaks GNS, which will enable server operators to deliver GNS-enabled web sites to your browser (and continue to deliver legacy links to legacy browsers)

3.9.27.5 Setup of the GNS CA

First you need to create a CA certificate that the proxy can use. To do so use the provided script gnunet-gns-proxy-ca:

```
$ gnunet-gns-proxy-setup-ca
```

This will create a personal certification authority for you and add this authority to the firefox and chrome database. The proxy will use the this CA certificate to generate `*.gnu` client certificates on the fly.

Note that the proxy uses libcurl. Make sure your version of libcurl uses GnuTLS and NOT OpenSSL. The proxy will **not** work with libcurl compiled against OpenSSL.

You can check the configuration your libcurl was build with by running:

```
curl --version
```

the output will look like this (without the linebreaks):

```
gnurl --version
curl 7.56.0 (x86_64-unknown-linux-gnu) libcurl/7.56.0 \
GnuTLS/3.5.13 zlib/1.2.11 libidn2/2.0.4
Release-Date: 2017-10-08
Protocols: http https
Features: AsynchDNS IDN IPv6 Largefile NTLM SSL libz \
TLS-SRP UnixSockets HTTPS-proxy
```

3.9.27.6 Testing the GNS setup

Now for testing purposes we can create some records in our zone to test the SSL functionality of the proxy:

```
$ gnunet-identity -C test
$ gnunet-namestore -a -e "1 d" -n "homepage" \
  -t A -V 131.159.74.67 -z test
$ gnunet-namestore -a -e "1 d" -n "homepage" \
  -t LEHO -V "gnunet.org" -z test
```

At this point we can start the proxy. Simply execute

```
$ gnunet-gns-proxy
```

Configure your browser to use this SOCKSv5 proxy on port 7777 and visit this link. If you use `Firefox` (or one of its deriviates/forks such as Icecat) you also have to go to `about:config` and set the key `network.proxy.socks_remote_dns` to `true`.

When you visit `https://homepage.test/`, you should get to the `https://gnunet.org/` frontpage and the browser (with the correctly configured proxy) should give you a valid SSL certificate for `homepage.gnu` and no warnings. It should look like this:

3.9.28 Configuring the GNUnet VPN

Before configuring the GNUnet VPN, please make sure that system-wide DNS interception is configured properly as described in the section on the GNUnet DNS setup. see Section 3.9.27 [Configuring the GNU Name System], page 48, if you haven't done so already.

The default options for the GNUnet VPN are usually sufficient to use GNUnet as a Layer 2 for your Internet connection. However, what you always have to specify is which IP protocol you want to tunnel: IPv4, IPv6 or both. Furthermore, if you tunnel both, you most likely should also tunnel all of your DNS requests. You theoretically can tunnel "only" your DNS traffic, but that usually makes little sense.

The other options as shown on the gnunet-setup tool are:

3.9.28.1 IPv4 address for interface

This is the IPv4 address the VPN interface will get. You should pick an 'private' IPv4 network that is not yet in use for you system. For example, if you use `10.0.0.1/255.255.0.0` already, you might use `10.1.0.1/255.255.0.0`. If you use `10.0.0.1/255.0.0.0` already, then you might use `192.168.0.1/255.255.0.0`. If your system is not in a private IP-network, using any of the above will work fine. You should try to make the mask of the address big enough (`255.255.0.0` or, even better, `255.0.0.0`) to allow more mappings of

remote IP Addresses into this range. However, even a `255.255.255.0` mask will suffice for most users.

3.9.28.2 IPv6 address for interface

The IPv6 address the VPN interface will get. Here you can specify any non-link-local address (the address should not begin with `fe80:`). A subnet Unique Local Unicast (`fd00::/8` prefix) that you are currently not using would be a good choice.

3.9.28.3 Configuring the GNUnet VPN DNS

To resolve names for remote nodes, activate the DNS exit option.

3.9.28.4 Configuring the GNUnet VPN Exit Service

If you want to allow other users to share your Internet connection (yes, this may be dangerous, just as running a Tor exit node) or want to provide access to services on your host (this should be less dangerous, as long as those services are secure), you have to enable the GNUnet exit daemon.

You then get to specify which exit functions you want to provide. By enabling the exit daemon, you will always automatically provide exit functions for manually configured local services (this component of the system is under development and not documented further at this time). As for those services you explicitly specify the target IP address and port, there is no significant security risk in doing so.

Furthermore, you can serve as a DNS, IPv4 or IPv6 exit to the Internet. Being a DNS exit is usually pretty harmless. However, enabling IPv4 or IPv6-exit without further precautions may enable adversaries to access your local network, send spam, attack other systems from your Internet connection and to other mischief that will appear to come from your machine. This may or may not get you into legal trouble. If you want to allow IPv4 or IPv6-exit functionality, you should strongly consider adding additional firewall rules manually to protect your local network and to restrict outgoing TCP traffic (i.e. by not allowing access to port 25). While we plan to improve exit-filtering in the future, you're currently on your own here. Essentially, be prepared for any kind of IP-traffic to exit the respective TUN interface (and GNUnet will enable IP-forwarding and NAT for the interface automatically).

Additional configuration options of the exit as shown by the gnunet-setup tool are:

3.9.28.5 IP Address of external DNS resolver

If DNS traffic is to exit your machine, it will be send to this DNS resolver. You can specify an IPv4 or IPv6 address.

3.9.28.6 IPv4 address for Exit interface

This is the IPv4 address the Interface will get. Make the mask of the address big enough (255.255.0.0 or, even better, 255.0.0.0) to allow more mappings of IP addresses into this range. As for the VPN interface, any unused, private IPv4 address range will do.

3.9.28.7 IPv6 address for Exit interface

The public IPv6 address the interface will get. If your kernel is not a very recent kernel and you are willing to manually enable IPv6-NAT, the IPv6 address you specify here must be a globally routed IPv6 address of your host.

Suppose your host has the address `2001:4ca0::1234/64`, then using `2001:4ca0::1:0/112` would be fine (keep the first 64 bits, then change at least one bit in the range before the bitmask, in the example above we changed bit 111 from 0 to 1).

You may also have to configure your router to route traffic for the entire subnet (`2001:4ca0::1:0/112` for example) through your computer (this should be automatic with IPv6, but obviously anything can be disabled).

3.9.29 Bandwidth Configuration

You can specify how many bandwidth GNUnet is allowed to use to receive and send data. This is important for users with limited bandwidth or traffic volume.

3.9.30 Configuring NAT

Most hosts today do not have a normal global IP address but instead are behind a router performing Network Address Translation (NAT) which assigns each host in the local network a private IP address. As a result, these machines cannot trivially receive inbound connections from the Internet. GNUnet supports NAT traversal to enable these machines to receive incoming connections from other peers despite their limitations.

In an ideal world, you can press the "Attempt automatic configuration" button in gnunet-setup to automatically configure your peer correctly. Alternatively, your distribution might have already triggered this automatic configuration during the installation process. However, automatic configuration can fail to determine the optimal settings, resulting in your peer either not receiving as many connections as possible, or in the worst case it not connecting to the network at all.

To manually configure the peer, you need to know a few things about your network setup. First, determine if you are behind a NAT in the first place. This is always the case if your IP address starts with "10.*" or "192.168.*". Next, if you have control over your NAT router, you may choose to manually configure it to allow GNUnet traffic to your host. If you have configured your NAT to forward traffic on ports 2086 (and possibly 1080) to your host, you can check the "NAT ports have been opened manually" option, which corresponds to the "PUNCHED_NAT" option in the configuration file. If you did not punch your NAT box, it may still be configured to support UPnP, which allows GNUnet to automatically configure it. In that case, you need to install the "upnpc" command, enable UPnP (or PMP) on your NAT box and set the "Enable NAT traversal via UPnP or PMP" option (corresponding to "ENABLE_UPNP" in the configuration file).

Some NAT boxes can be traversed using the autonomous NAT traversal method. This requires certain GNUnet components to be installed with "SUID" prividledges on your system (so if you're installing on a system you do not have administrative rights to, this will not work). If you installed as 'root', you can enable autonomous NAT traversal by checking the "Enable NAT traversal using ICMP method". The ICMP method requires a way to determine your NAT's external (global) IP address. This can be done using either UPnP,

DynDNS, or by manual configuration. If you have a DynDNS name or know your external IP address, you should enter that name under "External (public) IPv4 address" (which corresponds to the "EXTERNAL_ADDRESS" option in the configuration file). If you leave the option empty, GNUnet will try to determine your external IP address automatically (which may fail, in which case autonomous NAT traversal will then not work).

Finally, if you yourself are not behind NAT but want to be able to connect to NATed peers using autonomous NAT traversal, you need to check the "Enable connecting to NATed peers using ICMP method" box.

3.9.31 Peer configuration for distributions

The "GNUNET_DATA_HOME" in "[path]" in `/etc/gnunet.conf` should be manually set to "/var/lib/gnunet/data/" as the default "~/.local/share/gnunet/" is probably not that appropriate in this case. Similarly, distributions may consider pointing "GNUNET_RUNTIME_DIR" to "/var/run/gnunet/" and "GNUNET_HOME" to "/var/lib/gnunet/". Also, should a distribution decide to override system defaults, all of these changes should be done in a custom `/etc/gnunet.conf` and not in the files in the `config.d/` directory.

Given the proposed access permissions, the "gnunet-setup" tool must be run as use "gnunet" (and with option "-c /etc/gnunet.conf" so that it modifies the system configuration). As always, gnunet-setup should be run after the GNUnet peer was stopped using "gnunet-arm -e". Distributions might want to include a wrapper for gnunet-setup that allows the desktop-user to "sudo" (i.e. using gtksudo) to the "gnunet" user account and then runs "gnunet-arm -e", "gnunet-setup" and "gnunet-arm -s" in sequence.

3.10 How to start and stop a GNUnet peer

This section describes how to start a GNUnet peer. It assumes that you have already compiled and installed GNUnet and its' dependencies. Before you start a GNUnet peer, you may want to create a configuration file using gnunet-setup (but you do not have to). Sane defaults should exist in your `$GNUNET_PREFIX/share/gnunet/config.d/` directory, so in practice you could simply start without any configuration. If you want to configure your peer later, you need to stop it before invoking the `gnunet-setup` tool to customize further and to test your configuration (`gnunet-setup` has build-in test functions).

The most important option you might have to still set by hand is in [PATHS]. Here, you use the option "GNUNET_HOME" to specify the path where GNUnet should store its data. It defaults to `$HOME/`, which again should work for most users. Make sure that the directory specified as GNUNET_HOME is writable to the user that you will use to run GNUnet (note that you can run frontends using other users, GNUNET_HOME must only be accessible to the user used to run the background processes).

You will also need to make one central decision: should all of GNUnet be run under your normal UID, or do you want distinguish between system-wide (user-independent) GNUnet services and personal GNUnet services. The multi-user setup is slightly more complicated, but also more secure and generally recommended.

3.10.1 The Single-User Setup

For the single-user setup, you do not need to do anything special and can just start the GNUnet background processes using `gnunet-arm`. By default, GNUnet looks in `~/.config/gnunet.conf` for a configuration (or `$XDG_CONFIG_HOME/gnunet.conf` if `$XDG_CONFIG_HOME` is defined). If your configuration lives elsewhere, you need to pass the `-c FILENAME` option to all GNUnet commands.

Assuming the configuration file is called `~/.config/gnunet.conf`, you start your peer using the `gnunet-arm` command (say as user `gnunet`) using:

```
gnunet-arm -c ~/.config/gnunet.conf -s
```

The "-s" option here is for "start". The command should return almost instantly. If you want to stop GNUnet, you can use:

```
gnunet-arm -c ~/.config/gnunet.conf -e
```

The "-e" option here is for "end".

Note that this will only start the basic peer, no actual applications will be available. If you want to start the file-sharing service, use (after starting GNUnet):

```
gnunet-arm -c ~/.config/gnunet.conf -i fs
```

The "-i fs" option here is for "initialize" the "fs" (file-sharing) application. You can also selectively kill only file-sharing support using

```
gnunet-arm -c ~/.config/gnunet.conf -k fs
```

Assuming that you want certain services (like file-sharing) to be always automatically started whenever you start GNUnet, you can activate them by setting "FORCESTART=YES" in the respective section of the configuration file (for example, "[fs]"). Then GNUnet with file-sharing support would be started whenever you enter:

```
gnunet-arm -c ~/.config/gnunet.conf -s
```

Alternatively, you can combine the two options:

```
gnunet-arm -c ~/.config/gnunet.conf -s -i fs
```

Using `gnunet-arm` is also the preferred method for initializing GNUnet from `init`.

Finally, you should edit your `crontab` (using the `crontab` command) and insert a line

```
@reboot gnunet-arm -c ~/.config/gnunet.conf -s
```

to automatically start your peer whenever your system boots.

3.10.2 The Multi-User Setup

This requires you to create a user `gnunet` and an additional group `gnunetdns`, prior to running `make install` during installation. Then, you create a configuration file `/etc/gnunet.conf` which should contain the lines:

```
[arm]
SYSTEM_ONLY = YES
USER_ONLY = NO
```

Then, perform the same steps to run GNUnet as in the per-user configuration, except as user `gnunet` (including the `crontab` installation). You may also want to run `gnunet-setup` to configure your peer (databases, etc.). Make sure to pass `-c /etc/gnunet.conf` to all

commands. If you run `gnunet-setup` as user `gnunet`, you might need to change permissions on `/etc/gnunet.conf` so that the `gnunet` user can write to the file (during setup).

Afterwards, you need to perform another setup step for each normal user account from which you want to access GNUnet. First, grant the normal user (`$USER`) permission to the group gnunet:

```
# adduser $USER gnunet
```

Then, create a configuration file in `~/.config/gnunet.conf` for the $USER with the lines:

```
[arm]
SYSTEM_ONLY = NO
USER_ONLY = YES
```

This will ensure that `gnunet-arm` when started by the normal user will only run services that are per-user, and otherwise rely on the system-wide services. Note that the normal user may run gnunet-setup, but the configuration would be ineffective as the system-wide services will use `/etc/gnunet.conf` and ignore options set by individual users.

Again, each user should then start the peer using `gnunet-arm -s` — and strongly consider adding logic to start the peer automatically to their crontab.

Afterwards, you should see two (or more, if you have more than one USER) `gnunet-service-arm` processes running in your system.

3.10.3 Killing GNUnet services

It is not necessary to stop GNUnet services explicitly when shutting down your computer.

It should be noted that manually killing "most" of the `gnunet-service` processes is generally not a successful method for stopping a peer (since `gnunet-service-arm` will instantly restart them). The best way to explicitly stop a peer is using `gnunet-arm -e`; note that the per-user services may need to be terminated before the system-wide services will terminate normally.

3.10.4 Access Control for GNUnet

This chapter documents how we plan to make access control work within the GNUnet system for a typical peer. It should be read as a best-practice installation guide for advanced users and builders of binary distributions. The recommendations in this guide apply to POSIX-systems with full support for UNIX domain sockets only.

Note that this is an advanced topic. The discussion presumes a very good understanding of users, groups and file permissions. Normal users on hosts with just a single user can just install GNUnet under their own account (and possibly allow the installer to use SUDO to grant additional permissions for special GNUnet tools that need additional rights). The discussion below largely applies to installations where multiple users share a system and to installations where the best possible security is paramount.

A typical GNUnet system consists of components that fall into four categories:

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

System services and support tools
: System services should always run and offer services that can then be accessed by the normal users. System services do not require special permissions, but as they are not specific to a particular user, they probably should not run as a particular user. Also, there should typically only be one GNUnet peer per host. System services include the gnunet-service and gnunet-daemon programs; support tools include command-line programs such as gnunet-arm.

Priviledged helpers
: Some GNUnet components require root rights to open raw sockets or perform other special operations. These gnunet-helper binaries are typically installed SUID and run from services or daemons.

Critical services
: Some GNUnet services (such as the DNS service) can manipulate the service in deep and possibly highly security sensitive ways. For example, the DNS service can be used to intercept and alter any DNS query originating from the local machine. Access to the APIs of these critical services and their priviledged helpers must be tightly controlled.

3.10.4.1 Recommendation - Disable access to services via TCP

GNUnet services allow two types of access: via TCP socket or via UNIX domain socket. If the service is available via TCP, access control can only be implemented by restricting connections to a particular range of IP addresses. This is acceptable for non-critical services that are supposed to be available to all users on the local system or local network. However, as TCP is generally less efficient and it is rarely the case that a single GNUnet peer is supposed to serve an entire local network, the default configuration should disable TCP access to all GNUnet services on systems with support for UNIX domain sockets. As of GNUnet 0.9.2, configuration files with TCP access disabled should be generated by default. Users can re-enable TCP access to particular services simply by specifying a non-zero port number in the section of the respective service.

3.10.4.2 Recommendation - Run most services as system user "gnunet"

GNUnet's main services should be run as a separate user "gnunet" in a special group "gnunet". The user "gnunet" should start the peer using "gnunet-arm -s" during system startup. The home directory for this user should be `/var/lib/gnunet` and the configuration file should be `/etc/gnunet.conf`. Only the `gnunet` user should have the right to access `/var/lib/gnunet` (*mode: 700*).

3.10.4.3 Recommendation - Control access to services using group "gnunet"

Users that should be allowed to use the GNUnet peer should be added to the group "gnunet". Using GNUnet's access control mechanism for UNIX domain sockets, those services that are considered useful to ordinary users should be made available by setting "UNIX_MATCH_GID=YES" for those services. Again, as shipped, GNUnet provides reasonable defaults. Permissions to access the transport and core subsystems might additionally be granted without necessarily causing security concerns. Some services, such as DNS,

must NOT be made accessible to the "gnunet" group (and should thus only be accessible to the "gnunet" user and services running with this UID).

3.10.4.4 Recommendation - Limit access to certain SUID binaries by group "gnunet"

Most of GNUnet's SUID binaries should be safe even if executed by normal users. However, it is possible to reduce the risk a little bit more by making these binaries owned by the group "gnunet" and restricting their execution to user of the group "gnunet" as well (4750).

3.10.4.5 Recommendation - Limit access to critical gnunet-helper-dns to group "gnunetdns"

A special group "gnunetdns" should be created for controlling access to the "gnunet-helper-dns". The binary should then be owned by root and be in group "gnunetdns" and be installed SUID and only be group-executable (2750). **Note that the group "gnunetdns" should have no users in it at all, ever.** The "gnunet-service-dns" program should be executed by user "gnunet" (via gnunet-service-arm) with the binary owned by the user "root" and the group "gnunetdns" and be SGID (2700). This way, **only** "gnunet-service-dns" can change its group to "gnunetdns" and execute the helper, and the helper can then run as root (as per SUID). Access to the API offered by "gnunet-service-dns" is in turn restricted to the user "gnunet" (not the group!), which means that only "benign" services can manipulate DNS queries using "gnunet-service-dns".

3.10.4.6 Differences between "make install" and these recommendations

The current build system does not set all permissions automatically based on the recommendations above. In particular, it does not use the group "gnunet" at all (so setting gnunet-helpers other than the gnunet-helper-dns to be owned by group "gnunet" must be done manually). Furthermore, 'make install' will silently fail to set the DNS binaries to be owned by group "gnunetdns" unless that group already exists (!). An alternative name for the "gnunetdns" group can be specified using the `--with-gnunetdns=GRPNAME` configure option.

4 GNUnet Contributors Handbook

4.1 Contributing to GNUnet

4.2 Licenses of contributions

GNUnet is a GNU (`https://www.gnu.org/`) package. All code contributions must thus be put under the GNU Public License (GPL) (`https://www.gnu.org/copyleft/gpl.html`). All documentation should be put under FSF approved licenses (see fdl (`https://www.gnu.org/copyleft/fdl.html`)).

By submitting documentation, translations, and other content to GNUnet you automatically grant the right to publish code under the GNU Public License and documentation under either or both the GNU Public License or the GNU Free Documentation License. When contributing to the GNUnet project, GNU standards and the GNU philosophy (`https://www.gnu.org/philosophy/philosophy.html`) should be adhered to.

4.3 Copyright Assignment

We require a formal copyright assignment for GNUnet contributors to GNUnet e.V.; nevertheless, we do allow pseudonymous contributions. By signing the copyright agreement and submitting your code (or documentation) to us, you agree to share the rights to your code with GNUnet e.V.; GNUnet e.V. receives non-exclusive ownership rights, and in particular is allowed to dual-license the code. You retain non-exclusive rights to your contributions, so you can also share your contributions freely with other projects.

GNUnet e.V. will publish all accepted contributions under the GPLv3 or any later version. The association may decide to publish contributions under additional licenses (dual-licensing).

We do not intentionally remove your name from your contributions; however, due to extensive editing it is not always trivial to attribute contributors properly. If you find that you significantly contributed to a file (or the project as a whole) and are not listed in the respective authors file or section, please do let us know.

4.4 Contributing to the Reference Manual

- When writing documentation, please use gender-neutral wording (`https://en.wikipedia.org/wiki/Singular_they`) when referring to people, such as singular “they”, “their”, “them”, and so forth.
- Keep line length below 74 characters, except for URLs. URLs break in the PDF output when they contain linebreaks.
- Do not use tab characters (see chapter 2.1 texinfo manual)
- Write texts in the third person perspective.
- Use @footnote{} instead of putting an @*ref{} to the footnote on a collected footnote-page. In a 200+ pages handbook it’s better to have footnotes accessible without having to skip over to the end.

5 GNUnet Developer Handbook

0This book is intended to be an introduction for programmers that want to extend the GNUnet framework. GNUnet is more than a simple peer-to-peer application.

For developers, GNUnet is:

- developed by a community that believes in the GNU philosophy
- Free Software (Free as in Freedom), licensed under the GNU Affero General Public License[1]
- A set of standards, including coding conventions and architectural rules
- A set of layered protocols, both specifying the communication between peers as well as the communication between components of a single peer
- A set of libraries with well-defined APIs suitable for writing extensions

In particular, the architecture specifies that a peer consists of many processes communicating via protocols. Processes can be written in almost any language. `C`, `Java` and `Guile` APIs exist for accessing existing services and for writing extensions. It is possible to write extensions in other languages by implementing the necessary IPC protocols.

GNUnet can be extended and improved along many possible dimensions, and anyone interested in Free Software and Freedom-enhancing Networking is welcome to join the effort. This Developer Handbook attempts to provide an initial introduction to some of the key design choices and central components of the system. This part of the GNUNet documentation is far from complete, and we welcome informed contributions, be it in the form of new chapters, sections or insightful comments.

5.1 Developer Introduction

This Developer Handbook is intended as first introduction to GNUnet for new developers that want to extend the GNUnet framework. After the introduction, each of the GNUnet subsystems (directories in the `src/` tree) is (supposed to be) covered in its own chapter. In addition to this documentation, GNUnet developers should be aware of the services available on the GNUnet server to them.

New developers can have a look a the GNUnet tutorials for C and java available in the `src/` directory of the repository or under the following links:

- See *The GNUnet C Tutorial*.
- GNUnet Java tutorial

In addition to the GNUnet Reference Documentation you are reading, the GNUnet server at `https://gnunet.org` contains various resources for GNUnet developers and those who aspire to become regular contributors. They are all conveniently reachable via the "Developer" entry in the navigation menu. Some additional tools (such as static analysis reports) require a special developer access to perform certain operations. If you want (or require) access, you should contact Christian Grothoff (`http://grothoff.org/christian/`), GNUnet's maintainer.

[1] https://www.gnu.org/licenses/licenses.html#AGPL (`https://www.gnu.org/licenses/licenses.html#AGPL`)

The public subsystems on the GNUnet server that help developers are:

- The version control system (git) keeps our code and enables distributed development. It is publicly accessible at `https://gnunet.org/git/`. Only developers with write access can commit code, everyone else is encouraged to submit patches to the GNUnet-developers mailinglist: https://lists.gnu.org/mailman/listinfo/gnunet-developers (`https://lists.gnu.org/mailman/listinfo/gnunet-developers`)
- The bugtracking system (Mantis). We use it to track feature requests, open bug reports and their resolutions. It can be accessed at https://gnunet.org/bugs/ (`https://gnunet.org/bugs/`). Anyone can report bugs.
- Our site installation of the CI[2] system `Buildbot` is used to check GNUnet builds automatically on a range of platforms. The web interface of this CI is exposed at https://gnunet.org/buildbot/ (`https://gnunet.org/buildbot/`). Builds are triggered automatically 30 minutes after the last commit to our repository was made.
- The current quality of our automated test suite is assessed using Code coverage analysis. This analysis is run daily; however the webpage is only updated if all automated tests pass at that time. Testcases that improve our code coverage are always welcome.
- We try to automatically find bugs using a static analysis scan. This scan is run daily; however the webpage is only updated if all automated tests pass at the time. Note that not everything that is flagged by the analysis is a bug, sometimes even good code can be marked as possibly problematic. Nevertheless, developers are encouraged to at least be aware of all issues in their code that are listed.
- We use Gauger for automatic performance regression visualization. Details on how to use Gauger are here.
- We use junit (`http://junit.org/`) to automatically test `gnunet-java`. Automatically generated, current reports on the test suite are here.
- We use Cobertura to generate test coverage reports for gnunet-java. Current reports on test coverage are here.

5.1.1 Project overview

The GNUnet project consists at this point of several sub-projects. This section is supposed to give an initial overview about the various sub-projects. Note that this description also lists projects that are far from complete, including even those that have literally not a single line of code in them yet.

GNUnet sub-projects in order of likely relevance are currently:

`gnunet`
: Core of the P2P framework, including file-sharing, VPN and chat applications; this is what the Developer Handbook covers mostly

`gnunet-gtk`
: Gtk+-based user interfaces, including:
  - `gnunet-fs-gtk` (file-sharing),
  - `gnunet-statistics-gtk` (statistics over time),
  - `gnunet-peerinfo-gtk` (information about current connections and known peers),

[2] Continuous Integration

- gnunet-chat-gtk (chat GUI) and
- gnunet-setup (setup tool for "everything")

gnunet-fuse
: Mounting directories shared via GNUnet's file-sharing on GNU/Linux distributions

gnunet-update
: Installation and update tool

gnunet-ext
: Template for starting 'external' GNUnet projects

gnunet-java
: Java APIs for writing GNUnet services and applications

eclectic
: Code to run GNUnet nodes on testbeds for research, development, testing and evaluation

gnunet-qt
: Qt-based GNUnet GUI (is it deprecated?)

gnunet-cocoa
: cocoa-based GNUnet GUI (is it deprecated?)

gnunet-guile
: Guile bindings for GNUnet

We are also working on various supporting libraries and tools:

libextractor
: GNU libextractor (meta data extraction)

libmicrohttpd
: GNU libmicrohttpd (embedded HTTP(S) server library)

gauger
: Tool for performance regression analysis

monkey
: Tool for automated debugging of distributed systems

libmwmodem
: Library for accessing satellite connection quality reports

libgnurl
: gnURL (feature-restricted variant of cURL/libcurl)

Finally, there are various external projects (see links for a list of those that have a public website) which build on top of the GNUnet framework.

5.2 Internal dependencies

This section tries to give an overview of what processes a typical GNUnet peer running a particular application would consist of. All of the processes listed here should be automatically started by `gnunet-arm -s`. The list is given as a rough first guide to users for failure diagnostics. Ideally, end-users should never have to worry about these internal dependencies.

In terms of internal dependencies, a minimum file-sharing system consists of the following GNUnet processes (in order of dependency):

- gnunet-service-arm
- gnunet-service-resolver (required by all)
- gnunet-service-statistics (required by all)
- gnunet-service-peerinfo
- gnunet-service-transport (requires peerinfo)
- gnunet-service-core (requires transport)
- gnunet-daemon-hostlist (requires core)
- gnunet-daemon-topology (requires hostlist, peerinfo)
- gnunet-service-datastore
- gnunet-service-dht (requires core)
- gnunet-service-identity
- gnunet-service-fs (requires identity, mesh, dht, datastore, core)

A minimum VPN system consists of the following GNUnet processes (in order of dependency):

- gnunet-service-arm
- gnunet-service-resolver (required by all)
- gnunet-service-statistics (required by all)
- gnunet-service-peerinfo
- gnunet-service-transport (requires peerinfo)
- gnunet-service-core (requires transport)
- gnunet-daemon-hostlist (requires core)
- gnunet-service-dht (requires core)
- gnunet-service-mesh (requires dht, core)
- gnunet-service-dns (requires dht)
- gnunet-service-regex (requires dht)
- gnunet-service-vpn (requires regex, dns, mesh, dht)

A minimum GNS system consists of the following GNUnet processes (in order of dependency):

- gnunet-service-arm
- gnunet-service-resolver (required by all)
- gnunet-service-statistics (required by all)
- gnunet-service-peerinfo
- gnunet-service-transport (requires peerinfo)
- gnunet-service-core (requires transport)
- gnunet-daemon-hostlist (requires core)
- gnunet-service-dht (requires core)
- gnunet-service-mesh (requires dht, core)

- gnunet-service-dns (requires dht)
- gnunet-service-regex (requires dht)
- gnunet-service-vpn (requires regex, dns, mesh, dht)
- gnunet-service-identity
- gnunet-service-namestore (requires identity)
- gnunet-service-gns (requires vpn, dns, dht, namestore, identity)

5.3 Code overview

This section gives a brief overview of the GNUnet source code. Specifically, we sketch the function of each of the subdirectories in the `gnunet/src/` directory. The order given is roughly bottom-up (in terms of the layers of the system).

`util/` — libgnunetutil
: Library with general utility functions, all GNUnet binaries link against this library. Anything from memory allocation and data structures to cryptography and inter-process communication. The goal is to provide an OS-independent interface and more 'secure' or convenient implementations of commonly used primitives. The API is spread over more than a dozen headers, developers should study those closely to avoid duplicating existing functions. see Section 5.14 [libgnunetutil], page 91.

`hello/` — libgnunethello
: HELLO messages are used to describe under which addresses a peer can be reached (for example, protocol, IP, port). This library manages parsing and generating of HELLO messages.

`block/` — libgnunetblock
: The DHT and other components of GNUnet store information in units called 'blocks'. Each block has a type and the type defines a particular format and how that binary format is to be linked to a hash code (the key for the DHT and for databases). The block library is a wapper around block plugins which provide the necessary functions for each block type.

`statistics/` — statistics service
: The statistics service enables associating values (of type uint64_t) with a component name and a string. The main uses is debugging (counting events), performance tracking and user entertainment (what did my peer do today?).

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

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

`datacache/` — libgnunetdatacache
: The datacache library provides (temporary) block storage for the DHT. Existing plugins can store blocks in Sqlite, Postgres or MySQL databases. All data stored in the cache is lost when the peer is stopped or restarted (datacache uses temporary tables).

`datastore/` — datastore service
: The datastore service stores file-sharing blocks in databases for extended periods of time. In contrast to the datacache, data is not lost when peers restart. However, quota restrictions may still cause old, expired or low-priority data to be eventually discarded. Existing plugins can store blocks in Sqlite, Postgres or MySQL databases.

`template/` — service template
: Template for writing a new service. Does nothing.

`ats/` — Automatic Transport Selection
: The automatic transport selection (ATS) service is responsible for deciding which address (i.e. which transport plugin) should be used for communication with other peers, and at what bandwidth.

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

`fragmentation/` — libgnunetfragmentation
: Some transports (UDP and WLAN, mostly) have restrictions on the maximum transfer unit (MTU) for packets. The fragmentation library can be used to break larger packets into chunks of at most 1k and transmit the resulting fragments reliabily (with acknowledgement, retransmission, timeouts, etc.).

`transport/` — transport service
: The transport service is responsible for managing the basic P2P communication. It uses plugins to support P2P communication over TCP, UDP, HTTP, HTTPS and other protocols.The transport service validates peer addresses, enforces bandwidth restrictions, limits the total number of connections and enforces connectivity restrictions (i.e. friends-only).

`peerinfo-tool/` — gnunet-peerinfo
: This directory contains the gnunet-peerinfo binary which can be used to inspect the peers and HELLOs known to the peerinfo service.

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

`testing/` — libgnunettesting
: The testing library allows starting (and stopping) peers for writing testcases. It also supports automatic generation of configurations for peers ensuring that

the ports and paths are disjoint. libgnunettesting is also the foundation for the testbed service

`testbed/` — testbed service
: The testbed service is used for creating small or large scale deployments of GNUnet peers for evaluation of protocols. It facilitates peer depolyments on multiple hosts (for example, in a cluster) and establishing varous network topologies (both underlay and overlay).

`nse/` — Network Size Estimation
: The network size estimation (NSE) service implements a protocol for (securely) estimating the current size of the P2P network.

`dht/` — distributed hash table
: The distributed hash table (DHT) service provides a distributed implementation of a hash table to store blocks under hash keys in the P2P network.

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

`topology/` — topology service
: The topology service is responsible for maintaining the mesh topology. It tries to maintain connections to friends (depending on the configuration) and also tries to ensure that the peer has a decent number of active connections at all times. If necessary, new connections are added. All peers should run the topology service, otherwise they may end up not being connected to any other peer (unless some other service ensures that core establishes the required connections). The topology service also tells the transport service which connections are permitted (for friend-to-friend networking)

`fs/` — file-sharing
: The file-sharing (FS) service implements GNUnet's file-sharing application. Both anonymous file-sharing (using gap) and non-anonymous file-sharing (using dht) are supported.

`cadet/` — cadet service
: The CADET service provides a general-purpose routing abstraction to create end-to-end encrypted tunnels in mesh networks. We wrote a paper documenting key aspects of the design.

`tun/` — libgnunettun
: Library for building IPv4, IPv6 packets and creating checksums for UDP, TCP and ICMP packets. The header defines C structs for common Internet packet formats and in particular structs for interacting with TUN (virtual network) interfaces.

`mysql/` — libgnunetmysql
: Library for creating and executing prepared MySQL statements and to manage the connection to the MySQL database. Essentially a lightweight wrapper for the interaction between GNUnet components and libmysqlclient.

`dns/` Service that allows intercepting and modifying DNS requests of the local machine. Currently used for IPv4-IPv6 protocol translation (DNS-ALG) as implemented by "pt/" and for the GNUnet naming system. The service can also be configured to offer an exit service for DNS traffic.

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

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

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

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

`revocation/`
: Key revocation service, can be used to revoke the private key of an identity if it has been compromised

`namecache/`
: Cache for resolution results for the GNU name system; data is encrypted and can be shared among users, loss of the data should ideally only result in a performance degradation (persistence not required)

`namestore/`
: Database for the GNU name system with per-user private information, persistence required

`gns/` GNU name system, a GNU approach to DNS and PKI.

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

`regex/` Service for the (distributed) evaluation of regular expressions.

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

`consensus/`
: The consensus service will allow a set of peers to agree on a set of values via a distributed set union computation.

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

5.4 System Architecture

GNUnet developers like LEGOs. The blocks are indestructible, can be stacked together to construct complex buildings and it is generally easy to swap one block for a different one that has the same shape. GNUnet's architecture is based on LEGOs:

This chapter documents the GNUnet LEGO system, also known as GNUnet's system architecture.

The most common GNUnet component is a service. Services offer an API (or several, depending on what you count as "an API") which is implemented as a library. The library communicates with the main process of the service using a service-specific network protocol. The main process of the service typically doesn't fully provide everything that is needed — it has holes to be filled by APIs to other services.

A special kind of component in GNUnet are user interfaces and daemons. Like services, they have holes to be filled by APIs of other services. Unlike services, daemons do not implement their own network protocol and they have no API:

The GNUnet system provides a range of services, daemons and user interfaces, which are then combined into a layered GNUnet instance (also known as a peer).

Note that while it is generally possible to swap one service for another compatible service, there is often only one implementation. However, during development we often have a "new" version of a service in parallel with an "old" version. While the "new" version is not working, developers working on other parts of the service can continue their development by simply using the "old" service. Alternative design ideas can also be easily investigated by swapping out individual components. This is typically achieved by simply changing the name of the "BINARY" in the respective configuration section.

Key properties of GNUnet services are that they must be separate processes and that they must protect themselves by applying tight error checking against the network protocol they implement (thereby achieving a certain degree of robustness).

On the other hand, the APIs are implemented to tolerate failures of the service, isolating their host process from errors by the service. If the service process crashes, other services and daemons around it should not also fail, but instead wait for the service process to be restarted by ARM.

5.5 Subsystem stability

This section documents the current stability of the various GNUnet subsystems. Stability here describes the expected degree of compatibility with future versions of GNUnet. For each subsystem we distinguish between compatibility on the P2P network level (communication protocol between peers), the IPC level (communication between the service and the service library) and the API level (stability of the API). P2P compatibility is relevant

in terms of which applications are likely going to be able to communicate with future versions of the network. IPC communication is relevant for the implementation of language bindings that re-implement the IPC messages. Finally, API compatibility is relevant to developers that hope to be able to avoid changes to applications build on top of the APIs of the framework.

The following table summarizes our current view of the stability of the respective protocols or APIs:

Subsystem	P2P	IPC	C API
util	n/a	n/a	stable
arm	n/a	stable	stable
ats	n/a	unstable	testing
block	n/a	n/a	stable
cadet	testing	testing	testing
consensus	experimental	experimental	experimental
core	stable	stable	stable
datacache	n/a	n/a	stable
datastore	n/a	stable	stable
dht	stable	stable	stable
dns	stable	stable	stable
dv	testing	testing	n/a
exit	testing	n/a	n/a
fragmentation	stable	n/a	stable
fs	stable	stable	stable
gns	stable	stable	stable
hello	n/a	n/a	testing
hostlist	stable	stable	n/a
identity	stable	stable	n/a
multicast	experimental	experimental	experimental
mysql	stable	n/a	stable
namestore	n/a	stable	stable
nat	n/a	n/a	stable
nse	stable	stable	stable
peerinfo	n/a	stable	stable
psyc	experimental	experimental	experimental
pt	n/a	n/a	n/a
regex	stable	stable	stable
revocation	stable	stable	stable
social	experimental	experimental	experimental
statistics	n/a	stable	stable
testbed	n/a	testing	testing
testing	n/a	n/a	testing
topology	n/a	n/a	n/a
transport	stable	stable	stable
tun	n/a	n/a	stable
vpn	testing	n/a	n/a

Here is a rough explanation of the values:

'stable' No incompatible changes are planned at this time; for IPC/APIs, if there are incompatible changes, they will be minor and might only require minimal changes to existing code; for P2P, changes will be avoided if at all possible for the 0.10.x-series

'testing' No incompatible changes are planned at this time, but the code is still known to be in flux; so while we have no concrete plans, our expectation is that there will still be minor modifications; for P2P, changes will likely be extensions that should not break existing code

'unstable'
Changes are planned and will happen; however, they will not be totally radical and the result should still resemble what is there now; nevertheless, anticipated changes will break protocol/API compatibility

'experimental'
Changes are planned and the result may look nothing like what the API/protocol looks like today

'unknown' Someone should think about where this subsystem headed

'n/a' This subsystem does not have an API/IPC-protocol/P2P-protocol

5.6 Naming conventions and coding style guide

Here you can find some rules to help you write code for GNUnet.

5.6.1 Naming conventions

5.6.1.1 include files

- _lib: library without need for a process
- _service: library that needs a service process
- _plugin: plugin definition
- _protocol: structs used in network protocol
- exceptions:
  - gnunet_config.h — generated
  - platform.h — first included
  - plibc.h — external library
  - gnunet_common.h — fundamental routines
  - gnunet_directories.h — generated
  - gettext.h — external library

5.6.1.2 binaries

- gnunet-service-xxx: service process (has listen socket)
- gnunet-daemon-xxx: daemon process (no listen socket)
- gnunet-helper-xxx[-yyy]: SUID helper for module xxx

- gnunet-yyy: command-line tool for end-users
- libgnunet_plugin_xxx_yyy.so: plugin for API xxx
- libgnunetxxx.so: library for API xxx

5.6.1.3 logging

- services and daemons use their directory name in `GNUNET_log_setup` (i.e. 'core') and log using plain 'GNUNET_log'.
- command-line tools use their full name in `GNUNET_log_setup` (i.e. 'gnunet-publish') and log using plain 'GNUNET_log'.
- service access libraries log using '`GNUNET_log_from`' and use '`DIRNAME-api`' for the component (i.e. 'core-api')
- pure libraries (without associated service) use '`GNUNET_log_from`' with the component set to their library name (without lib or '`.so`'), which should also be their directory name (i.e. '`nat`')
- plugins should use '`GNUNET_log_from`' with the directory name and the plugin name combined to produce the component name (i.e. 'transport-tcp').
- logging should be unified per-file by defining a `LOG` macro with the appropriate arguments, along these lines:

```
#define LOG(kind,...)
GNUNET_log_from (kind, "example-api",__VA_ARGS__)
```

5.6.1.4 configuration

- paths (that are substituted in all filenames) are in PATHS (have as few as possible)
- all options for a particular module (`src/MODULE`) are under `[MODULE]`
- options for a plugin of a module are under `[MODULE-PLUGINNAME]`

5.6.1.5 exported symbols

- must start with `GNUNET_modulename_` and be defined in `modulename.c`
- exceptions: those defined in `gnunet_common.h`

5.6.1.6 private (library-internal) symbols (including structs and macros)

- must NOT start with any prefix
- must not be exported in a way that linkers could use them or other libraries might see them via headers; they must be either declared/defined in C source files or in headers that are in the respective directory under `src/modulename/` and NEVER be declared in `src/include/`.

5.6.1.7 testcases

- must be called `test_module-under-test_case-description.c`
- `"`case-description`"` maybe omitted if there is only one test

5.6.1.8 performance tests

- must be called `perf_module-under-test_case-description.c`
- "case-description" maybe omitted if there is only one performance test
- Must only be run if `HAVE_BENCHMARKS` is satisfied

5.6.1.9 src/ directories

- gnunet-NAME: end-user applications (i.e., gnunet-search, gnunet-arm)
- gnunet-service-NAME: service processes with accessor library (i.e., gnunet-service-arm)
- libgnunetNAME: accessor library (_service.h-header) or standalone library (_lib.h-header)
- gnunet-daemon-NAME: daemon process without accessor library (i.e., gnunet-daemon-hostlist) and no GNUnet management port
- libgnunet_plugin_DIR_NAME: loadable plugins (i.e., libgnunet_plugin_transport_tcp)

5.6.2 Coding style

- We follow the GNU Coding Standards (see *The GNU Coding Standards*);
- Indentation is done with spaces, two per level, no tabs;
- C99 struct initialization is fine;
- declare only one variable per line, for example:

  instead of

  ```
  int i,j;
  ```

  write:

  ```
  int i;
  int j;
  ```

  This helps keep diffs small and forces developers to think precisely about the type of every variable. Note that `char *` is different from `const char*` and `int` is different from `unsigned int` or `uint32_t`. Each variable type should be chosen with care.
- While `goto` should generally be avoided, having a `goto` to the end of a function to a block of clean up statements (free, close, etc.) can be acceptable.
- Conditions should be written with constants on the left (to avoid accidental assignment) and with the `true` target being either the `error` case or the significantly simpler continuation. For example:

  ```
  if (0 != stat ("filename," &sbuf)) {
    error();
   }
   else {
     /* handle normal case here */
   }
  ```

  instead of

  ```
  if (stat ("filename," &sbuf) == 0) {
    /* handle normal case here */
   } else {
  ```

```
  error();
 }
```

If possible, the error clause should be terminated with a `return` (or `goto` to some cleanup routine) and in this case, the `else` clause should be omitted:

```
if (0 != stat ("filename," &sbuf)) {
  error();
  return;
 }
/* handle normal case here */
```

This serves to avoid deep nesting. The 'constants on the left' rule applies to all constants (including. `GNUNET_SCHEDULER_NO_TASK`), NULL, and enums). With the two above rules (constants on left, errors in 'true' branch), there is only one way to write most branches correctly.

- Combined assignments and tests are allowed if they do not hinder code clarity. For example, one can write:

```
if (NULL == (value = lookup_function())) {
  error();
  return;
 }
```

- Use `break` and `continue` wherever possible to avoid deep(er) nesting. Thus, we would write:

```
next = head;
while (NULL != (pos = next)) {
  next = pos->next;
  if (! should_free (pos))
    continue;
  GNUNET_CONTAINER_DLL_remove (head, tail, pos);
  GNUNET_free (pos);
 }
```

instead of

```
next = head; while (NULL != (pos = next)) {
  next = pos->next;
  if (should_free (pos)) {
    /* unnecessary nesting! */
    GNUNET_CONTAINER_DLL_remove (head, tail, pos);
    GNUNET_free (pos);
   }
  }
```

- We primarily use `for` and `while` loops. A `while` loop is used if the method for advancing in the loop is not a straightforward increment operation. In particular, we use:

```
next = head;
while (NULL != (pos = next))
{
  next = pos->next;
  if (! should_free (pos))
```

```
    continue;
  GNUNET_CONTAINER_DLL_remove (head, tail, pos);
  GNUNET_free (pos);
}
```

to free entries in a list (as the iteration changes the structure of the list due to the free; the equivalent `for` loop does no longer follow the simple `for` paradigm of `for(INIT;TEST;INC)`). However, for loops that do follow the simple `for` paradigm we do use `for`, even if it involves linked lists:

```
/* simple iteration over a linked list */
for (pos = head;
     NULL != pos;
     pos = pos->next)
{
   use (pos);
}
```

- The first argument to all higher-order functions in GNUnet must be declared to be of type `void *` and is reserved for a closure. We do not use inner functions, as trampolines would conflict with setups that use non-executable stacks. The first statement in a higher-order function, which unusually should be part of the variable declarations, should assign the `cls` argument to the precise expected type. For example:

```
int callback (void *cls, char *args) {
  struct Foo *foo = cls;
  int other_variables;

   /* rest of function */
}
```

- It is good practice to write complex `if` expressions instead of using deeply nested `if` statements. However, except for addition and multiplication, all operators should use parens. This is fine:

```
if ( (1 == foo) || ((0 == bar) && (x != y)) )
  return x;
```

However, this is not:

```
if (1 == foo)
  return x;
if (0 == bar && x != y)
  return x;
```

Note that splitting the `if` statement above is debateable as the `return x` is a very trivial statement. However, once the logic after the branch becomes more complicated (and is still identical), the "or" formulation should be used for sure.

- There should be two empty lines between the end of the function and the comments describing the following function. There should be a single empty line after the initial variable declarations of a function. If a function has no local variables, there should be no initial empty line. If a long function consists of several complex steps, those steps might be separated by an empty line (possibly followed by a comment describing the following step). The code should not contain empty lines in arbitrary places; if in

doubt, it is likely better to NOT have an empty line (this way, more code will fit on the screen).

5.7 Build-system

If you have code that is likely not to compile or build rules you might want to not trigger for most developers, use `if HAVE_EXPERIMENTAL` in your `Makefile.am`. Then it is OK to (temporarily) add non-compiling (or known-to-not-port) code.

If you want to compile all testcases but NOT run them, run configure with the `--enable-test-suppression` option.

If you want to run all testcases, including those that take a while, run configure with the `--enable-expensive-testcases` option.

If you want to compile and run benchmarks, run configure with the `--enable-benchmarks` option.

If you want to obtain code coverage results, run configure with the `--enable-coverage` option and run the `coverage.sh` script in the `contrib/` directory.

5.8 Developing extensions for GNUnet using the gnunet-ext template

For developers who want to write extensions for GNUnet we provide the gnunet-ext template to provide an easy to use skeleton.

gnunet-ext contains the build environment and template files for the development of GNUnet services, command line tools, APIs and tests.

First of all you have to obtain gnunet-ext from git:

```
git clone https://gnunet.org/git/gnunet-ext.git
```

The next step is to bootstrap and configure it. For configure you have to provide the path containing GNUnet with `--with-gnunet=/path/to/gnunet` and the prefix where you want the install the extension using `--prefix=/path/to/install`:

```
./bootstrap
./configure --prefix=/path/to/install --with-gnunet=/path/to/gnunet
```

When your GNUnet installation is not included in the default linker search path, you have to add `/path/to/gnunet` to the file `/etc/ld.so.conf` and run `ldconfig` or your add it to the environmental variable `LD_LIBRARY_PATH` by using

```
export LD_LIBRARY_PATH=/path/to/gnunet/lib
```

5.9 Writing testcases

Ideally, any non-trivial GNUnet code should be covered by automated testcases. Testcases should reside in the same place as the code that is being tested. The name of source files implementing tests should begin with `test_` followed by the name of the file that contains the code that is being tested.

Testcases in GNUnet should be integrated with the autotools build system. This way, developers and anyone building binary packages will be able to run all testcases simply by running `make check`. The final testcases shipped with the distribution should output

at most some brief progress information and not display debug messages by default. The success or failure of a testcase must be indicated by returning zero (success) or non-zero (failure) from the main method of the testcase. The integration with the autotools is relatively straightforward and only requires modifications to the `Makefile.am` in the directory containing the testcase. For a testcase testing the code in `foo.c` the `Makefile.am` would contain the following lines:

```
check_PROGRAMS = test_foo
TESTS = $(check_PROGRAMS)
test_foo_SOURCES = test_foo.c
test_foo_LDADD = $(top_builddir)/src/util/libgnunetutil.la
```

Naturally, other libraries used by the testcase may be specified in the `LDADD` directive as necessary.

Often testcases depend on additional input files, such as a configuration file. These support files have to be listed using the `EXTRA_DIST` directive in order to ensure that they are included in the distribution.

Example:

```
EXTRA_DIST = test_foo_data.conf
```

Executing `make check` will run all testcases in the current directory and all subdirectories. Testcases can be compiled individually by running `make test_foo` and then invoked directly using `./test_foo`. Note that due to the use of plugins in GNUnet, it is typically necessary to run `make install` before running any testcases. Thus the canonical command `make check install` has to be changed to `make install check` for GNUnet.

5.10 Building GNUnet and its dependencies

In the following section we will outline how to build GNUnet and some of its dependencies. We will assume a fair amount of knowledge for building applications under UNIX-like systems. Furthermore we assume that the build environment is sane and that you are aware of any implications actions in this process could have. Instructions here can be seen as notes for developers (an extension to the 'HACKING' section in README) as well as package mantainers. **Users should rely on the available binary packages.** We will use Debian as an example Operating System environment. Substitute accordingly with your own Ooperating System environment.

For the full list of depenendencies, consult the appropriate, up-to-date section in the `README` file.

First, we need to build or install (depending on your OS) the following packages. If you build them from source, build them in this exact order:

```
libgpgerror, libgcrypt, libnettle, libunbound, GnuTLS (with libunbound
support)
```

After we have build and installed those packages, we continue with packages closer to GNUnet in this step: libgnurl (our libcurl fork), GNU libmicrohttpd, and GNU libextractor. Again, if your package manager provides one of these packages, use the packages provided from it unless you have good reasons (package version too old, conflicts, etc). We advise against compiling widely used packages such as GnuTLS yourself if your OS provides a variant already unless you take care of maintenance of the packages then.

In the optimistic case, this command will give you all the dependencies:

```
sudo apt-get install libgnurl libmicrohttpd libextractor
```

From experience we know that at the very least libgnurl is not available in some environments. You could substitute libgnurl with libcurl, but we recommend to install libgnurl, as it gives you a predefined libcurl with the small set GNUnet requires. In the past namespaces of libcurl and libgnurl were shared, which caused problems when you wanted to integrate both of them in one Operating System. This has been resolved, and they can be installed side by side now.

GNUnet and some of its function depend on a limited subset of cURL/libcurl. Rather than trying to enforce a certain configuration on the world, we opted to maintain a microfork of it that ensures we can link against the right set of features. We called this specialized set of libcurl "libgnurl". It is fully ABI compatible with libcurl and currently used by GNUnet and some of its dependencies.

We download libgnurl and its digital signature from the GNU fileserver, assuming `TMPDIR` exists[3]

```
cd \$TMPDIR
wget https://ftp.gnu.org/gnu/gnunet/gnurl-7.60.0.tar.Z
wget https://ftp.gnu.org/gnu/gnunet/gnurl-7.60.0.tar.Z.sig
```

Next, verify the digital signature of the file:

```
gpg --verify gnurl-7.60.0.tar.Z.sig
```

If gpg fails, you might try with `gpg2` on your OS. If the error states that "the key can not be found" or it is unknown, you have to retrieve the key (A88C8ADD129828D7EAC02E52E22F9BBFEE348588) from a keyserver first:

```
gpg --keyserver pgp.mit.edu --recv-keys A88C8ADD129828D7EAC02E52E22F9BBFEE348588
```

and rerun the verification command.

libgnurl will require the following packages to be present at runtime: gnutls (with DANE support / libunbound), libidn, zlib and at compile time: libtool, groff, perl, pkg-config, and python 2.7.

Once you have verified that all the required packages are present on your system, we can proceed to compile libgnurl:

```
tar -xvf gnurl-7.60.0.tar.Z
cd gnurl-7.60.0
sh configure --disable-ntlm-wb
make
make -C tests test
sudo make install
```

After you've compiled and installed libgnurl, we can proceed to building GNUnet.

First, in addition to the GNUnet sources you might require downloading the latest version of various dependencies, depending on how recent the software versions in your distribution of GNU/Linux are. Most distributions do not include sufficiently recent versions of

[3] It might be `/tmp`, `TMPDIR`, `TMP` or any other location. For consistency we assume `TMPDIR` points to `/tmp` for the remainder of this section.

these dependencies. Thus, a typically installation on a "modern" GNU/Linux distribution requires you to install the following dependencies (ideally in this order):

- libgpgerror and libgcrypt
- libnettle and libunbound (possibly from distribution), GnuTLS
- libgnurl (read the README)
- GNU libmicrohttpd
- GNU libextractor

Make sure to first install the various mandatory and optional dependencies including development headers from your distribution.

Other dependencies that you should strongly consider to install is a database (MySQL, sqlite or Postgres). The following instructions will assume that you installed at least sqlite. For most distributions you should be able to find pre-build packages for the database. Again, make sure to install the client libraries **and** the respective development headers (if they are packaged separately) as well.

You can find specific, detailed instructions for installing of the dependencies (and possibly the rest of the GNUnet installation) in the platform-specific descriptions, which can be found in the Index. Please consult them now. If your distribution is not listed, please study the build instructions for Debian stable, carefully as you try to install the dependencies for your own distribution. Contributing additional instructions for further platforms is always appreciated. Please take in mind that operating system development tends to move at a rather fast speed. Due to this you should be aware that some of the instructions could be outdated by the time you are reading this. If you find a mistake, please tell us about it (or even better: send a patch to the documentation to fix it!).

Before proceeding further, please double-check the dependency list. Note that in addition to satisfying the dependencies, you might have to make sure that development headers for the various libraries are also installed. There maybe files for other distributions, or you might be able to find equivalent packages for your distribution.

While it is possible to build and install GNUnet without having root access, we will assume that you have full control over your system in these instructions. First, you should create a system user *gnunet* and an additional group *gnunetdns*. On the GNU/Linux distributions Debian and Ubuntu, type:

```
sudo adduser --system --home /var/lib/gnunet --group \
--disabled-password gnunet
sudo addgroup --system gnunetdns
```

On other Unixes and GNU systems, this should have the same effect:

```
sudo useradd --system --groups gnunet --home-dir /var/lib/gnunet
sudo addgroup --system gnunetdns
```

Now compile and install GNUnet using:

```
tar xvf gnunet-0.11.0pre66.tar.gz
cd gnunet-0.11.0pre66
./configure --with-sudo=sudo --with-nssdir=/lib
make
sudo make install
```

If you want to be able to enable DEBUG-level log messages, add `--enable-logging=verbose` to the end of the `./configure` command. `DEBUG`-level log messages are in English only and should only be useful for developers (or for filing really detailed bug reports).

Next, edit the file `/etc/gnunet.conf` to contain the following:

```
[arm]
SYSTEM_ONLY = YES
USER_ONLY = NO
```

You may need to update your `ld.so` cache to include files installed in `/usr/local/lib`:

```
# ldconfig
```

Then, switch from user `root` to user `gnunet` to start the peer:

```
# su -s /bin/sh - gnunet
$ gnunet-arm -c /etc/gnunet.conf -s
```

You may also want to add the last line in the gnunet user's `crontab` prefixed with `@reboot` so that it is executed whenever the system is booted:

```
@reboot /usr/local/bin/gnunet-arm -c /etc/gnunet.conf -s
```

This will only start the system-wide GNUnet services. Type `exit` to get back your root shell. Now, you need to configure the per-user part. For each user that should get access to GNUnet on the system, run (replace alice with your username):

```
sudo adduser alice gnunet
```

to allow them to access the system-wide GNUnet services. Then, each user should create a configuration file `~/.config/gnunet.conf` with the lines:

```
[arm]
SYSTEM_ONLY = NO
USER_ONLY = YES
DEFAULTSERVICES = gns
```

and start the per-user services using

```
$ gnunet-arm -c ~/.config/gnunet.conf -s
```

Again, adding a `crontab` entry to autostart the peer is advised:

```
@reboot /usr/local/bin/gnunet-arm -c $HOME/.config/gnunet.conf -s
```

Note that some GNUnet services (such as SOCKS5 proxies) may need a system-wide TCP port for each user. For those services, systems with more than one user may require each user to specify a different port number in their personal configuration file.

Finally, the user should perform the basic initial setup for the GNU Name System (GNS) certificate authority. This is done by running:

```
$ gnunet-gns-proxy-setup-ca
```

The first generates the default zones, wheras the second setups the GNS Certificate Authority with the user's browser. Now, to activate GNS in the normal DNS resolution process, you need to edit your `/etc/nsswitch.conf` where you should find a line like this:

```
hosts: files mdns4_minimal [NOTFOUND=return] dns mdns4
```

The exact details may differ a bit, which is fine. Add the text "*gns [NOTFOUND=return]*" after "*files*". Keep in mind that we included a backslash ("\") here just for markup reasons. You should write the text below on **one line** and **without** the "\":

```
hosts: files gns [NOTFOUND=return] mdns4_minimal \
[NOTFOUND=return] dns mdns4
```

You might want to make sure that `/lib/libnss_gns.so.2` exists on your system, it should have been created during the installation.

5.11 TESTING library

The TESTING library is used for writing testcases which involve starting a single or multiple peers. While peers can also be started by testcases using the ARM subsystem, using TESTING library provides an elegant way to do this. The configurations of the peers are auto-generated from a given template to have non-conflicting port numbers ensuring that peers' services do not run into bind errors. This is achieved by testing ports' availability by binding a listening socket to them before allocating them to services in the generated configurations.

An another advantage while using TESTING is that it shortens the testcase startup time as the hostkeys for peers are copied from a pre-computed set of hostkeys instead of generating them at peer startup which may take a considerable amount of time when starting multiple peers or on an embedded processor.

TESTING also allows for certain services to be shared among peers. This feature is invaluable when testing with multiple peers as it helps to reduce the number of services run per each peer and hence the total number of processes run per testcase.

TESTING library only handles creating, starting and stopping peers. Features useful for testcases such as connecting peers in a topology are not available in TESTING but are available in the TESTBED subsystem. Furthermore, TESTING only creates peers on the localhost, however by using TESTBED testcases can benefit from creating peers across multiple hosts.

5.11.1 API

TESTING abstracts a group of peers as a TESTING system. All peers in a system have common hostname and no two services of these peers have a same port or a UNIX domain socket path.

TESTING system can be created with the function `GNUNET_TESTING_system_create()` which returns a handle to the system. This function takes a directory path which is used for generating the configurations of peers, an IP address from which connections to the peers' services should be allowed, the hostname to be used in peers' configuration, and an array of shared service specifications of type `struct GNUNET_TESTING_SharedService`.

The shared service specification must specify the name of the service to share, the configuration pertaining to that shared service and the maximum number of peers that are allowed to share a single instance of the shared service.

TESTING system created with `GNUNET_TESTING_system_create()` chooses ports from the default range `12000 - 56000` while auto-generating configurations for peers. This range can be customised with the function `GNUNET_TESTING_system_create_with_portrange()`.

This function is similar to `GNUNET_TESTING_system_create()` except that it take 2 additional parameters — the start and end of the port range to use.

A TESTING system is destroyed with the funciton `GNUNET_TESTING_system_destory()`. This function takes the handle of the system and a flag to remove the files created in the directory used to generate configurations.

A peer is created with the function `GNUNET_TESTING_peer_configure()`. This functions takes the system handle, a configuration template from which the configuration for the peer is auto-generated and the index from where the hostkey for the peer has to be copied from. When successfull, this function returs a handle to the peer which can be used to start and stop it and to obtain the identity of the peer. If unsuccessful, a NULL pointer is returned with an error message. This function handles the generated configuration to have non-conflicting ports and paths.

Peers can be started and stopped by calling the functions `GNUNET_TESTING_peer_start()` and `GNUNET_TESTING_peer_stop()` respectively. A peer can be destroyed by calling the function `GNUNET_TESTING_peer_destroy`. When a peer is destroyed, the ports and paths in allocated in its configuration are reclaimed for usage in new peers.

5.11.2 Finer control over peer stop

Using `GNUNET_TESTING_peer_stop()` is normally fine for testcases. However, calling this function for each peer is inefficient when trying to shutdown multiple peers as this function sends the termination signal to the given peer process and waits for it to terminate. It would be faster in this case to send the termination signals to the peers first and then wait on them. This is accomplished by the functions `GNUNET_TESTING_peer_kill()` which sends a termination signal to the peer, and the function `GNUNET_TESTING_peer_wait()` which waits on the peer.

Further finer control can be achieved by choosing to stop a peer asynchronously with the function `GNUNET_TESTING_peer_stop_async()`. This function takes a callback parameter and a closure for it in addition to the handle to the peer to stop. The callback function is called with the given closure when the peer is stopped. Using this function eliminates blocking while waiting for the peer to terminate.

An asynchronous peer stop can be cancelled by calling the function `GNUNET_TESTING_peer_stop_async_cancel()`. Note that calling this function does not prevent the peer from terminating if the termination signal has already been sent to it. It does, however, cancels the callback to be called when the peer is stopped.

5.11.3 Helper functions

Most of the testcases can benefit from an abstraction which configures a peer and starts it. This is provided by the function `GNUNET_TESTING_peer_run()`. This function takes the testing directory pathname, a configuration template, a callback and its closure. This function creates a peer in the given testing directory by using the configuration template, starts the peer and calls the given callback with the given closure.

The function `GNUNET_TESTING_peer_run()` starts the ARM service of the peer which starts the rest of the configured services. A similar function `GNUNET_TESTING_service_run` can be used to just start a single service of a peer. In this case, the peer's ARM service is not started; instead, only the given service is run.

5.11.4 Testing with multiple processes

When testing GNUnet, the splitting of the code into a services and clients often complicates testing. The solution to this is to have the testcase fork `gnunet-service-arm`, ask it to start the required server and daemon processes and then execute appropriate client actions (to test the client APIs or the core module or both). If necessary, multiple ARM services can be forked using different ports (!) to simulate a network. However, most of the time only one ARM process is needed. Note that on exit, the testcase should shutdown ARM with a `TERM` signal (to give it the chance to cleanly stop its child processes).

The following code illustrates spawning and killing an ARM process from a testcase:

```
static void run (void *cls,
                 char *const *args,
                 const char *cfgfile,
                 const struct GNUNET_CONFIGURATION_Handle *cfg) {
  struct GNUNET_OS_Process *arm_pid;
  arm_pid = GNUNET_OS_start_process (NULL,
                                     NULL,
                                     "gnunet-service-arm",
                                     "gnunet-service-arm",
                                     "-c",
                                     cfgname,
                                     NULL);
  /* do real test work here */
  if (0 != GNUNET_OS_process_kill (arm_pid, SIGTERM))
    GNUNET_log_strerror
      (GNUNET_ERROR_TYPE_WARNING, "kill");
  GNUNET_assert (GNUNET_OK == GNUNET_OS_process_wait (arm_pid));
  GNUNET_OS_process_close (arm_pid); }

GNUNET_PROGRAM_run (argc, argv,
                    "NAME-OF-TEST",
                    "nohelp",
                    options,
                    &run,
                    cls);
```

An alternative way that works well to test plugins is to implement a mock-version of the environment that the plugin expects and then to simply load the plugin directly.

5.12 Performance regression analysis with Gauger

To help avoid performance regressions, GNUnet uses Gauger. Gauger is a simple logging tool that allows remote hosts to send performance data to a central server, where this data can be analyzed and visualized. Gauger shows graphs of the repository revisions and the performace data recorded for each revision, so sudden performance peaks or drops can be identified and linked to a specific revision number.

In the case of GNUnet, the buildbots log the performance data obtained during the tests after each build. The data can be accesed on GNUnet's Gauger page.

The menu on the left allows to select either the results of just one build bot (under "Hosts") or review the data from all hosts for a given test result (under "Metrics"). In case of very different absolute value of the results, for instance arm vs. amd64 machines, the option "Normalize" on a metric view can help to get an idea about the performance evolution across all hosts.

Using Gauger in GNUnet and having the performance of a module tracked over time is very easy. First of course, the testcase must generate some consistent metric, which makes sense to have logged. Highly volatile or random dependant metrics probably are not ideal candidates for meaningful regression detection.

To start logging any value, just include `gauger.h` in your testcase code. Then, use the macro `GAUGER()` to make the Buildbots log whatever value is of interest for you to `gnunet.org`'s Gauger server. No setup is necessary as most Buildbots have already everything in place and new metrics are created on demand. To delete a metric, you need to contact a member of the GNUnet development team (a file will need to be removed manually from the respective directory).

The code in the test should look like this:

```
[other includes]
#include <gauger.h>

int main (int argc, char *argv[]) {

  [run test, generate data]
    GAUGER("YOUR_MODULE",
           "METRIC_NAME",
           (float)value,
           "UNIT"); }
```

Where:

YOUR_MODULE is a category in the gauger page and should be
the name of the module or subsystem like "Core" or "DHT"

METRIC is
the name of the metric being collected and should be concise and descriptive, like "PUT operations in sqlite-datastore".

value is the value
of the metric that is logged for this run.

UNIT is the unit in
which the value is measured, for instance "kb/s" or "kb of RAM/node".

If you wish to use Gauger for your own project, you can grab a copy of the latest stable release or check out Gauger's Subversion repository.

5.13 TESTBED Subsystem

The TESTBED subsystem facilitates testing and measuring of multi-peer deployments on a single host or over multiple hosts.

The architecture of the testbed module is divided into the following:

- Testbed API: An API which is used by the testing driver programs. It provides with functions for creating, destroying, starting, stopping peers, etc.
- Testbed service (controller): A service which is started through the Testbed API. This service handles operations to create, destroy, start, stop peers, connect them, modify their configurations.
- Testbed helper: When a controller has to be started on a host, the testbed API starts the testbed helper on that host which in turn starts the controller. The testbed helper receives a configuration for the controller through its stdin and changes it to ensure the controller doesn't run into any port conflict on that host.

The testbed service (controller) is different from the other GNUnet services in that it is not started by ARM and is not supposed to be run as a daemon. It is started by the testbed API through a testbed helper. In a typical scenario involving multiple hosts, a controller is started on each host. Controllers take up the actual task of creating peers, starting and stopping them on the hosts they run.

While running deployments on a single localhost the testbed API starts the testbed helper directly as a child process. When running deployments on remote hosts the testbed API starts Testbed Helpers on each remote host through remote shell. By default testbed API uses SSH as a remote shell. This can be changed by setting the environmental variable GNUNET_TESTBED_RSH_CMD to the required remote shell program. This variable can also contain parameters which are to be passed to the remote shell program. For e.g:

```
export GNUNET_TESTBED_RSH_CMD="ssh -o BatchMode=yes \
-o NoHostAuthenticationForLocalhost=yes %h"
```

Substitutions are allowed in the command string above, this allows for substitutions through placemarks which begin with a '%'. At present the following substitutions are supported

- %h: hostname
- %u: username
- %p: port

Note that the substitution placemark is replaced only when the corresponding field is available and only once. Specifying

```
%u@%h
```

doesn't work either. If you want to user username substitutions for `SSH`, use the argument `-l` before the username substitution.

For example:

```
ssh -l %u -p %p %h
```

The testbed API and the helper communicate through the helpers stdin and stdout. As the helper is started through a remote shell on remote hosts any output messages from the remote shell interfere with the communication and results in a failure while starting the helper. For this reason, it is suggested to use flags to make the remote shells produce no output messages and to have password-less logins. The default remote shell, SSH, the default options are:

```
-o BatchMode=yes -o NoHostBasedAuthenticationForLocalhost=yes"
```

Password-less logins should be ensured by using SSH keys.

Since the testbed API executes the remote shell as a non-interactive shell, certain scripts like .bashrc, .profiler may not be executed. If this is the case testbed API can be forced to execute an interactive shell by setting up the environmental variable `GNUNET_TESTBED_RSH_CMD_SUFFIX` to a shell program.

An example could be:

```
export GNUNET_TESTBED_RSH_CMD_SUFFIX="sh -lc"
```

The testbed API will then execute the remote shell program as:

```
$GNUNET_TESTBED_RSH_CMD -p $port $dest $GNUNET_TESTBED_RSH_CMD_SUFFIX \
gnunet-helper-testbed
```

On some systems, problems may arise while starting testbed helpers if GNUnet is installed into a custom location since the helper may not be found in the standard path. This can be addressed by setting the variable '`HELPER_BINARY_PATH`' to the path of the testbed helper. Testbed API will then use this path to start helper binaries both locally and remotely.

Testbed API can accessed by including the `gnunet_testbed_service.h` file and linking with `-lgnunettestbed`.

5.13.1 Supported Topologies

While testing multi-peer deployments, it is often needed that the peers are connected in some topology. This requirement is addressed by the function `GNUNET_TESTBED_overlay_connect()` which connects any given two peers in the testbed.

The API also provides a helper function `GNUNET_TESTBED_overlay_configure_topology()` to connect a given set of peers in any of the following supported topologies:

- `GNUNET_TESTBED_TOPOLOGY_CLIQUE`: All peers are connected with each other
- `GNUNET_TESTBED_TOPOLOGY_LINE`: Peers are connected to form a line
- `GNUNET_TESTBED_TOPOLOGY_RING`: Peers are connected to form a ring topology
- `GNUNET_TESTBED_TOPOLOGY_2D_TORUS`: Peers are connected to form a 2 dimensional torus topology. The number of peers may not be a perfect square, in that case the resulting torus may not have the uniform poloidal and toroidal lengths
- `GNUNET_TESTBED_TOPOLOGY_ERDOS_RENYI`: Topology is generated to form a random graph. The number of links to be present should be given
- `GNUNET_TESTBED_TOPOLOGY_SMALL_WORLD`: Peers are connected to form a 2D Torus with some random links among them. The number of random links are to be given
- `GNUNET_TESTBED_TOPOLOGY_SMALL_WORLD_RING`: Peers are connected to form a ring with some random links among them. The number of random links are to be given
- `GNUNET_TESTBED_TOPOLOGY_SCALE_FREE`: Connects peers in a topology where peer connectivity follows power law - new peers are connected with high probabililty to well connected peers.[4]

[4] See Emergence of Scaling in Random Networks. Science 286, 509-512, 1999 (pdf (`https://gnunet.org/git/bibliography.git/plain/docs/emergence_of_scaling_in_random_networks__barabasi_albert_science_286__1999.pdf`))

- `GNUNET_TESTBED_TOPOLOGY_FROM_FILE`: The topology information is loaded from a file. The path to the file has to be given. See Section 5.13.3 [Topology file format], page 87, for the format of this file.
- `GNUNET_TESTBED_TOPOLOGY_NONE`: No topology

The above supported topologies can be specified respectively by setting the variable `OVERLAY_TOPOLOGY` to the following values in the configuration passed to Testbed API functions `GNUNET_TESTBED_test_run()` and `GNUNET_TESTBED_run()`:

- `CLIQUE`
- `RING`
- `LINE`
- `2D_TORUS`
- `RANDOM`
- `SMALL_WORLD`
- `SMALL_WORLD_RING`
- `SCALE_FREE`
- `FROM_FILE`
- `NONE`

Topologies `RANDOM`, `SMALL_WORLD` and `SMALL_WORLD_RING` require the option `OVERLAY_RANDOM_LINKS` to be set to the number of random links to be generated in the configuration. The option will be ignored for the rest of the topologies.

Topology `SCALE_FREE` requires the options `SCALE_FREE_TOPOLOGY_CAP` to be set to the maximum number of peers which can connect to a peer and `SCALE_FREE_TOPOLOGY_M` to be set to how many peers a peer should be atleast connected to.

Similarly, the topology `FROM_FILE` requires the option `OVERLAY_TOPOLOGY_FILE` to contain the path of the file containing the topology information. This option is ignored for the rest of the topologies. See Section 5.13.3 [Topology file format], page 87, for the format of this file.

5.13.2 Hosts file format

The testbed API offers the function `GNUNET_TESTBED_hosts_load_from_file()` to load from a given file details about the hosts which testbed can use for deploying peers. This function is useful to keep the data about hosts separate instead of hard coding them in code.

Another helper function from testbed API, `GNUNET_TESTBED_run()` also takes a hosts file name as its parameter. It uses the above function to populate the hosts data structures and start controllers to deploy peers.

These functions require the hosts file to be of the following format:

- Each line is interpreted to have details about a host
- Host details should include the username to use for logging into the host, the hostname of the host and the port number to use for the remote shell program. All thee values should be given.

- These details should be given in the following format:

```
<username>@<hostname>:<port>
```

Note that having canonical hostnames may cause problems while resolving the IP addresses (See this bug). Hence it is advised to provide the hosts' IP numerical addresses as hostnames whenever possible.

5.13.3 Topology file format

A topology file describes how peers are to be connected. It should adhere to the following format for testbed to parse it correctly.

Each line should begin with the target peer id. This should be followed by a colon(':') and origin peer ids seperated by '|'. All spaces except for newline characters are ignored. The API will then try to connect each origin peer to the target peer.

For example, the following file will result in 5 overlay connections: [2->1], [3->1],[4->3], [0->3], [2->0] `1:2|3 3:4| 0 0: 2`

5.13.4 Testbed Barriers

The testbed subsystem's barriers API facilitates coordination among the peers run by the testbed and the experiment driver. The concept is similar to the barrier synchronisation mechanism found in parallel programming or multi-threading paradigms - a peer waits at a barrier upon reaching it until the barrier is reached by a predefined number of peers. This predefined number of peers required to cross a barrier is also called quorum. We say a peer has reached a barrier if the peer is waiting for the barrier to be crossed. Similarly a barrier is said to be reached if the required quorum of peers reach the barrier. A barrier which is reached is deemed as crossed after all the peers waiting on it are notified.

The barriers API provides the following functions:

- `GNUNET_TESTBED_barrier_init():` function to initialse a barrier in the experiment
- `GNUNET_TESTBED_barrier_cancel():` function to cancel a barrier which has been initialised before
- `GNUNET_TESTBED_barrier_wait():` function to signal barrier service that the caller has reached a barrier and is waiting for it to be crossed
- `GNUNET_TESTBED_barrier_wait_cancel():` function to stop waiting for a barrier to be crossed

Among the above functions, the first two, namely `GNUNET_TESTBED_barrier_init()` and `GNUNET_TESTBED_barrier_cancel()` are used by experiment drivers. All barriers should be initialised by the experiment driver by calling `GNUNET_TESTBED_barrier_init()`. This function takes a name to identify the barrier, the quorum required for the barrier to be crossed and a notification callback for notifying the experiment driver when the barrier is crossed. `GNUNET_TESTBED_barrier_cancel()` cancels an initialised barrier and frees the resources allocated for it. This function can be called upon a initialised barrier before it is crossed.

The remaining two functions `GNUNET_TESTBED_barrier_wait()` and `GNUNET_TESTBED_barrier_wait_cancel()` are used in the peer's processes. `GNUNET_TESTBED_barrier_wait()` connects to the local barrier service running on the same host the peer is running on and registers that the caller has reached the barrier and is waiting for the barrier to be

crossed. Note that this function can only be used by peers which are started by testbed as this function tries to access the local barrier service which is part of the testbed controller service. Calling `GNUNET_TESTBED_barrier_wait()` on an uninitialised barrier results in failure. `GNUNET_TESTBED_barrier_wait_cancel()` cancels the notification registered by `GNUNET_TESTBED_barrier_wait()`.

5.13.4.1 Implementation

Since barriers involve coordination between experiment driver and peers, the barrier service in the testbed controller is split into two components. The first component responds to the message generated by the barrier API used by the experiment driver (functions `GNUNET_TESTBED_barrier_init()` and `GNUNET_TESTBED_barrier_cancel()`) and the second component to the messages generated by barrier API used by peers (functions `GNUNET_TESTBED_barrier_wait()` and `GNUNET_TESTBED_barrier_wait_cancel()`).

Calling `GNUNET_TESTBED_barrier_init()` sends a `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_INIT` message to the master controller. The master controller then registers a barrier and calls `GNUNET_TESTBED_barrier_init()` for each its subcontrollers. In this way barrier initialisation is propagated to the controller hierarchy. While propagating initialisation, any errors at a subcontroller such as timeout during further propagation are reported up the hierarchy back to the experiment driver.

Similar to `GNUNET_TESTBED_barrier_init()`, `GNUNET_TESTBED_barrier_cancel()` propagates `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_CANCEL` message which causes controllers to remove an initialised barrier.

The second component is implemented as a separate service in the binary 'gnunet-service-testbed' which already has the testbed controller service. Although this deviates from the gnunet process architecture of having one service per binary, it is needed in this case as this component needs access to barrier data created by the first component. This component responds to `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_WAIT` messages from local peers when they call `GNUNET_TESTBED_barrier_wait()`. Upon receiving `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_WAIT` message, the service checks if the requested barrier has been initialised before and if it was not initialised, an error status is sent through `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_STATUS` message to the local peer and the connection from the peer is terminated. If the barrier is initialised before, the barrier's counter for reached peers is incremented and a notification is registered to notify the peer when the barrier is reached. The connection from the peer is left open.

When enough peers required to attain the quorum send `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_WAIT` messages, the controller sends a `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_STATUS` message to its parent informing that the barrier is crossed. If the controller has started further subcontrollers, it delays this message until it receives a similar notification from each of those subcontrollers. Finally, the barriers API at the experiment driver receives the `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_STATUS` when the barrier is reached at all the controllers.

The barriers API at the experiment driver responds to the `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_STATUS` message by echoing it back to the master controller and notifying the experiment controller through the notification callback that a barrier has been crossed. The echoed `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_STATUS` message is propagated by the master controller to the controller hierarchy. This propagation triggers the notifications

registered by peers at each of the controllers in the hierarchy. Note the difference between this downward propagation of the `GNUNET_MESSAGE_TYPE_TESTBED_BARRIER_STATUS` message from its upward propagation — the upward propagation is needed for ensuring that the barrier is reached by all the controllers and the downward propagation is for triggering that the barrier is crossed.

5.13.5 Automatic large-scale deployment in the PlanetLab testbed

PlanetLab is a testbed for computer networking and distributed systems research. It was established in 2002 and as of June 2010 was composed of 1090 nodes at 507 sites worldwide.

To automate the GNUnet we created a set of automation tools to simplify the large-scale deployment. We provide you a set of scripts you can use to deploy GNUnet on a set of nodes and manage your installation.

Please also check https://gnunet.org/installation-fedora8-svn and https://gnunet.org/installation-fedora12-svn to find detailled instructions how to install GNUnet on a PlanetLab node.

5.13.5.1 PlanetLab Automation for Fedora8 nodes

5.13.5.2 Install buildslave on PlanetLab nodes running fedora core 8

Since most of the PlanetLab nodes are running the very old Fedora core 8 image, installing the buildslave software is quite some pain. For our PlanetLab testbed we figured out how to install the buildslave software best.

Install Distribute for Python:

```
curl http://python-distribute.org/distribute_setup.py | sudo python
```

Install Distribute for zope.interface <= 3.8.0 (4.0 and 4.0.1 will not work):

```
export PYPI=https://pypi.python.org/packages/source
wget $PYPI/z/zope.interface/zope.interface-3.8.0.tar.gz
tar zvfz zope.interface-3.8.0.tar.gz
cd zope.interface-3.8.0
sudo python setup.py install
```

Install the buildslave software (0.8.6 was the latest version):

```
export GCODE="http://buildbot.googlecode.com/files"
wget $GCODE/buildbot-slave-0.8.6p1.tar.gz
tar xvfz buildbot-slave-0.8.6p1.tar.gz
cd buildslave-0.8.6p1
sudo python setup.py install
```

The setup will download the matching twisted package and install it. It will also try to install the latest version of zope.interface which will fail to install. Buildslave will work anyway since version 3.8.0 was installed before!

5.13.5.3 Setup a new PlanetLab testbed using GPLMT

- Get a new slice and assign nodes Ask your PlanetLab PI to give you a new slice and assign the nodes you need

- Install a buildmaster You can stick to the buildbot documentation: http://buildbot.net/buildbot/docs/current/manual/installation.html
- Install the buildslave software on all nodes To install the buildslave on all nodes assigned to your slice you can use the tasklist install_buildslave_fc8.xml provided with GPLMT:

```
./gplmt.py -c contrib/tumple_gnunet.conf -t \
contrib/tasklists/install_buildslave_fc8.xml -a -p <planetlab password>
```

- Create the buildmaster configuration and the slave setup commands

  The master and the and the slaves have need to have credentials and the master has to have all nodes configured. This can be done with the create_buildbot_configuration.py script in the scripts directory.

  This scripts takes a list of nodes retrieved directly from PlanetLab or read from a file and a configuration template and creates:

  - a tasklist which can be executed with gplmt to setup the slaves
  - a master.cfg file containing a PlanetLab nodes

  A configuration template is included in the <contrib>, most important is that the script replaces the following tags in the template:

  %GPLMT_BUILDER_DEFINITION : GPLMT_BUILDER_SUMMARY GPLMT_SLAVES %GPLMT_SCHEDULER_BUILDERS

  Create configuration for all nodes assigned to a slice:

```
./create_buildbot_configuration.py -u <planetlab username> \
-p <planetlab password> -s <slice> -m <buildmaster+port> \
-t <template>
```

  Create configuration for some nodes in a file:

```
./create_buildbot_configuration.p -f <node_file> \
-m <buildmaster+port> -t <template>
```

- Copy the master.cfg to the buildmaster and start it Use buildbot start <basedir> to start the server
- Setup the buildslaves

5.13.5.4 Why do i get an ssh error when using the regex profiler?

Why do i get an ssh error "Permission denied (publickey,password)." when using the regex profiler although passwordless ssh to localhost works using publickey and ssh-agent?

You have to generate a public/private-key pair with no password: ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_localhost and then add the following to your ~/.ssh/config file:

```
Host 127.0.0.1 IdentityFile ~/.ssh/id_localhost
```

now make sure your hostsfile looks like

```
[USERNAME]@127.0.0.1:22 [USERNAME]@127.0.0.1:22
```

You can test your setup by running ssh 127.0.0.1 in a terminal and then in the opened session run it again. If you were not asked for a password on either login, then you should be good to go.

5.13.6 TESTBED Caveats

This section documents a few caveats when using the GNUnet testbed subsystem.

5.13.6.1 CORE must be started

A uncomplicated issue is bug #3993[5]: Your configuration MUST somehow ensure that for each peer the `CORE` service is started when the peer is setup, otherwise `TESTBED` may fail to connect peers when the topology is initialized, as `TESTBED` will start some `CORE` services but not necessarily all (but it relies on all of them running). The easiest way is to set

```
[core]
FORCESTART = YES
```

in the configuration file. Alternatively, having any service that directly or indirectly depends on `CORE` being started with `FORCESTART` will also do. This issue largely arises if users try to over-optimize by not starting any services with `FORCESTART`.

5.13.6.2 ATS must want the connections

When TESTBED sets up connections, it only offers the respective HELLO information to the TRANSPORT service. It is then up to the ATS service to **decide** to use the connection. The ATS service will typically eagerly establish any connection if the number of total connections is low (relative to bandwidth). Details may further depend on the specific ATS backend that was configured. If ATS decides to NOT establish a connection (even though TESTBED provided the required information), then that connection will count as failed for TESTBED. Note that you can configure TESTBED to tolerate a certain number of connection failures (see '-e' option of gnunet-testbed-profiler). This issue largely arises for dense overlay topologies, especially if you try to create cliques with more than 20 peers.

5.14 libgnunetutil

libgnunetutil is the fundamental library that all GNUnet code builds upon. Ideally, this library should contain most of the platform dependent code (except for user interfaces and really special needs that only few applications have). It is also supposed to offer basic services that most if not all GNUnet binaries require. The code of libgnunetutil is in the `src/util/` directory. The public interface to the library is in the gnunet_util.h header. The functions provided by libgnunetutil fall roughly into the following categories (in roughly the order of importance for new developers):

- logging (common_logging.c)
- memory allocation (common_allocation.c)
- endianess conversion (common_endian.c)
- internationalization (common_gettext.c)
- String manipulation (string.c)
- file access (disk.c)
- buffered disk IO (bio.c)
- time manipulation (time.c)

[5] https://gnunet.org/bugs/view.php?id=3993 (`https://gnunet.org/bugs/view.php?id=3993`)

- configuration parsing (configuration.c)
- command-line handling (getopt*.c)
- cryptography (crypto_*.c)
- data structures (container_*.c)
- CPS-style scheduling (scheduler.c)
- Program initialization (program.c)
- Networking (network.c, client.c, server*.c, service.c)
- message queueing (mq.c)
- bandwidth calculations (bandwidth.c)
- Other OS-related (os*.c, plugin.c, signal.c)
- Pseudonym management (pseudonym.c)

It should be noted that only developers that fully understand this entire API will be able to write good GNUnet code.

Ideally, porting GNUnet should only require porting the gnunetutil library. More test-cases for the gnunetutil APIs are therefore a great way to make porting of GNUnet easier.

5.14.1 Logging

GNUnet is able to log its activity, mostly for the purposes of debugging the program at various levels.

`gnunet_common.h` defines several **log levels**:

ERROR for errors (really problematic situations, often leading to
: crashes)

WARNING for warnings (troubling situations that might have
: negative consequences, although not fatal)

INFO for various information.
: Used somewhat rarely, as GNUnet statistics is used to hold and display most of the information that users might find interesting.

DEBUG for debugging.
: Does not produce much output on normal builds, but when extra logging is enabled at compile time, a staggering amount of data is outputted under this log level.

Normal builds of GNUnet (configured with `--enable-logging[=yes]`) are supposed to log nothing under DEBUG level. The `--enable-logging=verbose` configure option can be used to create a build with all logging enabled. However, such build will produce large amounts of log data, which is inconvenient when one tries to hunt down a specific problem.

To mitigate this problem, GNUnet provides facilities to apply a filter to reduce the logs:

Logging by default When no log levels are configured in any other
: way (see below), GNUnet will default to the WARNING log level. This mostly applies to GNUnet command line utilities, services and daemons; tests will always set log level to WARNING or, if `--enable-logging=verbose` was passed to configure, to DEBUG. The default level is suggested for normal operation.

The -L option Most GNUnet executables accept an "-L loglevel" or
: "–log=loglevel" option. If used, it makes the process set a global log level to "loglevel". Thus it is possible to run some processes with -L DEBUG, for example, and others with -L ERROR to enable specific settings to diagnose problems with a particular process.

Configuration files. Because GNUnet
: service and deamon processes are usually launched by gnunet-arm, it is not possible to pass different custom command line options directly to every one of them. The options passed to `gnunet-arm` only affect gnunet-arm and not the rest of GNUnet. However, one can specify a configuration key "OPTIONS" in the section that corresponds to a service or a daemon, and put a value of "-L loglevel" there. This will make the respective service or daemon set its log level to "loglevel" (as the value of OPTIONS will be passed as a command-line argument).

  To specify the same log level for all services without creating separate "OPTIONS" entries in the configuration for each one, the user can specify a config key "GLOBAL_POSTFIX" in the [arm] section of the configuration file. The value of GLOBAL_POSTFIX will be appended to all command lines used by the ARM service to run other services. It can contain any option valid for all GNUnet commands, thus in particular the "-L loglevel" option. The ARM service itself is, however, unaffected by GLOBAL_POSTFIX; to set log level for it, one has to specify "OPTIONS" key in the [arm] section.

Environment variables.
: Setting global per-process log levels with "-L loglevel" does not offer sufficient log filtering granularity, as one service will call interface libraries and supporting libraries of other GNUnet services, potentially producing lots of debug log messages from these libraries. Also, changing the config file is not always convenient (especially when running the GNUnet test suite). To fix that, and to allow GNUnet to use different log filtering at runtime without re-compiling the whole source tree, the log calls were changed to be configurable at run time. To configure them one has to define environment variables "GNUNET_FORCE_LOGFILE", "GNUNET_LOG" and/or "GNUNET_FORCE_LOG":

  - "GNUNET_LOG" only affects the logging when no global log level is configured by any other means (that is, the process does not explicitly set its own log level, there are no "-L loglevel" options on command line or in configuration files), and can be used to override the default WARNING log level.
  - "GNUNET_FORCE_LOG" will completely override any other log configuration options given.
  - "GNUNET_FORCE_LOGFILE" will completely override the location of the file to log messages to. It should contain a relative or absolute file name. Setting GNUNET_FORCE_LOGFILE is equivalent to passing "–log-file=logfile" or "-l logfile" option (see below). It supports "[]" format in file names, but not "{}" (see below).

Because environment variables are inherited by child processes when they are launched, starting or re-starting the ARM service with these variables will propagate them to all other services.

"GNUNET_LOG" and "GNUNET_FORCE_LOG" variables must contain a specially formatted **logging definition** string, which looks like this:

```
[component];[file];[function];[from_line[-to_line]];loglevel[/component...]
```

That is, a logging definition consists of definition entries, separated by slashes ('/'). If only one entry is present, there is no need to add a slash to its end (although it is not forbidden either). All definition fields (component, file, function, lines and loglevel) are mandatory, but (except for the loglevel) they can be empty. An empty field means "match anything". Note that even if fields are empty, the semicolon (';') separators must be present. The loglevel field is mandatory, and must contain one of the log level names (ERROR, WARNING, INFO or DEBUG). The lines field might contain one non-negative number, in which case it matches only one line, or a range "from_line-to_line", in which case it matches any line in the interval [from_line;to_line] (that is, including both start and end line). GNUnet mostly defaults component name to the name of the service that is implemented in a process ('transport', 'core', 'peerinfo', etc), but logging calls can specify custom component names using `GNUNET_log_from`. File name and function name are provided by the compiler (__FILE__ and __FUNCTION__ built-ins).

Component, file and function fields are interpreted as non-extended regular expressions (GNU libc regex functions are used). Matching is case-sensitive, "^" and "$" will match the beginning and the end of the text. If a field is empty, its contents are automatically replaced with a ".*" regular expression, which matches anything. Matching is done in the default way, which means that the expression matches as long as it's contained anywhere in the string. Thus "GNUNET_" will match both "GNUNET_foo" and "BAR_GNUNET_BAZ". Use '^' and/or '$' to make sure that the expression matches at the start and/or at the end of the string. The semicolon (';') can't be escaped, and GNUnet will not use it in component names (it can't be used in function names and file names anyway).

Every logging call in GNUnet code will be (at run time) matched against the log definitions passed to the process. If a log definition fields are matching the call arguments, then the call log level is compared the the log level of that definition. If the call log level is less or equal to the definition log level, the call is allowed to proceed. Otherwise the logging call is forbidden, and nothing is logged. If no definitions matched at all, GNUnet will use the global log level or (if a global log level is not specified) will default to WARNING (that is, it will allow the call to proceed, if its level is less or equal to the global log level or to WARNING).

That is, definitions are evaluated from left to right, and the first matching definition is used to allow or deny the logging call. Thus it is advised to place narrow definitions at the beginning of the logdef string, and generic definitions - at the end.

Whether a call is allowed or not is only decided the first time this particular call is made. The evaluation result is then cached, so that any attempts to make the same call later will

be allowed or disallowed right away. Because of that runtime log level evaluation should not significantly affect the process performance. Log definition parsing is only done once, at the first call to `GNUNET_log_setup ()` made by the process (which is usually done soon after it starts).

At the moment of writing there is no way to specify logging definitions from configuration files, only via environment variables.

At the moment GNUnet will stop processing a log definition when it encounters an error in definition formatting or an error in regular expression syntax, and will not report the failure in any way.

5.14.1.1 Examples

`GNUNET_FORCE_LOG=";;;;DEBUG" gnunet-arm -s` Start GNUnet
: process tree, running all processes with DEBUG level (one should be careful with it, as log files will grow at alarming rate!)

`GNUNET_FORCE_LOG="core;;;;DEBUG" gnunet-arm -s` Start GNUnet
: process tree, running the core service under DEBUG level (everything else will use configured or default level).

Start GNUnet process tree, allowing any logging calls from
: gnunet-service-transport_validation.c (everything else will use configured or default level).

```
GNUNET_FORCE_LOG=";gnunet-service-transport_validation.c;;; DEBUG" \
gnunet-arm -s
```

Start GNUnet process tree, allowing any logging calls from
: gnunet-gnunet-service-fs_push.c (everything else will use configured or default level).

```
GNUNET_FORCE_LOG="fs;gnunet-service-fs_push.c;;;DEBUG" gnunet-arm -s
```

Start GNUnet process tree, allowing any logging calls from the
: GNUNET_NETWORK_socket_select function (everything else will use configured or default level).

```
GNUNET_FORCE_LOG=";;GNUNET_NETWORK_socket_select;;DEBUG" gnunet-arm -s
```

Start GNUnet process tree, allowing any logging calls from the
: components that have "transport" in their names, and are made from function that have "send" in their names. Everything else will be allowed to be logged only if it has WARNING level.

```
GNUNET_FORCE_LOG="transport.*;;.*send.*;;DEBUG/;;;;WARNING" gnunet-arm -s
```

On Windows, one can use batch files to run GNUnet processes with special environment variables, without affecting the whole system. Such batch file will look like this:

```
set GNUNET_FORCE_LOG=;;do_transmit;;DEBUG gnunet-arm -s
```

(note the absence of double quotes in the environment variable definition, as opposed to earlier examples, which use the shell). Another limitation, on Windows, GNUNET_FORCE_LOGFILE **MUST** be set in order to GNUNET_FORCE_LOG to work.

5.14.1.2 Log files

GNUnet can be told to log everything into a file instead of stderr (which is the default) using the "–log-file=logfile" or "-l logfile" option. This option can also be passed via command line, or from the "OPTION" and "GLOBAL_POSTFIX" configuration keys (see above). The file name passed with this option is subject to GNUnet filename expansion. If specified in "GLOBAL_POSTFIX", it is also subject to ARM service filename expansion, in particular, it may contain "{}" (left and right curly brace) sequence, which will be replaced by ARM with the name of the service. This is used to keep logs from more than one service separate, while only specifying one template containing "{}" in GLOBAL_POSTFIX.

As part of a secondary file name expansion, the first occurrence of "[]" sequence ("left square brace" followed by "right square brace") in the file name will be replaced with a process identifier or the process when it initializes its logging subsystem. As a result, all processes will log into different files. This is convenient for isolating messages of a particular process, and prevents I/O races when multiple processes try to write into the file at the same time. This expansion is done independently of "{}" expansion that ARM service does (see above).

The log file name that is specified via "-l" can contain format characters from the 'strftime' function family. For example, "%Y" will be replaced with the current year. Using "basename-%Y-%m-%d.log" would include the current year, month and day in the log file. If a GNUnet process runs for long enough to need more than one log file, it will eventually clean up old log files. Currently, only the last three log files (plus the current log file) are preserved. So once the fifth log file goes into use (so after 4 days if you use "%Y-%m-%d" as above), the first log file will be automatically deleted. Note that if your log file name only contains "%Y", then log files would be kept for 4 years and the logs from the first year would be deleted once year 5 begins. If you do not use any date-related string format codes, logs would never be automatically deleted by GNUnet.

5.14.1.3 Updated behavior of GNUNET_log

It's currently quite common to see constructions like this all over the code:

```
#if MESH_DEBUG
GNUNET_log (GNUNET_ERROR_TYPE_DEBUG, "MESH: client disconnected\n");
#endif
```

The reason for the #if is not to avoid displaying the message when disabled (GNUNET_ERROR_TYPE takes care of that), but to avoid the compiler including it in the binary at all, when compiling GNUnet for platforms with restricted storage space / memory (MIPS routers, ARM plug computers / dev boards, etc).

This presents several problems: the code gets ugly, hard to write and it is very easy to forget to include the #if guards, creating non-consistent code. A new change in GNUNET_log aims to solve these problems.

This change requires to `./configure` with at least `--enable-logging=verbose` to see debug messages.

Here is an example of code with dense debug statements:

```
switch (restrict_topology) {
case GNUNET_TESTING_TOPOLOGY_CLIQUE:#if VERBOSE_TESTING
GNUNET_log (GNUNET_ERROR_TYPE_DEBUG, _("Blacklisting all but clique
```

```
topology\n")); #endif unblacklisted_connections = create_clique (pg,
&remove_connections, BLACKLIST, GNUNET_NO); break; case
GNUNET_TESTING_TOPOLOGY_SMALL_WORLD_RING: #if VERBOSE_TESTING GNUNET_log
(GNUNET_ERROR_TYPE_DEBUG, _("Blacklisting all but small world (ring)
topology\n")); #endif unblacklisted_connections = create_small_world_ring
(pg,&remove_connections, BLACKLIST); break;
```

Pretty hard to follow, huh?

From now on, it is not necessary to include the #if / #endif statements to achieve the same behavior. The `GNUNET_log` and `GNUNET_log_from` macros take care of it for you, depending on the configure option:

- If `--enable-logging` is set to `no`, the binary will contain no log messages at all.
- If `--enable-logging` is set to `yes`, the binary will contain no DEBUG messages, and therefore running with `-L DEBUG` will have no effect. Other messages (ERROR, WARNING, INFO, etc) will be included.
- If `--enable-logging` is set to `verbose`, or `veryverbose` the binary will contain DEBUG messages (still, it will be neccessary to run with `-L DEBUG` or set the DEBUG config option to show them).

If you are a developer:

- please make sure that you `./configure --enable-logging={verbose,veryverbose}`, so you can see DEBUG messages.
- please remove the `#if` statements around `GNUNET_log (GNUNET_ERROR_TYPE_DEBUG, ...)` lines, to improve the readibility of your code.

Since now activating DEBUG automatically makes it VERBOSE and activates **all** debug messages by default, you probably want to use the https://gnunet.org/logging functionality to filter only relevant messages. A suitable configuration could be:

```
$ export GNUNET_FORCE_LOG="^YOUR_SUBSYSTEM$;;;;DEBUG/;;;;WARNING"
```

Which will behave almost like enabling DEBUG in that subsytem before the change. Of course you can adapt it to your particular needs, this is only a quick example.

5.14.2 Interprocess communication API (IPC)

In GNUnet a variety of new message types might be defined and used in interprocess communication, in this tutorial we use the `struct AddressLookupMessage` as a example to introduce how to construct our own message type in GNUnet and how to implement the message communication between service and client. (Here, a client uses the `struct AddressLookupMessage` as a request to ask the server to return the address of any other peer connecting to the service.)

5.14.2.1 Define new message types

First of all, you should define the new message type in `gnunet_protocols.h`:

```
 // Request to look addresses of peers in server.
#define GNUNET_MESSAGE_TYPE_TRANSPORT_ADDRESS_LOOKUP 29
  // Response to the address lookup request.
#define GNUNET_MESSAGE_TYPE_TRANSPORT_ADDRESS_REPLY 30
```

5.14.2.2 Define message struct

After the type definition, the specified message structure should also be described in the header file, e.g. transport.h in our case.

```
struct AddressLookupMessage {
  struct GNUNET_MessageHeader header;
  int32_t numeric_only GNUNET_PACKED;
  struct GNUNET_TIME_AbsoluteNBO timeout;
  uint32_t addrlen GNUNET_PACKED;
  /* followed by 'addrlen' bytes of the actual address, then
     followed by the 0-terminated name of the transport */ };
GNUNET_NETWORK_STRUCT_END
```

Please note `GNUNET_NETWORK_STRUCT_BEGIN` and `GNUNET_PACKED` which both ensure correct alignment when sending structs over the network.

5.14.2.3 Client - Establish connection

At first, on the client side, the underlying API is employed to create a new connection to a service, in our example the transport service would be connected.

```
struct GNUNET_CLIENT_Connection *client;
client = GNUNET_CLIENT_connect ("transport", cfg);
```

5.14.2.4 Client - Initialize request message

When the connection is ready, we initialize the message. In this step, all the fields of the message should be properly initialized, namely the size, type, and some extra user-defined data, such as timeout, name of transport, address and name of transport.

```
struct AddressLookupMessage *msg;
size_t len = sizeof (struct AddressLookupMessage)
  + addressLen
  + strlen (nameTrans)
  + 1;
msg->header->size = htons (len);
msg->header->type = htons
(GNUNET_MESSAGE_TYPE_TRANSPORT_ADDRESS_LOOKUP);
msg->timeout = GNUNET_TIME_absolute_hton (abs_timeout);
msg->addrlen = htonl (addressLen);
char *addrbuf = (char *) &msg[1];
memcpy (addrbuf, address, addressLen);
char *tbuf = &addrbuf[addressLen];
memcpy (tbuf, nameTrans, strlen (nameTrans) + 1);
```

Note that, here the functions `htonl`, `htons` and `GNUNET_TIME_absolute_hton` are applied to convert little endian into big endian, about the usage of the big/small endian order and the corresponding conversion function please refer to Introduction of Big Endian and Little Endian.

5.14.2.5 Client - Send request and receive response

FIXME: This is very outdated, see the tutorial for the current API!

Next, the client would send the constructed message as a request to the service and wait for the response from the service. To accomplish this goal, there are a number of API calls that can be used. In this example, `GNUNET_CLIENT_transmit_and_get_response` is chosen as the most appropriate function to use.

```
GNUNET_CLIENT_transmit_and_get_response
(client, msg->header, timeout, GNUNET_YES, &address_response_processor,
arp_ctx);
```

the argument `address_response_processor` is a function with `GNUNET_CLIENT_MessageHandler` type, which is used to process the reply message from the service.

5.14.2.6 Server - Startup service

After receiving the request message, we run a standard GNUnet service startup sequence using `GNUNET_SERVICE_run`, as follows,

```
int main(int argc, char**argv) {
  GNUNET_SERVICE_run(argc, argv, "transport"
  GNUNET_SERVICE_OPTION_NONE, &run, NULL)); }
```

5.14.2.7 Server - Add new handles for specified messages

in the function above the argument `run` is used to initiate transport service,and defined like this:

```
static void run (void *cls,
struct GNUNET_SERVER_Handle *serv,
const struct GNUNET_CONFIGURATION_Handle *cfg) {
  GNUNET_SERVER_add_handlers (serv, handlers); }
```

Here, `GNUNET_SERVER_add_handlers` must be called in the run function to add new handlers in the service. The parameter `handlers` is a list of `struct GNUNET_SERVER_MessageHandler` to tell the service which function should be called when a particular type of message is received, and should be defined in this way:

```
static struct GNUNET_SERVER_MessageHandler handlers[] = {
  {&handle_start,
   NULL,
   GNUNET_MESSAGE_TYPE_TRANSPORT_START,
   0},
  {&handle_send,
   NULL,
   GNUNET_MESSAGE_TYPE_TRANSPORT_SEND,
   0},
  {&handle_try_connect,
   NULL,
   GNUNET_MESSAGE_TYPE_TRANSPORT_TRY_CONNECT,
   sizeof (struct TryConnectMessage)
  },
  {&handle_address_lookup,
   NULL,
   GNUNET_MESSAGE_TYPE_TRANSPORT_ADDRESS_LOOKUP,
```

```
     0},
    {NULL,
     NULL,
     0,
     0}
  };
```

As shown, the first member of the struct in the first area is a callback function, which is called to process the specified message types, given as the third member. The second parameter is the closure for the callback function, which is set to `NULL` in most cases, and the last parameter is the expected size of the message of this type, usually we set it to 0 to accept variable size, for special cases the exact size of the specified message also can be set. In addition, the terminator sign depicted as `{NULL, NULL, 0, 0}` is set in the last aera.

5.14.2.8 Server - Process request message

After the initialization of transport service, the request message would be processed. Before handling the main message data, the validity of this message should be checked out, e.g., to check whether the size of message is correct.

```
size = ntohs (message->size);
if (size < sizeof (struct AddressLookupMessage)) {
  GNUNET_break_op (0);
  GNUNET_SERVER_receive_done (client, GNUNET_SYSERR);
  return; }
```

Note that, opposite to the construction method of the request message in the client, in the server the function `nothl` and `ntohs` should be employed during the extraction of the data from the message, so that the data in big endian order can be converted back into little endian order. See more in detail please refer to Introduction of Big Endian and Little Endian.

Moreover in this example, the name of the transport stored in the message is a 0-terminated string, so we should also check whether the name of the transport in the received message is 0-terminated:

```
nameTransport = (const char *) &address[addressLen];
if (nameTransport[size - sizeof
                 (struct AddressLookupMessage)
                 - addressLen - 1] != '\0') {
  GNUNET_break_op (0);
  GNUNET_SERVER_receive_done (client,
                              GNUNET_SYSERR);
  return; }
```

Here, `GNUNET_SERVER_receive_done` should be called to tell the service that the request is done and can receive the next message. The argument `GNUNET_SYSERR` here indicates that the service didn't understand the request message, and the processing of this request would be terminated.

In comparison to the aforementioned situation, when the argument is equal to `GNUNET_OK`, the service would continue to process the requst message.

5.14.2.9 Server - Response to client

Once the processing of current request is done, the server should give the response to the client. A new `struct AddressLookupMessage` would be produced by the server in a similar way as the client did and sent to the client, but here the type should be `GNUNET_MESSAGE_TYPE_TRANSPORT_ADDRESS_REPLY` rather than `GNUNET_MESSAGE_TYPE_TRANSPORT_ADDRESS_LOOKUP` in client.

```
struct AddressLookupMessage *msg;
size_t len = sizeof (struct AddressLookupMessage)
  + addressLen
  + strlen (nameTrans) + 1;
msg->header->size = htons (len);
msg->header->type = htons
  (GNUNET_MESSAGE_TYPE_TRANSPORT_ADDRESS_REPLY);

// ...

struct GNUNET_SERVER_TransmitContext *tc;
tc = GNUNET_SERVER_transmit_context_create (client);
GNUNET_SERVER_transmit_context_append_data
(tc,
 NULL,
 0,
 GNUNET_MESSAGE_TYPE_TRANSPORT_ADDRESS_REPLY);
GNUNET_SERVER_transmit_context_run (tc, rtimeout);
```

Note that, there are also a number of other APIs provided to the service to send the message.

5.14.2.10 Server - Notification of clients

Often a service needs to (repeatedly) transmit notifications to a client or a group of clients. In these cases, the client typically has once registered for a set of events and then needs to receive a message whenever such an event happens (until the client disconnects). The use of a notification context can help manage message queues to clients and handle disconnects. Notification contexts can be used to send individualized messages to a particular client or to broadcast messages to a group of clients. An individualized notification might look like this:

```
GNUNET_SERVER_notification_context_unicast(nc,
                                           client,
                                           msg,
                                           GNUNET_YES);
```

Note that after processing the original registration message for notifications, the server code still typically needs to call `GNUNET_SERVER_receive_done` so that the client can transmit further messages to the server.

5.14.2.11 Conversion between Network Byte Order (Big Endian) and Host Byte Order

Here we can simply comprehend big endian and little endian as Network Byte Order and Host Byte Order respectively. What is the difference between both two?

Usually in our host computer we store the data byte as Host Byte Order, for example, we store a integer in the RAM which might occupies 4 Byte, as Host Byte Order the higher Byte would be stored at the lower address of RAM, and the lower Byte would be stored at the higher address of RAM. However, contrast to this, Network Byte Order just take the totally opposite way to store the data, says, it will store the lower Byte at the lower address, and the higher Byte will stay at higher address.

For the current communication of network, we normally exchange the information by surveying the data package, every two host wants to communicate with each other must send and receive data package through network. In order to maintain the identity of data through the transmission in the network, the order of the Byte storage must changed before sending and after receiving the data.

There ten convenient functions to realize the conversion of Byte Order in GNUnet, as following:

uint16_t htons(uint16_t hostshort) Convert host byte order to net
: byte order with short int

uint32_t htonl(uint32_t hostlong) Convert host byte
: order to net byte order with long int

uint16_t ntohs(uint16_t netshort)
: Convert net byte order to host byte order with short int

uint32_t
: ntohl(uint32_t netlong) Convert net byte order to host byte order with long int

unsigned long long GNUNET_ntohll (unsigned long long netlonglong)
: Convert net byte order to host byte order with long long int

unsigned long long GNUNET_htonll (unsigned long long hostlonglong)
: Convert host byte order to net byte order with long long int

struct GNUNET_TIME_RelativeNBO GNUNET_TIME_relative_hton
: (struct GNUNET_TIME_Relative a) Convert relative time to network byte order.

struct GNUNET_TIME_Relative GNUNET_TIME_relative_ntoh
: (struct GNUNET_TIME_RelativeNBO a) Convert relative time from network byte order.

struct GNUNET_TIME_AbsoluteNBO GNUNET_TIME_absolute_hton
: (struct GNUNET_TIME_Absolute a) Convert relative time to network byte order.

struct GNUNET_TIME_Absolute GNUNET_TIME_absolute_ntoh
: (struct GNUNET_TIME_AbsoluteNBO a) Convert relative time from network byte order.

5.14.3 Cryptography API

The gnunetutil APIs provides the cryptographic primitives used in GNUnet. GNUnet uses 2048 bit RSA keys for the session key exchange and for signing messages by peers and most other public-key operations. Most researchers in cryptography consider 2048 bit RSA keys as secure and practically unbreakable for a long time. The API provides functions to create a fresh key pair, read a private key from a file (or create a new file if the file does not exist), encrypt, decrypt, sign, verify and extraction of the public key into a format suitable for network transmission.

For the encryption of files and the actual data exchanged between peers GNUnet uses 256-bit AES encryption. Fresh, session keys are negotiated for every new connection. Again, there is no published technique to break this cipher in any realistic amount of time. The API provides functions for generation of keys, validation of keys (important for checking that decryptions using RSA succeeded), encryption and decryption.

GNUnet uses SHA-512 for computing one-way hash codes. The API provides functions to compute a hash over a block in memory or over a file on disk.

The crypto API also provides functions for randomizing a block of memory, obtaining a single random number and for generating a permuation of the numbers 0 to n-1. Random number generation distinguishes between WEAK and STRONG random number quality; WEAK random numbers are pseudo-random whereas STRONG random numbers use entropy gathered from the operating system.

Finally, the crypto API provides a means to deterministically generate a 1024-bit RSA key from a hash code. These functions should most likely not be used by most applications; most importantly, GNUNET_CRYPTO_rsa_key_create_from_hash does not create an RSA-key that should be considered secure for traditional applications of RSA.

5.14.4 Message Queue API

Introduction Often, applications need to queue messages that are to be sent to other GNUnet peers, clients or services. As all of GNUnet's message-based communication APIs, by design, do not allow messages to be queued, it is common to implement custom message queues manually when they are needed. However, writing very similar code in multiple places is tedious and leads to code duplication.

MQ (for Message Queue) is an API that provides the functionality to implement and use message queues. We intend to eventually replace all of the custom message queue implementations in GNUnet with MQ.

Basic Concepts The two most important entities in MQ are queues and envelopes.

Every queue is backed by a specific implementation (e.g. for mesh, stream, connection, server client, etc.) that will actually deliver the queued messages. For convenience, some queues also allow to specify a list of message handlers. The message queue will then also wait for incoming messages and dispatch them appropriately.

An envelope holds the the memory for a message, as well as metadata (Where is the envelope queued? What should happen after it has been sent?). Any envelope can only be queued in one message queue.

Creating Queues The following is a list of currently available message queues. Note that to avoid layering issues, message queues for higher level APIs are not part of `libgnunetutil`, but the respective API itself provides the queue implementation.

`GNUNET_MQ_queue_for_connection_client`
: Transmits queued messages over a `GNUNET_CLIENT_Connection` handle. Also supports receiving with message handlers.

`GNUNET_MQ_queue_for_server_client`
: Transmits queued messages over a `GNUNET_SERVER_Client` handle. Does not support incoming message handlers.

`GNUNET_MESH_mq_create` Transmits queued messages over a
: `GNUNET_MESH_Tunnel` handle. Does not support incoming message handlers.

`GNUNET_MQ_queue_for_callbacks` This is the most general
: implementation. Instead of delivering and receiving messages with one of GNUnet's communication APIs, implementation callbacks are called. Refer to "Implementing Queues" for a more detailed explanation.

Allocating Envelopes A GNUnet message (as defined by the GNUNET_MessageHeader) has three parts: The size, the type, and the body.

MQ provides macros to allocate an envelope containing a message conveniently, automatically setting the size and type fields of the message.

Consider the following simple message, with the body consisting of a single number value.

```
struct NumberMessage {
  /** Type: GNUNET_MESSAGE_TYPE_EXAMPLE_1 */
  struct GNUNET_MessageHeader header;
  uint32_t number GNUNET_PACKED;
};
```

An envelope containing an instance of the NumberMessage can be constructed like this:

```
struct GNUNET_MQ_Envelope *ev;
struct NumberMessage *msg;
ev = GNUNET_MQ_msg (msg, GNUNET_MESSAGE_TYPE_EXAMPLE_1);
msg->number = htonl (42);
```

In the above code, `GNUNET_MQ_msg` is a macro. The return value is the newly allocated envelope. The first argument must be a pointer to some `struct` containing a `struct GNUNET_MessageHeader header` field, while the second argument is the desired message type, in host byte order.

The `msg` pointer now points to an allocated message, where the message type and the message size are already set. The message's size is inferred from the type of the `msg` pointer: It will be set to 'sizeof(*msg)', properly converted to network byte order.

If the message body's size is dynamic, the the macro `GNUNET_MQ_msg_extra` can be used to allocate an envelope whose message has additional space allocated after the `msg` structure.

If no structure has been defined for the message, `GNUNET_MQ_msg_header_extra` can be used to allocate additional space after the message header. The first argument then must be a pointer to a `GNUNET_MessageHeader`.

Envelope Properties A few functions in MQ allow to set additional properties on envelopes:

`GNUNET_MQ_notify_sent` Allows to specify a function that will
be called once the envelope's message has been sent irrevocably. An envelope can be canceled precisely up to the point where the notify sent callback has been called.

`GNUNET_MQ_disable_corking` No corking will be used when
sending the message. Not every queue supports this flag, per default, envelopes are sent with corking.

Sending Envelopes Once an envelope has been constructed, it can be queued for sending with `GNUNET_MQ_send`.

Note that in order to avoid memory leaks, an envelope must either be sent (the queue will free it) or destroyed explicitly with `GNUNET_MQ_discard`.

Canceling Envelopes An envelope queued with `GNUNET_MQ_send` can be canceled with `GNUNET_MQ_cancel`. Note that after the notify sent callback has been called, canceling a message results in undefined behavior. Thus it is unsafe to cancel an envelope that does not have a notify sent callback. When canceling an envelope, it is not necessary to call `GNUNET_MQ_discard`, and the envelope can't be sent again.

Implementing Queues `TODO`

5.14.5 Service API

Most GNUnet code lives in the form of services. Services are processes that offer an API for other components of the system to build on. Those other components can be command-line tools for users, graphical user interfaces or other services. Services provide their API using an IPC protocol. For this, each service must listen on either a TCP port or a UNIX domain socket; for this, the service implementation uses the server API. This use of server is exposed directly to the users of the service API. Thus, when using the service API, one is usually also often using large parts of the server API. The service API provides various convenience functions, such as parsing command-line arguments and the configuration file, which are not found in the server API. The dual to the service/server API is the client API, which can be used to access services.

The most common way to start a service is to use the `GNUNET_SERVICE_run` function from the program's main function. `GNUNET_SERVICE_run` will then parse the command line and configuration files and, based on the options found there, start the server. It will then give back control to the main program, passing the server and the configuration to the `GNUNET_SERVICE_Main` callback. `GNUNET_SERVICE_run` will also take care of starting the scheduler loop. If this is inappropriate (for example, because the scheduler loop is already running), `GNUNET_SERVICE_start` and related functions provide an alternative to `GNUNET_SERVICE_run`.

When starting a service, the service_name option is used to determine which sections in the configuration file should be used to configure the service. A typical value here is the name of the `src/` sub-directory, for example `statistics`. The same string would also be given to `GNUNET_CLIENT_connect` to access the service.

Once a service has been initialized, the program should use the `GNUNET_SERVICE_Main` callback to register message handlers using `GNUNET_SERVER_add_handlers`. The service will already have registered a handler for the "TEST" message.

The option bitfield (`enum GNUNET_SERVICE_Options`) determines how a service should behave during shutdown. There are three key strategies:

instant (`GNUNET_SERVICE_OPTION_NONE`)
: Upon receiving the shutdown signal from the scheduler, the service immediately terminates the server, closing all existing connections with clients.

manual (`GNUNET_SERVICE_OPTION_MANUAL_SHUTDOWN`)
: The service does nothing by itself during shutdown. The main program will need to take the appropriate action by calling GNUNET_SERVER_destroy or GNUNET_SERVICE_stop (depending on how the service was initialized) to terminate the service. This method is used by gnunet-service-arm and rather uncommon.

soft (`GNUNET_SERVICE_OPTION_SOFT_SHUTDOWN`)
: Upon receiving the shutdown signal from the scheduler, the service immediately tells the server to stop listening for incoming clients. Requests from normal existing clients are still processed and the server/service terminates once all normal clients have disconnected. Clients that are not expected to ever disconnect (such as clients that monitor performance values) can be marked as 'monitor' clients using GNUNET_SERVER_client_mark_monitor. Those clients will continue to be processed until all 'normal' clients have disconnected. Then, the server will terminate, closing the monitor connections. This mode is for example used by 'statistics', allowing existing 'normal' clients to set (possibly persistent) statistic values before terminating.

5.14.6 Optimizing Memory Consumption of GNUnet's (Multi-) Hash Maps

A commonly used data structure in GNUnet is a (multi-)hash map. It is most often used to map a peer identity to some data structure, but also to map arbitrary keys to values (for example to track requests in the distributed hash table or in file-sharing). As it is commonly used, the DHT is actually sometimes responsible for a large share of GNUnet's overall memory consumption (for some processes, 30% is not uncommon). The following text documents some API quirks (and their implications for applications) that were recently introduced to minimize the footprint of the hash map.

5.14.6.1 Analysis

The main reason for the "excessive" memory consumption by the hash map is that GNUnet uses 512-bit cryptographic hash codes — and the (multi-)hash map also uses the same 512-bit 'struct GNUNET_HashCode'. As a result, storing just the keys requires 64 bytes of memory for each key. As some applications like to keep a large number of entries in the hash map (after all, that's what maps are good for), 64 bytes per hash is significant: keeping a pointer to the value and having a linked list for collisions consume between 8 and 16 bytes, and 'malloc' may add about the same overhead per allocation, putting us in the 16 to 32 byte per entry ballpark. Adding a 64-byte key then triples the overall memory requirement for the hash map.

To make things "worse", most of the time storing the key in the hash map is not required: it is typically already in memory elsewhere! In most cases, the values stored in the hash

map are some application-specific struct that _also_ contains the hash. Here is a simplified example:

```
struct MyValue {
struct GNUNET_HashCode key;
unsigned int my_data; };

// ...
val = GNUNET_malloc (sizeof (struct MyValue));
val->key = key;
val->my_data = 42;
GNUNET_CONTAINER_multihashmap_put (map, &key, val, ...);
```

This is a common pattern as later the entries might need to be removed, and at that time it is convenient to have the key immediately at hand:

```
GNUNET_CONTAINER_multihashmap_remove (map, &val->key, val);
```

Note that here we end up with two times 64 bytes for the key, plus maybe 64 bytes total for the rest of the 'struct MyValue' and the map entry in the hash map. The resulting redundant storage of the key increases overall memory consumption per entry from the "optimal" 128 bytes to 192 bytes. This is not just an extreme example: overheads in practice are actually sometimes close to those highlighted in this example. This is especially true for maps with a significant number of entries, as there we tend to really try to keep the entries small.

5.14.6.2 Solution

The solution that has now been implemented is to **optionally** allow the hash map to not make a (deep) copy of the hash but instead have a pointer to the hash/key in the entry. This reduces the memory consumption for the key from 64 bytes to 4 to 8 bytes. However, it can also only work if the key is actually stored in the entry (which is the case most of the time) and if the entry does not modify the key (which in all of the code I'm aware of has been always the case if there key is stored in the entry). Finally, when the client stores an entry in the hash map, it **must** provide a pointer to the key within the entry, not just a pointer to a transient location of the key. If the client code does not meet these requirements, the result is a dangling pointer and undefined behavior of the (multi-)hash map API.

5.14.6.3 Migration

To use the new feature, first check that the values contain the respective key (and never modify it). Then, all calls to `GNUNET_CONTAINER_multihashmap_put` on the respective map must be audited and most likely changed to pass a pointer into the value's struct. For the initial example, the new code would look like this:

```
struct MyValue {
struct GNUNET_HashCode key;
unsigned int my_data; };

// ...
val = GNUNET_malloc (sizeof (struct MyValue));
```

```
val->key = key; val->my_data = 42;
GNUNET_CONTAINER_multihashmap_put (map, &val->key, val, ...);
```

Note that `&val` was changed to `&val->key` in the argument to the `put` call. This is critical as often `key` is on the stack or in some other transient data structure and thus having the hash map keep a pointer to `key` would not work. Only the key inside of `val` has the same lifetime as the entry in the map (this must of course be checked as well). Naturally, `val->key` must be intiialized before the `put` call. Once all `put` calls have been converted and double-checked, you can change the call to create the hash map from

```
map =
GNUNET_CONTAINER_multihashmap_create (SIZE, GNUNET_NO);
```

to

```
map = GNUNET_CONTAINER_multihashmap_create (SIZE, GNUNET_YES);
```

If everything was done correctly, you now use about 60 bytes less memory per entry in `map`. However, if now (or in the future) any call to `put` does not ensure that the given key is valid until the entry is removed from the map, undefined behavior is likely to be observed.

5.14.6.4 Conclusion

The new optimization can is often applicable and can result in a reduction in memory consumption of up to 30% in practice. However, it makes the code less robust as additional invariants are imposed on the multi hash map client. Thus applications should refrain from enabling the new mode unless the resulting performance increase is deemed significant enough. In particular, it should generally not be used in new code (wait at least until benchmarks exist).

5.14.6.5 Availability

The new multi hash map code was committed in SVN 24319 (will be in GNUnet 0.9.4). Various subsystems (transport, core, dht, file-sharing) were previously audited and modified to take advantage of the new capability. In particular, memory consumption of the file-sharing service is expected to drop by 20-30% due to this change.

5.14.7 CONTAINER_MDLL API

This text documents the GNUNET_CONTAINER_MDLL API. The GNUNET_CONTAINER_MDLL API is similar to the GNUNET_CONTAINER_DLL API in that it provides operations for the construction and manipulation of doubly-linked lists. The key difference to the (simpler) DLL-API is that the MDLL-version allows a single element (instance of a "struct") to be in multiple linked lists at the same time.

Like the DLL API, the MDLL API stores (most of) the data structures for the doubly-linked list with the respective elements; only the 'head' and 'tail' pointers are stored "elsewhere" — and the application needs to provide the locations of head and tail to each of the calls in the MDLL API. The key difference for the MDLL API is that the "next" and "previous" pointers in the struct can no longer be simply called "next" and "prev" — after all, the element may be in multiple doubly-linked lists, so we cannot just have one "next" and one "prev" pointer!

The solution is to have multiple fields that must have a name of the format "next_XX" and "prev_XX" where "XX" is the name of one of the doubly-linked lists. Here is a simple example:

```
struct MyMultiListElement {
  struct MyMultiListElement *next_ALIST;
  struct MyMultiListElement *prev_ALIST;
  struct MyMultiListElement *next_BLIST;
  struct MyMultiListElement *prev_BLIST;
  void
  *data;
};
```

Note that by convention, we use all-uppercase letters for the list names. In addition, the program needs to have a location for the head and tail pointers for both lists, for example:

```
static struct MyMultiListElement *head_ALIST;
static struct MyMultiListElement *tail_ALIST;
static struct MyMultiListElement *head_BLIST;
static struct MyMultiListElement *tail_BLIST;
```

Using the MDLL-macros, we can now insert an element into the ALIST:

```
GNUNET_CONTAINER_MDLL_insert (ALIST, head_ALIST, tail_ALIST, element);
```

Passing "ALIST" as the first argument to MDLL specifies which of the next/prev fields in the 'struct MyMultiListElement' should be used. The extra "ALIST" argument and the "_ALIST" in the names of the next/prev-members are the only differences between the MDDL and DLL-API. Like the DLL-API, the MDLL-API offers functions for inserting (at head, at tail, after a given element) and removing elements from the list. Iterating over the list should be done by directly accessing the "next_XX" and/or "prev_XX" members.

5.15 Automatic Restart Manager (ARM)

GNUnet's Automated Restart Manager (ARM) is the GNUnet service responsible for system initialization and service babysitting. ARM starts and halts services, detects configuration changes and restarts services impacted by the changes as needed. It's also responsible for restarting services in case of crashes and is planned to incorporate automatic debugging for diagnosing service crashes providing developers insights about crash reasons. The purpose of this document is to give GNUnet developer an idea about how ARM works and how to interact with it.

5.15.1 Basic functionality

- ARM source code can be found under "src/arm". Service processes are managed by the functions in "gnunet-service-arm.c" which is controlled with "gnunet-arm.c" (main function in that file is ARM's entry point).
- The functions responsible for communicating with ARM , starting and stopping services -including ARM service itself- are provided by the ARM API "arm_api.c". Function: GNUNET_ARM_connect() returns to the caller an ARM handle after setting it to the caller's context (configuration and scheduler in use). This handle can be used afterwards by the caller to communicate with ARM. Functions

GNUNET_ARM_start_service() and GNUNET_ARM_stop_service() are used for starting and stopping services respectively.

- A typical example of using these basic ARM services can be found in file test_arm_api.c. The test case connects to ARM, starts it, then uses it to start a service "resolver", stops the "resolver" then stops "ARM".

5.15.2 Key configuration options

Configurations for ARM and services should be available in a .conf file (As an example, see test_arm_api_data.conf). When running ARM, the configuration file to use should be passed to the command:

```
$ gnunet-arm -s -c configuration_to_use.conf
```

If no configuration is passed, the default configuration file will be used (see GNUNET_PREFIX/share/gnunet/defaults.conf which is created from contrib/defaults.conf). Each of the services is having a section starting by the service name between square brackets, for example: "[arm]". The following options configure how ARM configures or interacts with the various services:

PORT Port number on which the service is listening for incoming TCP
: connections. ARM will start the services should it notice a request at this port.

HOSTNAME Specifies on which host the service is deployed. Note
: that ARM can only start services that are running on the local system (but will not check that the hostname matches the local machine name). This option is used by the `gnunet_client_lib.h` implementation to determine which system to connect to. The default is "localhost".

BINARY The name of the service binary file.
OPTIONS To be passed to the service.
PREFIX A command to pre-pend to the actual command, for example,
: running a service with "valgrind" or "gdb"

DEBUG Run in debug mode (much verbosity).
AUTOSTART ARM will listen to UNIX domain socket and/or TCP port of
: the service and start the service on-demand.

FORCESTART ARM will always start this service when the peer
: is started.

ACCEPT_FROM IPv4 addresses the service accepts connections from.
ACCEPT_FROM6 IPv6 addresses the service accepts connections from.

Options that impact the operation of ARM overall are in the "[arm]" section. ARM is a normal service and has (except for AUTOSTART) all of the options that other services do. In addition, ARM has the following options:

GLOBAL_PREFIX Command to be pre-pended to all services that are
: going to run.

GLOBAL_POSTFIX Global option that will be supplied to all the
: services that are going to run.

5.15.3 ARM - Availability

As mentioned before, one of the features provided by ARM is starting services on demand. Consider the example of one service "client" that wants to connect to another service a "server". The "client" will ask ARM to run the "server". ARM starts the "server". The "server" starts listening to incoming connections. The "client" will establish a connection with the "server". And then, they will start to communicate together. One problem with that scheme is that it's slow! The "client" service wants to communicate with the "server" service at once and is not willing wait for it to be started and listening to incoming connections before serving its request. One solution for that problem will be that ARM starts all services as default services. That solution will solve the problem, yet, it's not quite practical, for some services that are going to be started can never be used or are going to be used after a relatively long time. The approach followed by ARM to solve this problem is as follows:

- For each service having a PORT field in the configuration file and that is not one of the default services (a service that accepts incoming connections from clients), ARM creates listening sockets for all addresses associated with that service.
- The "client" will immediately establish a connection with the "server".
- ARM — pretending to be the "server" — will listen on the respective port and notice the incoming connection from the "client" (but not accept it), instead
- Once there is an incoming connection, ARM will start the "server", passing on the listen sockets (now, the service is started and can do its work).
- Other client services now can directly connect directly to the "server".

5.15.4 Reliability

One of the features provided by ARM, is the automatic restart of crashed services. ARM needs to know which of the running services died. Function "gnunet-service-arm.c/maint_child_death()" is responsible for that. The function is scheduled to run upon receiving a SIGCHLD signal. The function, then, iterates ARM's list of services running and monitors which service has died (crashed). For all crashing services, ARM restarts them. Now, considering the case of a service having a serious problem causing it to crash each time it's started by ARM. If ARM keeps blindly restarting such a service, we are going to have the pattern: start-crash-restart-crash-restart-crash and so forth!! Which is of course not practical. For that reason, ARM schedules the service to be restarted after waiting for some delay that grows exponentially with each crash/restart of that service. To clarify the idea, considering the following example:

- Service S crashed.
- ARM receives the SIGCHLD and inspects its list of services to find the dead one(s).
- ARM finds S dead and schedules it for restarting after "backoff" time which is initially set to 1ms. ARM will double the backoff time correspondent to S (now backoff(S) = 2ms)
- Because there is a severe problem with S, it crashed again.
- Again ARM receives the SIGCHLD and detects that it's S again that's crashed. ARM schedules it for restarting but after its new backoff time (which became 2ms), and doubles its backoff time (now backoff(S) = 4).

- and so on, until backoff(S) reaches a certain threshold (`EXPONENTIAL_BACKOFF_THRESHOLD` is set to half an hour), after reaching it, backoff(S) will remain half an hour, hence ARM won't be busy for a lot of time trying to restart a problematic service.

5.16 TRANSPORT Subsystem

This chapter documents how the GNUnet transport subsystem works. The GNUnet transport subsystem consists of three main components: the transport API (the interface used by the rest of the system to access the transport service), the transport service itself (most of the interesting functions, such as choosing transports, happens here) and the transport plugins. A transport plugin is a concrete implementation for how two GNUnet peers communicate; many plugins exist, for example for communication via TCP, UDP, HTTP, HTTPS and others. Finally, the transport subsystem uses supporting code, especially the NAT/UPnP library to help with tasks such as NAT traversal.

Key tasks of the transport service include:

- Create our HELLO message, notify clients and neighbours if our HELLO changes (using NAT library as necessary)
- Validate HELLOs from other peers (send PING), allow other peers to validate our HELLO's addresses (send PONG)
- Upon request, establish connections to other peers (using address selection from ATS subsystem) and maintain them (again using PINGs and PONGs) as long as desired
- Accept incoming connections, give ATS service the opportunity to switch communication channels
- Notify clients about peers that have connected to us or that have been disconnected from us
- If a (stateful) connection goes down unexpectedly (without explicit DISCONNECT), quickly attempt to recover (without notifying clients) but do notify clients quickly if reconnecting fails
- Send (payload) messages arriving from clients to other peers via transport plugins and receive messages from other peers, forwarding those to clients
- Enforce inbound traffic limits (using flow-control if it is applicable); outbound traffic limits are enforced by CORE, not by us (!)
- Enforce restrictions on P2P connection as specified by the blacklist configuration and blacklisting clients

Note that the term `"clients"` in the list above really refers to the GNUnet-CORE service, as CORE is typically the only client of the transport service.

5.16.1 Address validation protocol

This section documents how the GNUnet transport service validates connections with other peers. It is a high-level description of the protocol necessary to understand the details of the implementation. It should be noted that when we talk about PING and PONG messages in this section, we refer to transport-level PING and PONG messages, which are different from core-level PING and PONG messages (both in implementation and function).

The goal of transport-level address validation is to minimize the chances of a successful man-in-the-middle attack against GNUnet peers on the transport level. Such an attack would not allow the adversary to decrypt the P2P transmissions, but a successful attacker could at least measure traffic volumes and latencies (raising the adversaries capablities by those of a global passive adversary in the worst case). The scenarios we are concerned about is an attacker, Mallory, giving a `HELLO` to Alice that claims to be for Bob, but contains Mallory's IP address instead of Bobs (for some transport). Mallory would then forward the traffic to Bob (by initiating a connection to Bob and claiming to be Alice). As a further complication, the scheme has to work even if say Alice is behind a NAT without traversal support and hence has no address of her own (and thus Alice must always initiate the connection to Bob).

An additional constraint is that `HELLO` messages do not contain a cryptographic signature since other peers must be able to edit (i.e. remove) addresses from the `HELLO` at any time (this was not true in GNUnet 0.8.x). A basic **assumption** is that each peer knows the set of possible network addresses that it **might** be reachable under (so for example, the external IP address of the NAT plus the LAN address(es) with the respective ports).

The solution is the following. If Alice wants to validate that a given address for Bob is valid (i.e. is actually established **directly** with the intended target), she sends a PING message over that connection to Bob. Note that in this case, Alice initiated the connection so only Alice knows which address was used for sure (Alice may be behind NAT, so whatever address Bob sees may not be an address Alice knows she has). Bob checks that the address given in the `PING` is actually one of Bob's addresses (ie: does not belong to Mallory), and if it is, sends back a `PONG` (with a signature that says that Bob owns/uses the address from the `PING`). Alice checks the signature and is happy if it is valid and the address in the `PONG` is the address Alice used. This is similar to the 0.8.x protocol where the `HELLO` contained a signature from Bob for each address used by Bob. Here, the purpose code for the signature is `GNUNET_SIGNATURE_PURPOSE_TRANSPORT_PONG_OWN`. After this, Alice will remember Bob's address and consider the address valid for a while (12h in the current implementation). Note that after this exchange, Alice only considers Bob's address to be valid, the connection itself is not considered 'established'. In particular, Alice may have many addresses for Bob that Alice considers valid.

The `PONG` message is protected with a nonce/challenge against replay attacks[6] and uses an expiration time for the signature (but those are almost implementation details).

5.17 NAT library

The goal of the GNUnet NAT library is to provide a general-purpose API for NAT traversal **without** third-party support. So protocols that involve contacting a third peer to help establish a connection between two peers are outside of the scope of this API. That does not mean that GNUnet doesn't support involving a third peer (we can do this with the distance-vector transport or using application-level protocols), it just means that the NAT API is not concerned with this possibility. The API is written so that it will work for IPv6-NAT in the future as well as current IPv4-NAT. Furthermore, the NAT API is always used, even for peers that are not behind NAT — in that case, the mapping provided is simply the identity.

[6] replay (`http://en.wikipedia.org/wiki/Replay_attack`)

NAT traversal is initiated by calling `GNUNET_NAT_register`. Given a set of addresses that the peer has locally bound to (TCP or UDP), the NAT library will return (via callback) a (possibly longer) list of addresses the peer **might** be reachable under. Internally, depending on the configuration, the NAT library will try to punch a hole (using UPnP) or just "know" that the NAT was manually punched and generate the respective external IP address (the one that should be globally visible) based on the given information.

The NAT library also supports ICMP-based NAT traversal. Here, the other peer can request connection-reversal by this peer (in this special case, the peer is even allowed to configure a port number of zero). If the NAT library detects a connection-reversal request, it returns the respective target address to the client as well. It should be noted that connection-reversal is currently only intended for TCP, so other plugins **must** pass `NULL` for the reversal callback. Naturally, the NAT library also supports requesting connection reversal from a remote peer (`GNUNET_NAT_run_client`).

Once initialized, the NAT handle can be used to test if a given address is possibly a valid address for this peer (`GNUNET_NAT_test_address`). This is used for validating our addresses when generating PONGs.

Finally, the NAT library contains an API to test if our NAT configuration is correct. Using `GNUNET_NAT_test_start` **before** binding to the respective port, the NAT library can be used to test if the configuration works. The test function act as a local client, initialize the NAT traversal and then contact a `gnunet-nat-server` (running by default on `gnunet.org`) and ask for a connection to be established. This way, it is easy to test if the current NAT configuration is valid.

5.18 Distance-Vector plugin

The Distance Vector (DV) transport is a transport mechanism that allows peers to act as relays for each other, thereby connecting peers that would otherwise be unable to connect. This gives a larger connection set to applications that may work better with more peers to choose from (for example, File Sharing and/or DHT).

The Distance Vector transport essentially has two functions. The first is "gossiping" connection information about more distant peers to directly connected peers. The second is taking messages intended for non-directly connected peers and encapsulating them in a DV wrapper that contains the required information for routing the message through forwarding peers. Via gossiping, optimal routes through the known DV neighborhood are discovered and utilized and the message encapsulation provides some benefits in addition to simply getting the message from the correct source to the proper destination.

The gossiping function of DV provides an up to date routing table of peers that are available up to some number of hops. We call this a fisheye view of the network (like a fish, nearby objects are known while more distant ones unknown). Gossip messages are sent only to directly connected peers, but they are sent about other knowns peers within the "fisheye distance". Whenever two peers connect, they immediately gossip to each other about their appropriate other neighbors. They also gossip about the newly connected peer to previously connected neighbors. In order to keep the routing tables up to date, disconnect notifications are propogated as gossip as well (because disconnects may not be sent/received, timeouts are also used remove stagnant routing table entries).

Routing of messages via DV is straightforward. When the DV transport is notified of a message destined for a non-direct neighbor, the appropriate forwarding peer is selected, and the base message is encapsulated in a DV message which contains information about the initial peer and the intended recipient. At each forwarding hop, the initial peer is validated (the forwarding peer ensures that it has the initial peer in its neighborhood, otherwise the message is dropped). Next the base message is re-encapsulated in a new DV message for the next hop in the forwarding chain (or delivered to the current peer, if it has arrived at the destination).

Assume a three peer network with peers Alice, Bob and Carol. Assume that

```
Alice <-> Bob and Bob <-> Carol
```

are direct (e.g. over TCP or UDP transports) connections, but that Alice cannot directly connect to Carol. This may be the case due to NAT or firewall restrictions, or perhaps based on one of the peers respective configurations. If the Distance Vector transport is enabled on all three peers, it will automatically discover (from the gossip protocol) that Alice and Carol can connect via Bob and provide a "virtual" Alice <-> Carol connection. Routing between Alice and Carol happens as follows; Alice creates a message destined for Carol and notifies the DV transport about it. The DV transport at Alice looks up Carol in the routing table and finds that the message must be sent through Bob for Carol. The message is encapsulated setting Alice as the initiator and Carol as the destination and sent to Bob. Bob receives the messages, verifies that both Alice and Carol are known to Bob, and re-wraps the message in a new DV message for Carol. The DV transport at Carol receives this message, unwraps the original message, and delivers it to Carol as though it came directly from Alice.

5.19 SMTP plugin

This section describes the new SMTP transport plugin for GNUnet as it exists in the 0.7.x and 0.8.x branch. SMTP support is currently not available in GNUnet 0.9.x. This page also describes the transport layer abstraction (as it existed in 0.7.x and 0.8.x) in more detail and gives some benchmarking results. The performance results presented are quite old and maybe outdated at this point.

- Why use SMTP for a peer-to-peer transport?
- SMTPHow does it work?
- How do I configure my peer?
- How do I test if it works?
- How fast is it?
- Is there any additional documentation?

5.19.1 Why use SMTP for a peer-to-peer transport?

There are many reasons why one would not want to use SMTP:

- SMTP is using more bandwidth than TCP, UDP or HTTP
- SMTP has a much higher latency.
- SMTP requires significantly more computation (encoding and decoding time) for the peers.

- SMTP is significantly more complicated to configure.
- SMTP may be abused by tricking GNUnet into sending mail to non-participating third parties.

So why would anybody want to use SMTP?

- SMTP can be used to contact peers behind NAT boxes (in virtual private networks).
- SMTP can be used to circumvent policies that limit or prohibit peer-to-peer traffic by masking as "legitimate" traffic.
- SMTP uses E-mail addresses which are independent of a specific IP, which can be useful to address peers that use dynamic IP addresses.
- SMTP can be used to initiate a connection (e.g. initial address exchange) and peers can then negotiate the use of a more efficient protocol (e.g. TCP) for the actual communication.

In summary, SMTP can for example be used to send a message to a peer behind a NAT box that has a dynamic IP to tell the peer to establish a TCP connection to a peer outside of the private network. Even an extraordinary overhead for this first message would be irrelevant in this type of situation.

5.19.2 How does it work?

When a GNUnet peer needs to send a message to another GNUnet peer that has advertised (only) an SMTP transport address, GNUnet base64-encodes the message and sends it in an E-mail to the advertised address. The advertisement contains a filter which is placed in the E-mail header, such that the receiving host can filter the tagged E-mails and forward it to the GNUnet peer process. The filter can be specified individually by each peer and be changed over time. This makes it impossible to censor GNUnet E-mail messages by searching for a generic filter.

5.19.3 How do I configure my peer?

First, you need to configure `procmail` to filter your inbound E-mail for GNUnet traffic. The GNUnet messages must be delivered into a pipe, for example `/tmp/gnunet.smtp`. You also need to define a filter that is used by `procmail` to detect GNUnet messages. You are free to choose whichever filter you like, but you should make sure that it does not occur in your other E-mail. In our example, we will use `X-mailer: GNUnet`. The `~/.procmailrc` configuration file then looks like this:

```
:0:
* ^X-mailer: GNUnet
/tmp/gnunet.smtp
# where do you want your other e-mail delivered to
# (default: /var/spool/mail/)
:0: /var/spool/mail/
```

After adding this file, first make sure that your regular E-mail still works (e.g. by sending an E-mail to yourself). Then edit the GNUnet configuration. In the section `SMTP` you need to specify your E-mail address under `EMAIL`, your mail server (for outgoing mail) under `SERVER`, the filter (X-mailer: GNUnet in the example) under `FILTER` and the name of the pipe under `PIPE`. The completed section could then look like this:

```
EMAIL = me@mail.gnu.org MTU = 65000 SERVER = mail.gnu.org:25 FILTER =
```

```
"X-mailer: GNUnet" PIPE = /tmp/gnunet.smtp
```

Finally, you need to add `smtp` to the list of `TRANSPORTS` in the `GNUNETD` section. GNUnet peers will use the E-mail address that you specified to contact your peer until the advertisement times out. Thus, if you are not sure if everything works properly or if you are not planning to be online for a long time, you may want to configure this timeout to be short, e.g. just one hour. For this, set `HELLOEXPIRES` to `1` in the `GNUNETD` section.

This should be it, but you may probably want to test it first.

5.19.4 How do I test if it works?

Any transport can be subjected to some rudimentary tests using the `gnunet-transport-check` tool. The tool sends a message to the local node via the transport and checks that a valid message is received. While this test does not involve other peers and can not check if firewalls or other network obstacles prohibit proper operation, this is a great testcase for the SMTP transport since it tests pretty much nearly all of the functionality.

`gnunet-transport-check` should only be used without running `gnunetd` at the same time. By default, `gnunet-transport-check` tests all transports that are specified in the configuration file. But you can specifically test SMTP by giving the option `--transport=smtp`.

Note that this test always checks if a transport can receive and send. While you can configure most transports to only receive or only send messages, this test will only work if you have configured the transport to send and receive messages.

5.19.5 How fast is it?

We have measured the performance of the UDP, TCP and SMTP transport layer directly and when used from an application using the GNUnet core. Measureing just the transport layer gives the better view of the actual overhead of the protocol, whereas evaluating the transport from the application puts the overhead into perspective from a practical point of view.

The loopback measurements of the SMTP transport were performed on three different machines spanning a range of modern SMTP configurations. We used a PIII-800 running RedHat 7.3 with the Purdue Computer Science configuration which includes filters for spam. We also used a Xenon 2 GHZ with a vanilla RedHat 8.0 sendmail configuration. Furthermore, we used qmail on a PIII-1000 running Sorcerer GNU Linux (SGL). The numbers for UDP and TCP are provided using the SGL configuration. The qmail benchmark uses qmail's internal filtering whereas the sendmail benchmarks relies on procmail to filter and deliver the mail. We used the transport layer to send a message of b bytes (excluding transport protocol headers) directly to the local machine. This way, network latency and packet loss on the wire have no impact on the timings. n messages were sent sequentially over the transport layer, sending message i+1 after the i-th message was received. All messages were sent over the same connection and the time to establish the connection was not taken into account since this overhead is miniscule in practice — as long as a connection is used for a significant number of messages.

Transport	UDP	TCP	SMTP (Purdue sendmail)	SMTP (RH 8.0)	SMTP (SGL qmail)
11 bytes	31 ms	55 ms	781 s	77 s	24 s
407 bytes	37 ms	62 ms	789 s	78 s	25 s
1,221 bytes	46 ms	73 ms	804 s	78 s	25 s

The benchmarks show that UDP and TCP are, as expected, both significantly faster compared with any of the SMTP services. Among the SMTP implementations, there can be significant differences depending on the SMTP configuration. Filtering with an external tool like procmail that needs to re-parse its configuration for each mail can be very expensive. Applying spam filters can also significantly impact the performance of the underlying SMTP implementation. The microbenchmark shows that SMTP can be a viable solution for initiating peer-to-peer sessions: a couple of seconds to connect to a peer are probably not even going to be noticed by users. The next benchmark measures the possible throughput for a transport. Throughput can be measured by sending multiple messages in parallel and measuring packet loss. Note that not only UDP but also the TCP transport can actually loose messages since the TCP implementation drops messages if the `write` to the socket would block. While the SMTP protocol never drops messages itself, it is often so slow that only a fraction of the messages can be sent and received in the given time-bounds. For this benchmark we report the message loss after allowing t time for sending m messages. If messages were not sent (or received) after an overall timeout of t, they were considered lost. The benchmark was performed using two Xeon 2 GHZ machines running RedHat 8.0 with sendmail. The machines were connected with a direct 100 MBit ethernet connection. Figures udp1200, tcp1200 and smtp-MTUs show that the throughput for messages of size 1,200 octects is 2,343 kbps, 3,310 kbps and 6 kbps for UDP, TCP and SMTP respectively. The high per-message overhead of SMTP can be improved by increasing the MTU, for example, an MTU of 12,000 octets improves the throughput to 13 kbps as figure smtp-MTUs shows. Our research paper) has some more details on the benchmarking results.

5.20 Bluetooth plugin

This page describes the new Bluetooth transport plugin for GNUnet. The plugin is still in the testing stage so don't expect it to work perfectly. If you have any questions or problems just post them here or ask on the IRC channel.

- What do I need to use the Bluetooth plugin transport?
- BluetoothHow does it work?
- What possible errors should I be aware of?
- How do I configure my peer?
- How can I test it?

5.20.1 What do I need to use the Bluetooth plugin transport?

If you are a GNU/Linux user and you want to use the Bluetooth transport plugin you should install the `BlueZ development libraries` (if they aren't already installed). For instructions about how to install the libraries you should check out the BlueZ site (http://www.bluez.org (`http://www.bluez.org/`)). If you don't know if you have the necesarry libraries, don't worry, just run the GNUnet configure script and you will be

able to see a notification at the end which will warn you if you don't have the necessary libraries.

If you are a Windows user you should have installed the *MinGW/MSys2* with the latest updates (especially the *ws2bth* header). If this is your first build of GNUnet on Windows you should check out the SBuild repository. It will semi-automatically assembles a *MinGW/MSys2* installation with a lot of extra packages which are needed for the GNUnet build. So this will ease your work! Finally you just have to be sure that you have the correct drivers for your Bluetooth device installed and that your device is on and in a discoverable mode. The Windows Bluetooth Stack supports only the RFCOMM protocol so we cannot turn on your device programatically!

5.20.2 How does it work2?

The Bluetooth transport plugin uses virtually the same code as the WLAN plugin and only the helper binary is different. The helper takes a single argument, which represents the interface name and is specified in the configuration file. Here are the basic steps that are followed by the helper binary used on GNU/Linux:

- it verifies if the name corresponds to a Bluetooth interface name
- it verifies if the iterface is up (if it is not, it tries to bring it up)
- it tries to enable the page and inquiry scan in order to make the device discoverable and to accept incoming connection requests *The above operations require root access so you should start the transport plugin with root privileges.*
- it finds an available port number and registers a SDP service which will be used to find out on which port number is the server listening on and switch the socket in listening mode
- it sends a HELLO message with its address
- finally it forwards traffic from the reading sockets to the STDOUT and from the STDIN to the writing socket

Once in a while the device will make an inquiry scan to discover the nearby devices and it will send them randomly HELLO messages for peer discovery.

5.20.3 What possible errors should I be aware of?

This section is dedicated for GNU/Linux users

Well there are many ways in which things could go wrong but I will try to present some tools that you could use to debug and some scenarios.

- `bluetoothd -n -d` : use this command to enable logging in the foreground and to print the logging messages
- `hciconfig`: can be used to configure the Bluetooth devices. If you run it without any arguments it will print information about the state of the interfaces. So if you receive an error that the device couldn't be brought up you should try to bring it manually and to see if it works (use `hciconfig -a hciX up`). If you can't and the Bluetooth address has the form 00:00:00:00:00:00 it means that there is something wrong with the D-Bus daemon or with the Bluetooth daemon. Use `bluetoothd` tool to see the logs
- `sdptool` can be used to control and interogate SDP servers. If you encounter problems regarding the SDP server (like the SDP server is down) you should check out if the

D-Bus daemon is running correctly and to see if the Bluetooth daemon started correctly(use `bluetoothd` tool). Also, sometimes the SDP service could work but somehow the device couldn't register his service. Use `sdptool browse [dev-address]` to see if the service is registered. There should be a service with the name of the interface and GNUnet as provider.

- `hcitool` : another useful tool which can be used to configure the device and to send some particular commands to it.
- `hcidump` : could be used for low level debugging

5.20.4 How do I configure my peer2?

On GNU/Linux, you just have to be sure that the interface name corresponds to the one that you want to use. Use the `hciconfig` tool to check that. By default it is set to hci0 but you can change it.

A basic configuration looks like this:

```
[transport-bluetooth]
# Name of the interface (typically hciX)
INTERFACE = hci0
# Real hardware, no testing
TESTMODE = 0 TESTING_IGNORE_KEYS = ACCEPT_FROM;
```

In order to use the Bluetooth transport plugin when the transport service is started, you must add the plugin name to the default transport service plugins list. For example:

```
[transport] ...  PLUGINS = dns bluetooth ...
```

If you want to use only the Bluetooth plugin set *PLUGINS = bluetooth*

On Windows, you cannot specify which device to use. The only thing that you should do is to add *bluetooth* on the plugins list of the transport service.

5.20.5 How can I test it?

If you have two Bluetooth devices on the same machine and you are using GNU/Linux you must:

- create two different file configuration (one which will use the first interface (*hci0*) and the other which will use the second interface (*hci1*)). Let's name them *peer1.conf* and *peer2.conf*.
- run *gnunet-peerinfo -c peerX.conf -s* in order to generate the peers private keys. The **X** must be replace with 1 or 2.
- run *gnunet-arm -c peerX.conf -s -i=transport* in order to start the transport service. (Make sure that you have "bluetooth" on the transport plugins list if the Bluetooth transport service doesn't start.)
- run *gnunet-peerinfo -c peer1.conf -s* to get the first peer's ID. If you already know your peer ID (you saved it from the first command), this can be skipped.
- run *gnunet-transport -c peer2.conf -p=PEER1_ID -s* to start sending data for benchmarking to the other peer.

This scenario will try to connect the second peer to the first one and then start sending data for benchmarking.

On Windows you cannot test the plugin functionality using two Bluetooth devices from the same machine because after you install the drivers there will occur some conflicts between the Bluetooth stacks. (At least that is what happend on my machine : I wasn't able to use the Bluesoleil stack and the WINDCOMM one in the same time).

If you have two different machines and your configuration files are good you can use the same scenario presented on the begining of this section.

Another way to test the plugin functionality is to create your own application which will use the GNUnet framework with the Bluetooth transport service.

5.20.6 The implementation of the Bluetooth transport plugin

This page describes the implementation of the Bluetooth transport plugin.

First I want to remind you that the Bluetooth transport plugin uses virtually the same code as the WLAN plugin and only the helper binary is different. Also the scope of the helper binary from the Bluetooth transport plugin is the same as the one used for the wlan transport plugin: it acceses the interface and then it forwards traffic in both directions between the Bluetooth interface and stdin/stdout of the process involved.

The Bluetooth plugin transport could be used both on GNU/Linux and Windows platforms.

- Linux functionality
- Windows functionality
- Pending Features

5.20.6.1 Linux functionality

In order to implement the plugin functionality on GNU/Linux I used the BlueZ stack. For the communication with the other devices I used the RFCOMM protocol. Also I used the HCI protocol to gain some control over the device. The helper binary takes a single argument (the name of the Bluetooth interface) and is separated in two stages:

5.20.6.2 THE INITIALIZATION

- first, it checks if we have root privilegies (*Remember that we need to have root privilegies in order to be able to bring the interface up if it is down or to change its state.*).
- second, it verifies if the interface with the given name exists.

  If the interface with that name exists and it is a Bluetooth interface:
- it creates a RFCOMM socket which will be used for listening and call the *open_device* method

  On the *open_device* method:
  - creates a HCI socket used to send control events to the the device
  - searches for the device ID using the interface name
  - saves the device MAC address
  - checks if the interface is down and tries to bring it UP
  - checks if the interface is in discoverable mode and tries to make it discoverable
  - closes the HCI socket and binds the RFCOMM one
  - switches the RFCOMM socket in listening mode

    - registers the SDP service (the service will be used by the other devices to get the port on which this device is listening on)
- drops the root privilegies

  If the interface is not a Bluetooth interface the helper exits with a suitable error

5.20.6.3 THE LOOP

The helper binary uses a list where it saves all the connected neighbour devices (*neighbours.devices*) and two buffers (*write_pout* and *write_std*). The first message which is send is a control message with the device's MAC address in order to announce the peer presence to the neighbours. Here are a short description of what happens in the main loop:

- Every time when it receives something from the STDIN it processes the data and saves the message in the first buffer (*write_pout*). When it has something in the buffer, it gets the destination address from the buffer, searches the destination address in the list (if there is no connection with that device, it creates a new one and saves it to the list) and sends the message.
- Every time when it receives something on the listening socket it accepts the connection and saves the socket on a list with the reading sockets.
- Every time when it receives something from a reading socket it parses the message, verifies the CRC and saves it in the *write_std* buffer in order to be sent later to the STDOUT.

So in the main loop we use the select function to wait until one of the file descriptor saved in one of the two file descriptors sets used is ready to use. The first set (*rfds*) represents the reading set and it could contain the list with the reading sockets, the STDIN file descriptor or the listening socket. The second set (*wfds*) is the writing set and it could contain the sending socket or the STDOUT file descriptor. After the select function returns, we check which file descriptor is ready to use and we do what is supposed to do on that kind of event. *For example:* if it is the listening socket then we accept a new connection and save the socket in the reading list; if it is the STDOUT file descriptor, then we write to STDOUT the message from the *write_std* buffer.

To find out on which port a device is listening on we connect to the local SDP server and searche the registered service for that device.

You should be aware of the fact that if the device fails to connect to another one when trying to send a message it will attempt one more time. If it fails again, then it skips the message. Also you should know that the transport Bluetooth plugin has support for **broadcast messages***.*

5.20.6.4 Details about the broadcast implementation

First I want to point out that the broadcast functionality for the CONTROL messages is not implemented in a conventional way. Since the inquiry scan time is too big and it will take some time to send a message to all the discoverable devices I decided to tackle the problem in a different way. Here is how I did it:

- If it is the first time when I have to broadcast a message I make an inquiry scan and save all the devices' addresses to a vector.

- After the inquiry scan ends I take the first address from the list and I try to connect to it. If it fails, I try to connect to the next one. If it succeeds, I save the socket to a list and send the message to the device.
- When I have to broadcast another message, first I search on the list for a new device which I'm not connected to. If there is no new device on the list I go to the beginning of the list and send the message to the old devices. After 5 cycles I make a new inquiry scan to check out if there are new discoverable devices and save them to the list. If there are no new discoverable devices I reset the cycling counter and go again through the old list and send messages to the devices saved in it.

Therefore:

- every time when I have a broadcast message I look up on the list for a new device and send the message to it
- if I reached the end of the list for 5 times and I'm connected to all the devices from the list I make a new inquiry scan. *The number of the list's cycles after an inquiry scan could be increased by redefining the MAX_LOOPS variable*
- when there are no new devices I send messages to the old ones.

Doing so, the broadcast control messages will reach the devices but with delay.

NOTICE: When I have to send a message to a certain device first I check on the broadcast list to see if we are connected to that device. If not we try to connect to it and in case of success we save the address and the socket on the list. If we are already connected to that device we simply use the socket.

5.20.6.5 Windows functionality

For Windows I decided to use the Microsoft Bluetooth stack which has the advantage of coming standard from Windows XP SP2. The main disadvantage is that it only supports the RFCOMM protocol so we will not be able to have a low level control over the Bluetooth device. Therefore it is the user responsability to check if the device is up and in the discoverable mode. Also there are no tools which could be used for debugging in order to read the data coming from and going to a Bluetooth device, which obviously hindered my work. Another thing that slowed down the implementation of the plugin (besides that I wasn't too accomodated with the win32 API) was that there were some bugs on MinGW regarding the Bluetooth. Now they are solved but you should keep in mind that you should have the latest updates (especially the *ws2bth* header).

Besides the fact that it uses the Windows Sockets, the Windows implemenation follows the same principles as the GNU/Linux one:

- It has a initalization part where it initializes the Windows Sockets, creates a RFCOMM socket which will be binded and switched to the listening mode and registers a SDP service. In the Microsoft Bluetooth API there are two ways to work with the SDP:
  - an easy way which works with very simple service records
  - a hard way which is useful when you need to update or to delete the record

Since I only needed the SDP service to find out on which port the device is listening on and that did not change, I decided to use the easy way. In order to register the service I used the *WSASetService* function and I generated the *Universally Unique Identifier* with the *guidgen.exe* Windows's tool.

In the loop section the only difference from the GNU/Linux implementation is that I used the `GNUNET_NETWORK` library for functions like *accept*, *bind*, *connect* or *select*. I decided to use the `GNUNET_NETWORK` library because I also needed to interact with the STDIN and STDOUT handles and on Windows the select function is only defined for sockets, and it will not work for arbitrary file handles.

Another difference between GNU/Linux and Windows implementation is that in GNU/Linux, the Bluetooth address is represented in 48 bits while in Windows is represented in 64 bits. Therefore I had to do some changes on *plugin_transport_wlan* header.

Also, currently on Windows the Bluetooth plugin doesn't have support for broadcast messages. When it receives a broadcast message it will skip it.

5.20.6.6 Pending features

- Implement the broadcast functionality on Windows *(currently working on)*
- Implement a testcase for the helper : *The testcase consists of a program which emaluates the plugin and uses the helper. It will simulate connections, disconnections and data transfers.*

If you have a new idea about a feature of the plugin or suggestions about how I could improve the implementation you are welcome to comment or to contact me.

5.21 WLAN plugin

This section documents how the wlan transport plugin works. Parts which are not implemented yet or could be better implemented are described at the end.

5.22 ATS Subsystem

ATS stands for "automatic transport selection", and the function of ATS in GNUnet is to decide on which address (and thus transport plugin) should be used for two peers to communicate, and what bandwidth limits should be imposed on such an individual connection. To help ATS make an informed decision, higher-level services inform the ATS service about their requirements and the quality of the service rendered. The ATS service also interacts with the transport service to be appraised of working addresses and to communicate its resource allocation decisions. Finally, the ATS service's operation can be observed using a monitoring API.

The main logic of the ATS service only collects the available addresses, their performance characteristics and the applications requirements, but does not make the actual allocation decision. This last critical step is left to an ATS plugin, as we have implemented (currently three) different allocation strategies which differ significantly in their performance and maturity, and it is still unclear if any particular plugin is generally superior.

5.23 CORE Subsystem

The CORE subsystem in GNUnet is responsible for securing link-layer communications between nodes in the GNUnet overlay network. CORE builds on the TRANSPORT subsystem

which provides for the actual, insecure, unreliable link-layer communication (for example, via UDP or WLAN), and then adds fundamental security to the connections:

- confidentiality with so-called perfect forward secrecy; we use ECDHE[7] powered by Curve25519[8] for the key exchange and then use symmetric encryption, encrypting with both AES-256[9] and Twofish[10]
- authentication (http://en.wikipedia.org/wiki/Authentication) is achieved by signing the ephemeral keys using Ed25519[11], a deterministic variant of ECDSA[12]
- integrity protection (using SHA-512[13] to do encrypt-then-MAC[14])
- Replay[15] protection (using nonces, timestamps, challenge-response, message counters and ephemeral keys)
- liveness (keep-alive messages, timeout)

5.23.1 Limitations

CORE does not perform routing (http://en.wikipedia.org/wiki/Routing); using CORE it is only possible to communicate with peers that happen to already be "directly" connected with each other. CORE also does not have an API to allow applications to establish such "direct" connections — for this, applications can ask TRANSPORT, but TRANSPORT might not be able to establish a "direct" connection. The TOPOLOGY subsystem is responsible for trying to keep a few "direct" connections open at all times. Applications that need to talk to particular peers should use the CADET subsystem, as it can establish arbitrary "indirect" connections.

Because CORE does not perform routing, CORE must only be used directly by applications that either perform their own routing logic (such as anonymous file-sharing) or that do not require routing, for example because they are based on flooding the network. CORE communication is unreliable and delivery is possibly out-of-order. Applications that require reliable communication should use the CADET service. Each application can only queue one message per target peer with the CORE service at any time; messages cannot be larger than approximately 63 kilobytes. If messages are small, CORE may group multiple messages (possibly from different applications) prior to encryption. If permitted by the application (using the cork (http://baus.net/on-tcp_cork/) option), CORE may delay transmissions to facilitate grouping of multiple small messages. If cork is not enabled, CORE will transmit the message as soon as TRANSPORT allows it (TRANSPORT is responsible for limiting bandwidth and congestion control). CORE does not allow flow control; applications are expected to process messages at line-speed. If flow control is needed, applications should use the CADET service.

---

[7] Elliptic-curve Diffie—Hellman (http://en.wikipedia.org/wiki/Elliptic_curve_Diffie%E2%80%93Hellman)
[8] Curve25519 (http://cr.yp.to/ecdh.html)
[9] AES-256 (http://en.wikipedia.org/wiki/Rijndael)
[10] Twofish (http://en.wikipedia.org/wiki/Twofish)
[11] Ed25519 (http://ed25519.cr.yp.to/)
[12] ECDSA (http://en.wikipedia.org/wiki/ECDSA)
[13] SHA-512 (http://en.wikipedia.org/wiki/SHA-2)
[14] encrypt-then-MAC (http://en.wikipedia.org/wiki/Authenticated_encryption)
[15] replay (http://en.wikipedia.org/wiki/Replay_attack)

5.23.2 When is a peer "connected"?

In addition to the security features mentioned above, CORE also provides one additional key feature to applications using it, and that is a limited form of protocol-compatibility checking. CORE distinguishes between TRANSPORT-level connections (which enable communication with other peers) and application-level connections. Applications using the CORE API will (typically) learn about application-level connections from CORE, and not about TRANSPORT-level connections. When a typical application uses CORE, it will specify a set of message types (from `gnunet_protocols.h`) that it understands. CORE will then notify the application about connections it has with other peers if and only if those applications registered an intersecting set of message types with their CORE service. Thus, it is quite possible that CORE only exposes a subset of the established direct connections to a particular application — and different applications running above CORE might see different sets of connections at the same time.

A special case are applications that do not register a handler for any message type. CORE assumes that these applications merely want to monitor connections (or "all" messages via other callbacks) and will notify those applications about all connections. This is used, for example, by the `gnunet-core` command-line tool to display the active connections. Note that it is also possible that the TRANSPORT service has more active connections than the CORE service, as the CORE service first has to perform a key exchange with connecting peers before exchanging information about supported message types and notifying applications about the new connection.

5.23.3 libgnunetcore

The CORE API (defined in `gnunet_core_service.h`) is the basic messaging API used by P2P applications built using GNUnet. It provides applications the ability to send and receive encrypted messages to the peer's "directly" connected neighbours.

As CORE connections are generally "direct" connections, applications must not assume that they can connect to arbitrary peers this way, as "direct" connections may not always be possible. Applications using CORE are notified about which peers are connected. Creating new "direct" connections must be done using the TRANSPORT API.

The CORE API provides unreliable, out-of-order delivery. While the implementation tries to ensure timely, in-order delivery, both message losses and reordering are not detected and must be tolerated by the application. Most important, the core will NOT perform retransmission if messages could not be delivered.

Note that CORE allows applications to queue one message per connected peer. The rate at which each connection operates is influenced by the preferences expressed by local application as well as restrictions imposed by the other peer. Local applications can express their preferences for particular connections using the "performance" API of the ATS service.

Applications that require more sophisticated transmission capabilities such as TCP-like behavior, or if you intend to send messages to arbitrary remote peers, should use the CADET API.

The typical use of the CORE API is to connect to the CORE service using `GNUNET_CORE_connect`, process events from the CORE service (such as peers connecting, peers disconnecting and incoming messages) and send messages to connected peers using `GNUNET_CORE_notify_transmit_ready`. Note that applications must cancel pending transmission

requests if they receive a disconnect event for a peer that had a transmission pending; furthermore, queueing more than one transmission request per peer per application using the service is not permitted.

The CORE API also allows applications to monitor all communications of the peer prior to encryption (for outgoing messages) or after decryption (for incoming messages). This can be useful for debugging, diagnostics or to establish the presence of cover traffic (for anonymity). As monitoring applications are often not interested in the payload, the monitoring callbacks can be configured to only provide the message headers (including the message type and size) instead of copying the full data stream to the monitoring client.

The init callback of the `GNUNET_CORE_connect` function is called with the hash of the public key of the peer. This public key is used to identify the peer globally in the GNUnet network. Applications are encouraged to check that the provided hash matches the hash that they are using (as theoretically the application may be using a different configuration file with a different private key, which would result in hard to find bugs).

As with most service APIs, the CORE API isolates applications from crashes of the CORE service. If the CORE service crashes, the application will see disconnect events for all existing connections. Once the connections are re-established, the applications will be receive matching connect events.

5.23.4 The CORE Client-Service Protocol

This section describes the protocol between an application using the CORE service (the client) and the CORE service process itself.

5.23.4.1 Setup2

When a client connects to the CORE service, it first sends a `InitMessage` which specifies options for the connection and a set of message type values which are supported by the application. The options bitmask specifies which events the client would like to be notified about. The options include:

GNUNET_CORE_OPTION_NOTHING No notifications
GNUNET_CORE_OPTION_STATUS_CHANGE Peers connecting and disconnecting
GNUNET_CORE_OPTION_FULL_INBOUND All inbound messages (after decryption) with full payload

GNUNET_CORE_OPTION_HDR_INBOUND Just the `MessageHeader` of all inbound messages

GNUNET_CORE_OPTION_FULL_OUTBOUND All outbound messages (prior to encryption) with full payload

GNUNET_CORE_OPTION_HDR_OUTBOUND Just the `MessageHeader` of all outbound messages

Typical applications will only monitor for connection status changes.

The CORE service responds to the `InitMessage` with an `InitReplyMessage` which contains the peer's identity. Afterwards, both CORE and the client can send messages.

5.23.4.2 Notifications

The CORE will send `ConnectNotifyMessage`s and `DisconnectNotifyMessage`s whenever peers connect or disconnect from the CORE (assuming their type maps overlap with the message types registered by the client). When the CORE receives a message that matches the set of message types specified during the `InitMessage` (or if monitoring is enabled in for inbound messages in the options), it sends a `NotifyTrafficMessage` with the peer identity of the sender and the decrypted payload. The same message format (except with `GNUNET_MESSAGE_TYPE_CORE_NOTIFY_OUTBOUND` for the message type) is used to notify clients monitoring outbound messages; here, the peer identity given is that of the receiver.

5.23.4.3 Sending

When a client wants to transmit a message, it first requests a transmission slot by sending a `SendMessageRequest` which specifies the priority, deadline and size of the message. Note that these values may be ignored by CORE. When CORE is ready for the message, it answers with a `SendMessageReady` response. The client can then transmit the payload with a `SendMessage` message. Note that the actual message size in the `SendMessage` is allowed to be smaller than the size in the original request. A client may at any time send a fresh `SendMessageRequest`, which then superceeds the previous `SendMessageRequest`, which is then no longer valid. The client can tell which `SendMessageRequest` the CORE service's `SendMessageReady` message is for as all of these messages contain a "unique" request ID (based on a counter incremented by the client for each request).

5.23.5 The CORE Peer-to-Peer Protocol

5.23.5.1 Creating the EphemeralKeyMessage

When the CORE service starts, each peer creates a fresh ephemeral (ECC) public-private key pair and signs the corresponding `EphemeralKeyMessage` with its long-term key (which we usually call the peer's identity; the hash of the public long term key is what results in a `struct GNUNET_PeerIdentity` in all GNUnet APIs. The ephemeral key is ONLY used for an ECDHE[16] exchange by the CORE service to establish symmetric session keys. A peer will use the same `EphemeralKeyMessage` for all peers for `REKEY_FREQUENCY`, which is usually 12 hours. After that time, it will create a fresh ephemeral key (forgetting the old one) and broadcast the new `EphemeralKeyMessage` to all connected peers, resulting in fresh symmetric session keys. Note that peers independently decide on when to discard ephemeral keys; it is not a protocol violation to discard keys more often. Ephemeral keys are also never stored to disk; restarting a peer will thus always create a fresh ephemeral key. The use of ephemeral keys is what provides forward secrecy (`http://en.wikipedia.org/wiki/Forward_secrecy`).

Just before transmission, the `EphemeralKeyMessage` is patched to reflect the current sender_status, which specifies the current state of the connection from the point of view of the sender. The possible values are:

- `KX_STATE_DOWN` Initial value, never used on the network
- `KX_STATE_KEY_SENT` We sent our ephemeral key, do not know the key of the other peer

[16] Elliptic-curve Diffie—Hellman (`http://en.wikipedia.org/wiki/Elliptic_curve_Diffie%E2%80%93Hellman`)

- `KX_STATE_KEY_RECEIVED` This peer has received a valid ephemeral key of the other peer, but we are waiting for the other peer to confirm it's authenticity (ability to decode) via challenge-response.
- `KX_STATE_UP` The connection is fully up from the point of view of the sender (now performing keep-alives)
- `KX_STATE_REKEY_SENT` The sender has initiated a rekeying operation; the other peer has so far failed to confirm a working connection using the new ephemeral key

5.23.5.2 Establishing a connection

Peers begin their interaction by sending a `EphemeralKeyMessage` to the other peer once the TRANSPORT service notifies the CORE service about the connection. A peer receiving an `EphemeralKeyMessage` with a status indicating that the sender does not have the receiver's ephemeral key, the receiver's `EphemeralKeyMessage` is sent in response. Additionally, if the receiver has not yet confirmed the authenticity of the sender, it also sends an (encrypted)`PingMessage` with a challenge (and the identity of the target) to the other peer. Peers receiving a `PingMessage` respond with an (encrypted) `PongMessage` which includes the challenge. Peers receiving a `PongMessage` check the challenge, and if it matches set the connection to `KX_STATE_UP`.

5.23.5.3 Encryption and Decryption

All functions related to the key exchange and encryption/decryption of messages can be found in `gnunet-service-core_kx.c` (except for the cryptographic primitives, which are in `util/crypto*.c`). Given the key material from ECDHE, a Key derivation function[17] is used to derive two pairs of encryption and decryption keys for AES-256 and TwoFish, as well as initialization vectors and authentication keys (for HMAC[18]). The HMAC is computed over the encrypted payload. Encrypted messages include an iv_seed and the HMAC in the header.

Each encrypted message in the CORE service includes a sequence number and a timestamp in the encrypted payload. The CORE service remembers the largest observed sequence number and a bit-mask which represents which of the previous 32 sequence numbers were already used. Messages with sequence numbers lower than the largest observed sequence number minus 32 are discarded. Messages with a timestamp that is less than `REKEY_TOLERANCE` off (5 minutes) are also discarded. This of course means that system clocks need to be reasonably synchronized for peers to be able to communicate. Additionally, as the ephemeral key changes every 12 hours, a peer would not even be able to decrypt messages older than 12 hours.

5.23.5.4 Type maps

Once an encrypted connection has been established, peers begin to exchange type maps. Type maps are used to allow the CORE service to determine which (encrypted) connections should be shown to which applications. A type map is an array of 65536 bits representing the different types of messages understood by applications using the CORE service. Each CORE service maintains this map, simply by setting the respective bit for each message

---

[17] Key derivation function (`https://en.wikipedia.org/wiki/Key_derivation_function`)

[18] HMAC (`https://en.wikipedia.org/wiki/HMAC`)

type supported by any of the applications using the CORE service. Note that bits for message types embedded in higher-level protocols (such as MESH) will not be included in these type maps.

Typically, the type map of a peer will be sparse. Thus, the CORE service attempts to compress its type map using `gzip`-style compression ("deflate") prior to transmission. However, if the compression fails to compact the map, the map may also be transmitted without compression (resulting in `GNUNET_MESSAGE_TYPE_CORE_COMPRESSED_TYPE_MAP` or `GNUNET_MESSAGE_TYPE_CORE_BINARY_TYPE_MAP` messages respectively). Upon receiving a type map, the respective CORE service notifies applications about the connection to the other peer if they support any message type indicated in the type map (or no message type at all). If the CORE service experience a connect or disconnect event from an application, it updates its type map (setting or unsetting the respective bits) and notifies its neighbours about the change. The CORE services of the neighbours then in turn generate connect and disconnect events for the peer that sent the type map for their respective applications. As CORE messages may be lost, the CORE service confirms receiving a type map by sending back a `GNUNET_MESSAGE_TYPE_CORE_CONFIRM_TYPE_MAP`. If such a confirmation (with the correct hash of the type map) is not received, the sender will retransmit the type map (with exponential back-off).

5.24 CADET Subsystem

The CADET subsystem in GNUnet is responsible for secure end-to-end communications between nodes in the GNUnet overlay network. CADET builds on the CORE subsystem which provides for the link-layer communication and then adds routing, forwarding and additional security to the connections. CADET offers the same cryptographic services as CORE, but on an end-to-end level. This is done so peers retransmitting traffic on behalf of other peers cannot access the payload data.

- CADET provides confidentiality with so-called perfect forward secrecy; we use ECDHE powered by Curve25519 for the key exchange and then use symmetric encryption, encrypting with both AES-256 and Twofish
- authentication is achieved by signing the ephemeral keys using Ed25519, a deterministic variant of ECDSA
- integrity protection (using SHA-512 to do encrypt-then-MAC, although only 256 bits are sent to reduce overhead)
- replay protection (using nonces, timestamps, challenge-response, message counters and ephemeral keys)
- liveness (keep-alive messages, timeout)

Additional to the CORE-like security benefits, CADET offers other properties that make it a more universal service than CORE.

- CADET can establish channels to arbitrary peers in GNUnet. If a peer is not immediately reachable, CADET will find a path through the network and ask other peers to retransmit the traffic on its behalf.
- CADET offers (optional) reliability mechanisms. In a reliable channel traffic is guaranteed to arrive complete, unchanged and in-order.

- CADET takes care of flow and congestion control mechanisms, not allowing the sender to send more traffic than the receiver or the network are able to process.

5.24.1 libgnunetcadet

The CADET API (defined in `gnunet_cadet_service.h`) is the messaging API used by P2P applications built using GNUnet. It provides applications the ability to send and receive encrypted messages to any peer participating in GNUnet. The API is heavily base on the CORE API.

CADET delivers messages to other peers in "channels". A channel is a permanent connection defined by a destination peer (identified by its public key) and a port number. Internally, CADET tunnels all channels towards a destiantion peer using one session key and relays the data on multiple "connections", independent from the channels.

Each channel has optional paramenters, the most important being the reliability flag. Should a message get lost on TRANSPORT/CORE level, if a channel is created with as reliable, CADET will retransmit the lost message and deliver it in order to the destination application.

To communicate with other peers using CADET, it is necessary to first connect to the service using `GNUNET_CADET_connect`. This function takes several parameters in form of callbacks, to allow the client to react to various events, like incoming channels or channels that terminate, as well as specify a list of ports the client wishes to listen to (at the moment it is not possible to start listening on further ports once connected, but nothing prevents a client to connect several times to CADET, even do one connection per listening port). The function returns a handle which has to be used for any further interaction with the service.

To connect to a remote peer a client has to call the `GNUNET_CADET_channel_create` function. The most important parameters given are the remote peer's identity (it public key) and a port, which specifies which application on the remote peer to connect to, similar to TCP/UDP ports. CADET will then find the peer in the GNUnet network and establish the proper low-level connections and do the necessary key exchanges to assure and authenticated, secure and verified communication. Similar to `GNUNET_CADET_connect`,`GNUNET_CADET_create_channel` returns a handle to interact with the created channel.

For every message the client wants to send to the remote application, `GNUNET_CADET_notify_transmit_ready` must be called, indicating the channel on which the message should be sent and the size of the message (but not the message itself!). Once CADET is ready to send the message, the provided callback will fire, and the message contents are provided to this callback.

Please note the CADET does not provide an explicit notification of when a channel is connected. In loosely connected networks, like big wireless mesh networks, this can take several seconds, even minutes in the worst case. To be alerted when a channel is online, a client can call `GNUNET_CADET_notify_transmit_ready` immediately after `GNUNET_CADET_create_channel`. When the callback is activated, it means that the channel is online. The callback can give 0 bytes to CADET if no message is to be sent, this is ok.

If a transmission was requested but before the callback fires it is no longer needed, it can be cancelled with `GNUNET_CADET_notify_transmit_ready_cancel`, which uses the handle given back by `GNUNET_CADET_notify_transmit_ready`. As in the case of CORE,

only one message can be requested at a time: a client must not call `GNUNET_CADET_notify_transmit_ready` again until the callback is called or the request is cancelled.

When a channel is no longer needed, a client can call `GNUNET_CADET_channel_destroy` to get rid of it. Note that CADET will try to transmit all pending traffic before notifying the remote peer of the destruction of the channel, including retransmitting lost messages if the channel was reliable.

Incoming channels, channels being closed by the remote peer, and traffic on any incoming or outgoing channels are given to the client when CADET executes the callbacks given to it at the time of `GNUNET_CADET_connect`.

Finally, when an application no longer wants to use CADET, it should call `GNUNET_CADET_disconnect`, but first all channels and pending transmissions must be closed (otherwise CADET will complain).

5.25 NSE Subsystem

NSE stands for *Network Size Estimation*. The NSE subsystem provides other subsystems and users with a rough estimate of the number of peers currently participating in the GNUnet overlay. The computed value is not a precise number as producing a precise number in a decentralized, efficient and secure way is impossible. While NSE's estimate is inherently imprecise, NSE also gives the expected range. For a peer that has been running in a stable network for a while, the real network size will typically (99.7% of the time) be in the range of [2/3 estimate, 3/2 estimate]. We will now give an overview of the algorithm used to calculate the estimate; all of the details can be found in this technical report.

5.25.1 Motivation

Some subsytems, like DHT, need to know the size of the GNUnet network to optimize some parameters of their own protocol. The decentralized nature of GNUnet makes efficient and securely counting the exact number of peers infeasable. Although there are several decentralized algorithms to count the number of peers in a system, so far there is none to do so securely. Other protocols may allow any malicious peer to manipulate the final result or to take advantage of the system to perform *Denial of Service* (DoS) attacks against the network. GNUnet's NSE protocol avoids these drawbacks.

5.25.1.1 Security

The NSE subsystem is designed to be resilient against these attacks. It uses proofs of work (`http://en.wikipedia.org/wiki/Proof-of-work_system`) to prevent one peer from impersonating a large number of participants, which would otherwise allow an adversary to artifically inflate the estimate. The DoS protection comes from the time-based nature of the protocol: the estimates are calculated periodically and out-of-time traffic is either ignored or stored for later retransmission by benign peers. In particular, peers cannot trigger global network communication at will.

5.25.2 Principle

The algorithm calculates the estimate by finding the globally closest peer ID to a random, time-based value.

The idea is that the closer the ID is to the random value, the more "densely packed" the ID space is, and therefore, more peers are in the network.

5.25.2.1 Example

Suppose all peers have IDs between 0 and 100 (our ID space), and the random value is 42. If the closest peer has the ID 70 we can imagine that the average "distance" between peers is around 30 and therefore the are around 3 peers in the whole ID space. On the other hand, if the closest peer has the ID 44, we can imagine that the space is rather packed with peers, maybe as much as 50 of them. Naturally, we could have been rather unlucky, and there is only one peer and happens to have the ID 44. Thus, the current estimate is calculated as the average over multiple rounds, and not just a single sample.

5.25.2.2 Algorithm

Given that example, one can imagine that the job of the subsystem is to efficiently communicate the ID of the closest peer to the target value to all the other peers, who will calculate the estimate from it.

5.25.2.3 Target value

The target value itself is generated by hashing the current time, rounded down to an agreed value. If the rounding amount is 1h (default) and the time is 12:34:56, the time to hash would be 12:00:00. The process is repeated each rouning amount (in this example would be every hour). Every repetition is called a round.

5.25.2.4 Timing

The NSE subsystem has some timing control to avoid everybody broadcasting its ID all at one. Once each peer has the target random value, it compares its own ID to the target and calculates the hypothetical size of the network if that peer were to be the closest. Then it compares the hypothetical size with the estimate from the previous rounds. For each value there is an assiciated point in the period, let's call it "broadcast time". If its own hypothetical estimate is the same as the previous global estimate, its "broadcast time" will be in the middle of the round. If its bigger it will be earlier and if its smaller (the most likely case) it will be later. This ensures that the peers closests to the target value start broadcasting their ID the first.

5.25.2.5 Controlled Flooding

When a peer receives a value, first it verifies that it is closer than the closest value it had so far, otherwise it answers the incoming message with a message containing the better value. Then it checks a proof of work that must be included in the incoming message, to ensure that the other peer's ID is not made up (otherwise a malicious peer could claim to have an ID of exactly the target value every round). Once validated, it compares the brodcast time of the received value with the current time and if it's not too early, sends the received value to its neighbors. Otherwise it stores the value until the correct broadcast time comes. This prevents unnecessary traffic of sub-optimal values, since a better value can come before the broadcast time, rendering the previous one obsolete and saving the traffic that would have been used to broadcast it to the neighbors.

5.25.2.6 Calculating the estimate

Once the closest ID has been spread across the network each peer gets the exact distance betweed this ID and the target value of the round and calculates the estimate with a mathematical formula described in the tech report. The estimate generated with this method for a single round is not very precise. Remember the case of the example, where the only peer is the ID 44 and we happen to generate the target value 42, thinking there are 50 peers in the network. Therefore, the NSE subsystem remembers the last 64 estimates and calculates an average over them, giving a result of which usually has one bit of uncertainty (the real size could be half of the estimate or twice as much). Note that the actual network size is calculated in powers of two of the raw input, thus one bit of uncertainty means a factor of two in the size estimate.

5.25.3 libgnunetnse

The NSE subsystem has the simplest API of all services, with only two calls: `GNUNET_NSE_connect` and `GNUNET_NSE_disconnect`.

The connect call gets a callback function as a parameter and this function is called each time the network agrees on an estimate. This usually is once per round, with some exceptions: if the closest peer has a late local clock and starts spreading his ID after everyone else agreed on a value, the callback might be activated twice in a round, the second value being always bigger than the first. The default round time is set to 1 hour.

The disconnect call disconnects from the NSE subsystem and the callback is no longer called with new estimates.

5.25.3.1 Results

The callback provides two values: the average and the standard deviation (`http://en.wikipedia.org/wiki/Standard_deviation`) of the last 64 rounds. The values provided by the callback function are logarithmic, this means that the real estimate numbers can be obtained by calculating 2 to the power of the given value (2average). From a statistics point of view this means that:

- 68% of the time the real size is included in the interval [(2average-stddev), 2]
- 95% of the time the real size is included in the interval [(2average-2*stddev, 2^average+2*stddev]
- 99.7% of the time the real size is included in the interval [(2average-3*stddev, 2average+3*stddev]

The expected standard variation for 64 rounds in a network of stable size is 0.2. Thus, we can say that normally:

- 68% of the time the real size is in the range [-13%, +15%]
- 95% of the time the real size is in the range [-24%, +32%]
- 99.7% of the time the real size is in the range [-34%, +52%]

As said in the introduction, we can be quite sure that usually the real size is between one third and three times the estimate. This can of course vary with network conditions. Thus, applications may want to also consider the provided standard deviation value, not only the average (in particular, if the standard veriation is very high, the average maybe meaningless: the network size is changing rapidly).

5.25.3.2 libgnunetnse -Examples

Let's close with a couple examples.

Average: 10, std dev: 1 Here the estimate would be
2^10 = 1024 peers.[19]

Average 22, std dev: 0.2 Here the estimate would be
2^22 = 4 Million peers.[20]

To put this in perspective, if someone remembers the LHC Higgs boson results, were announced with "5 sigma" and "6 sigma" certainties. In this case a 5 sigma minimum would be 2 million and a 6 sigma minimum, 1.8 million.

5.25.4 The NSE Client-Service Protocol

As with the API, the client-service protocol is very simple, only has 2 different messages, defined in `src/nse/nse.h`:

- `GNUNET_MESSAGE_TYPE_NSE_START` This message has no parameters and is sent from the client to the service upon connection.
- `GNUNET_MESSAGE_TYPE_NSE_ESTIMATE` This message is sent from the service to the client for every new estimate and upon connection. Contains a timestamp for the estimate, the average and the standard deviation for the respective round.

When the `GNUNET_NSE_disconnect` API call is executed, the client simply disconnects from the service, with no message involved.

5.25.5 The NSE Peer-to-Peer Protocol

The NSE subsystem only has one message in the P2P protocol, the `GNUNET_MESSAGE_TYPE_NSE_P2P_FLOOD` message.

This message key contents are the timestamp to identify the round (differences in system clocks may cause some peers to send messages way too early or way too late, so the timestamp allows other peers to identify such messages easily), the proof of work (`http://en.wikipedia.org/wiki/Proof-of-work_system`) used to make it difficult to mount a Sybil attack (`http://en.wikipedia.org/wiki/Sybil_attack`), and the public key, which is used to verify the signature on the message.

Every peer stores a message for the previous, current and next round. The messages for the previous and current round are given to peers that connect to us. The message for the next round is simply stored until our system clock advances to the next round. The message for the current round is what we are flooding the network with right now. At the beginning of each round the peer does the following:

- calculates his own distance to the target value
- creates, signs and stores the message for the current round (unless it has a better message in the "next round" slot which came early in the previous round)

---

[19] The range in which we can be 95% sure is: [2^8, 2^12] = [256, 4096]. We can be very (>99.7%) sure that the network is not a hundred peers and absolutely sure that it is not a million peers, but somewhere around a thousand.

[20] The range in which we can be 99.7% sure is: [2^21.4, 2^22.6] = [2.8M, 6.3M]. We can be sure that the network size is around four million, with absolutely way of it being 1 million.

- calculates, based on the stored round message (own or received) when to stard flooding it to its neighbors

Upon receiving a message the peer checks the validity of the message (round, proof of work, signature). The next action depends on the contents of the incoming message:

- if the message is worse than the current stored message, the peer sends the current message back immediately, to stop the other peer from spreading suboptimal results
- if the message is better than the current stored message, the peer stores the new message and calculates the new target time to start spreading it to its neighbors (excluding the one the message came from)
- if the message is for the previous round, it is compared to the message stored in the "previous round slot", which may then be updated
- if the message is for the next round, it is compared to the message stored in the "next round slot", which again may then be updated

Finally, when it comes to send the stored message for the current round to the neighbors there is a random delay added for each neighbor, to avoid traffic spikes and minimize cross-messages.

5.26 HOSTLIST Subsystem

Peers in the GNUnet overlay network need address information so that they can connect with other peers. GNUnet uses so called HELLO messages to store and exchange peer addresses. GNUnet provides several methods for peers to obtain this information:

- out-of-band exchange of HELLO messages (manually, using for example gnunet-peerinfo)
- HELLO messages shipped with GNUnet (automatic with distribution)
- UDP neighbor discovery in LAN (IPv4 broadcast, IPv6 multicast)
- topology gossiping (learning from other peers we already connected to), and
- the HOSTLIST daemon covered in this section, which is particularly relevant for bootstrapping new peers.

New peers have no existing connections (and thus cannot learn from gossip among peers), may not have other peers in their LAN and might be started with an outdated set of HELLO messages from the distribution. In this case, getting new peers to connect to the network requires either manual effort or the use of a HOSTLIST to obtain HELLOs.

5.26.1 HELLOs

The basic information peers require to connect to other peers are contained in so called HELLO messages you can think of as a business card. Besides the identity of the peer (based on the cryptographic public key) a HELLO message may contain address information that specifies ways to contact a peer. By obtaining HELLO messages, a peer can learn how to contact other peers.

5.26.2 Overview for the HOSTLIST subsystem

The HOSTLIST subsystem provides a way to distribute and obtain contact information to connect to other peers using a simple HTTP GET request. It's implementation is split in

three parts, the main file for the daemon itself (`gnunet-daemon-hostlist.c`), the HTTP client used to download peer information (`hostlist-client.c`) and the server component used to provide this information to other peers (`hostlist-server.c`). The server is basically a small HTTP web server (based on GNU libmicrohttpd) which provides a list of HELLOs known to the local peer for download. The client component is basically a HTTP client (based on libcurl) which can download hostlists from one or more websites. The hostlist format is a binary blob containing a sequence of HELLO messages. Note that any HTTP server can theoretically serve a hostlist, the build-in hostlist server makes it simply convenient to offer this service.

5.26.2.1 Features

The HOSTLIST daemon can:

- provide HELLO messages with validated addresses obtained from PEERINFO to download for other peers
- download HELLO messages and forward these message to the TRANSPORT subsystem for validation
- advertises the URL of this peer's hostlist address to other peers via gossip
- automatically learn about hostlist servers from the gossip of other peers

5.26.2.2 HOSTLIST - Limitations

The HOSTLIST daemon does not:

- verify the cryptographic information in the HELLO messages
- verify the address information in the HELLO messages

5.26.3 Interacting with the HOSTLIST daemon

The HOSTLIST subsystem is currently implemented as a daemon, so there is no need for the user to interact with it and therefore there is no command line tool and no API to communicate with the daemon. In the future, we can envision changing this to allow users to manually trigger the download of a hostlist.

Since there is no command line interface to interact with HOSTLIST, the only way to interact with the hostlist is to use STATISTICS to obtain or modify information about the status of HOSTLIST:

```
$ gnunet-statistics -s hostlist
```

In particular, HOSTLIST includes a **persistent** value in statistics that specifies when the hostlist server might be queried next. As this value is exponentially increasing during runtime, developers may want to reset or manually adjust it. Note that HOSTLIST (but not STATISTICS) needs to be shutdown if changes to this value are to have any effect on the daemon (as HOSTLIST does not monitor STATISTICS for changes to the download frequency).

5.26.4 Hostlist security address validation

Since information obtained from other parties cannot be trusted without validation, we have to distinguish between *validated* and *not validated* addresses. Before using (and so trusting) information from other parties, this information has to be double-checked (validated). Address validation is not done by HOSTLIST but by the TRANSPORT service.

The HOSTLIST component is functionally located between the PEERINFO and the TRANSPORT subsystem. When acting as a server, the daemon obtains valid (*validated*) peer information (HELLO messages) from the PEERINFO service and provides it to other peers. When acting as a client, it contacts the HOSTLIST servers specified in the configuration, downloads the (unvalidated) list of HELLO messages and forwards these information to the TRANSPORT server to validate the addresses.

5.26.5 The HOSTLIST daemon

The hostlist daemon is the main component of the HOSTLIST subsystem. It is started by the ARM service and (if configured) starts the HOSTLIST client and server components.

If the daemon provides a hostlist itself it can advertise it's own hostlist to other peers. To do so it sends a `GNUNET_MESSAGE_TYPE_HOSTLIST_ADVERTISEMENT` message to other peers when they connect to this peer on the CORE level. This hostlist advertisement message contains the URL to access the HOSTLIST HTTP server of the sender. The daemon may also subscribe to this type of message from CORE service, and then forward these kind of message to the HOSTLIST client. The client then uses all available URLs to download peer information when necessary.

When starting, the HOSTLIST daemon first connects to the CORE subsystem and if hostlist learning is enabled, registers a CORE handler to receive this kind of messages. Next it starts (if configured) the client and server. It passes pointers to CORE connect and disconnect and receive handlers where the client and server store their functions, so the daemon can notify them about CORE events.

To clean up on shutdown, the daemon has a cleaning task, shutting down all subsystems and disconnecting from CORE.

5.26.6 The HOSTLIST server

The server provides a way for other peers to obtain HELLOs. Basically it is a small web server other peers can connect to and download a list of HELLOs using standard HTTP; it may also advertise the URL of the hostlist to other peers connecting on CORE level.

5.26.6.1 The HTTP Server

During startup, the server starts a web server listening on the port specified with the HTTPPORT value (default 8080). In addition it connects to the PEERINFO service to obtain peer information. The HOSTLIST server uses the GNUNET_PEERINFO_iterate function to request HELLO information for all peers and adds their information to a new hostlist if they are suitable (expired addresses and HELLOs without addresses are both not suitable) and the maximum size for a hostlist is not exceeded (MAX_BYTES_PER_HOSTLISTS = 500000). When PEERINFO finishes (with a last NULL callback), the server destroys the previous hostlist response available for download on the web server and replaces it with the updated hostlist. The hostlist format is basically a sequence of HELLO messages (as obtained from PEERINFO) without any special tokenization. Since each HELLO message contains a size field, the response can easily be split into separate HELLO messages by the client.

A HOSTLIST client connecting to the HOSTLIST server will receive the hostlist as a HTTP response and the the server will terminate the connection with the result code `HTTP 200 OK`. The connection will be closed immediately if no hostlist is available.

5.26.6.2 Advertising the URL

The server also advertises the URL to download the hostlist to other peers if hostlist advertisement is enabled. When a new peer connects and has hostlist learning enabled, the server sends a `GNUNET_MESSAGE_TYPE_HOSTLIST_ADVERTISEMENT` message to this peer using the CORE service.

5.26.7 The HOSTLIST client

The client provides the functionality to download the list of HELLOs from a set of URLs. It performs a standard HTTP request to the URLs configured and learned from advertisement messages received from other peers. When a HELLO is downloaded, the HOSTLIST client forwards the HELLO to the TRANSPORT service for validation.

The client supports two modes of operation:

- download of HELLOs (bootstrapping)
- learning of URLs

5.26.7.1 Bootstrapping

For bootstrapping, it schedules a task to download the hostlist from the set of known URLs. The downloads are only performed if the number of current connections is smaller than a minimum number of connections (at the moment 4). The interval between downloads increases exponentially; however, the exponential growth is limited if it becomes longer than an hour. At that point, the frequency growth is capped at (#number of connections * 1h).

Once the decision has been taken to download HELLOs, the daemon chooses a random URL from the list of known URLs. URLs can be configured in the configuration or be learned from advertisement messages. The client uses a HTTP client library (libcurl) to initiate the download using the libcurl multi interface. Libcurl passes the data to the callback_download function which stores the data in a buffer if space is available and the maximum size for a hostlist download is not exceeded (MAX_BYTES_PER_HOSTLISTS = 500000). When a full HELLO was downloaded, the HOSTLIST client offers this HELLO message to the TRANSPORT service for validation. When the download is finished or failed, statistical information about the quality of this URL is updated.

5.26.7.2 Learning

The client also manages hostlist advertisements from other peers. The HOSTLIST daemon forwards `GNUNET_MESSAGE_TYPE_HOSTLIST_ADVERTISEMENT` messages to the client subsystem, which extracts the URL from the message. Next, a test of the newly obtained URL is performed by triggering a download from the new URL. If the URL works correctly, it is added to the list of working URLs.

The size of the list of URLs is restricted, so if an additional server is added and the list is full, the URL with the worst quality ranking (determined through successful downloads and number of HELLOs e.g.) is discarded. During shutdown the list of URLs is saved to a file for persistance and loaded on startup. URLs from the configuration file are never discarded.

5.26.8 Usage

To start HOSTLIST by default, it has to be added to the DEFAULTSERVICES section for the ARM services. This is done in the default configuration.

For more information on how to configure the HOSTLIST subsystem see the installation handbook: Configuring the hostlist to bootstrap Configuring your peer to provide a hostlist

5.27 IDENTITY Subsystem

Identities of "users" in GNUnet are called egos. Egos can be used as pseudonyms ("fake names") or be tied to an organization (for example, "GNU") or even the actual identity of a human. GNUnet users are expected to have many egos. They might have one tied to their real identity, some for organizations they manage, and more for different domains where they want to operate under a pseudonym.

The IDENTITY service allows users to manage their egos. The identity service manages the private keys egos of the local user; it does not manage identities of other users (public keys). Public keys for other users need names to become manageable. GNUnet uses the *GNU Name System* (GNS) to give names to other users and manage their public keys securely. This chapter is about the IDENTITY service, which is about the management of private keys.

On the network, an ego corresponds to an ECDSA key (over Curve25519, using RFC 6979, as required by GNS). Thus, users can perform actions under a particular ego by using (signing with) a particular private key. Other users can then confirm that the action was really performed by that ego by checking the signature against the respective public key.

The IDENTITY service allows users to associate a human-readable name with each ego. This way, users can use names that will remind them of the purpose of a particular ego. The IDENTITY service will store the respective private keys and allows applications to access key information by name. Users can change the name that is locally (!) associated with an ego. Egos can also be deleted, which means that the private key will be removed and it thus will not be possible to perform actions with that ego in the future.

Additionally, the IDENTITY subsystem can associate service functions with egos. For example, GNS requires the ego that should be used for the shorten zone. GNS will ask IDENTITY for an ego for the "gns-short" service. The IDENTITY service has a mapping of such service strings to the name of the ego that the user wants to use for this service, for example "my-short-zone-ego".

Finally, the IDENTITY API provides access to a special ego, the anonymous ego. The anonymous ego is special in that its private key is not really private, but fixed and known to everyone. Thus, anyone can perform actions as anonymous. This can be useful as with this trick, code does not have to contain a special case to distinguish between anonymous and pseudonymous egos.

5.27.1 libgnunetidentity

5.27.1.1 Connecting to the service

First, typical clients connect to the identity service using `GNUNET_IDENTITY_connect`. This function takes a callback as a parameter. If the given callback parameter is non-null, it will be invoked to notify the application about the current state of the identities in the system.

- First, it will be invoked on all known egos at the time of the connection. For each ego, a handle to the ego and the user's name for the ego will be passed to the callback. Furthermore, a `void **` context argument will be provided which gives the client the opportunity to associate some state with the ego.
- Second, the callback will be invoked with NULL for the ego, the name and the context. This signals that the (initial) iteration over all egos has completed.
- Then, the callback will be invoked whenever something changes about an ego. If an ego is renamed, the callback is invoked with the ego handle of the ego that was renamed, and the new name. If an ego is deleted, the callback is invoked with the ego handle and a name of NULL. In the deletion case, the application should also release resources stored in the context.
- When the application destroys the connection to the identity service using `GNUNET_IDENTITY_disconnect`, the callback is again invoked with the ego and a name of NULL (equivalent to deletion of the egos). This should again be used to clean up the per-ego context.

The ego handle passed to the callback remains valid until the callback is invoked with a name of NULL, so it is safe to store a reference to the ego's handle.

5.27.1.2 Operations on Egos

Given an ego handle, the main operations are to get its associated private key using `GNUNET_IDENTITY_ego_get_private_key` or its associated public key using `GNUNET_IDENTITY_ego_get_public_key`.

The other operations on egos are pretty straightforward. Using `GNUNET_IDENTITY_create`, an application can request the creation of an ego by specifying the desired name. The operation will fail if that name is already in use. Using `GNUNET_IDENTITY_rename` the name of an existing ego can be changed. Finally, egos can be deleted using `GNUNET_IDENTITY_delete`. All of these operations will trigger updates to the callback given to the `GNUNET_IDENTITY_connect` function of all applications that are connected with the identity service at the time. `GNUNET_IDENTITY_cancel` can be used to cancel the operations before the respective continuations would be called. It is not guaranteed that the operation will not be completed anyway, only the continuation will no longer be called.

5.27.1.3 The anonymous Ego

A special way to obtain an ego handle is to call `GNUNET_IDENTITY_ego_get_anonymous`, which returns an ego for the "anonymous" user — anyone knows and can get the private key for this user, so it is suitable for operations that are supposed to be anonymous but require signatures (for example, to avoid a special path in the code). The anonymous ego is always valid and accessing it does not require a connection to the identity service.

5.27.1.4 Convenience API to lookup a single ego

As applications commonly simply have to lookup a single ego, there is a convenience API to do just that. Use `GNUNET_IDENTITY_ego_lookup` to lookup a single ego by name. Note that this is the user's name for the ego, not the service function. The resulting ego will be returned via a callback and will only be valid during that callback. The operation can be cancelled via `GNUNET_IDENTITY_ego_lookup_cancel` (cancellation is only legal before the callback is invoked).

5.27.1.5 Associating egos with service functions

The `GNUNET_IDENTITY_set` function is used to associate a particular ego with a service function. The name used by the service and the ego are given as arguments. Afterwards, the service can use its name to lookup the associated ego using `GNUNET_IDENTITY_get`.

5.27.2 The IDENTITY Client-Service Protocol

A client connecting to the identity service first sends a message with type `GNUNET_MESSAGE_TYPE_IDENTITY_START` to the service. After that, the client will receive information about changes to the egos by receiving messages of type `GNUNET_MESSAGE_TYPE_IDENTITY_UPDATE`. Those messages contain the private key of the ego and the user's name of the ego (or zero bytes for the name to indicate that the ego was deleted). A special bit `end_of_list` is used to indicate the end of the initial iteration over the identity service's egos.

The client can trigger changes to the egos by sending `CREATE`, `RENAME` or `DELETE` messages. The CREATE message contains the private key and the desired name. The RENAME message contains the old name and the new name. The DELETE message only needs to include the name of the ego to delete. The service responds to each of these messages with a `RESULT_CODE` message which indicates success or error of the operation, and possibly a human-readable error message.

Finally, the client can bind the name of a service function to an ego by sending a `SET_DEFAULT` message with the name of the service function and the private key of the ego. Such bindings can then be resolved using a `GET_DEFAULT` message, which includes the name of the service function. The identity service will respond to a GET_DEFAULT request with a SET_DEFAULT message containing the respective information, or with a RESULT_CODE to indicate an error.

5.28 NAMESTORE Subsystem

The NAMESTORE subsystem provides persistent storage for local GNS zone information. All local GNS zone information are managed by NAMESTORE. It provides both the functionality to administer local GNS information (e.g. delete and add records) as well as to retrieve GNS information (e.g to list name information in a client). NAMESTORE does only manage the persistent storage of zone information belonging to the user running the service: GNS information from other users obtained from the DHT are stored by the NAMECACHE subsystem.

NAMESTORE uses a plugin-based database backend to store GNS information with good performance. Here sqlite, MySQL and PostgreSQL are supported database backends. NAMESTORE clients interact with the IDENTITY subsystem to obtain cryptographic information about zones based on egos as described with the IDENTITY subsystem, but internally NAMESTORE refers to zones using the ECDSA private key. In addition, it collaborates with the NAMECACHE subsystem and stores zone information when local information are modified in the GNS cache to increase look-up performance for local information.

NAMESTORE provides functionality to look-up and store records, to iterate over a specific or all zones and to monitor zones for changes. NAMESTORE functionality can be accessed using the NAMESTORE api or the NAMESTORE command line tool.

5.28.1 libgnunetnamestore

To interact with NAMESTORE clients first connect to the NAMESTORE service using the `GNUNET_NAMESTORE_connect` passing a configuration handle. As a result they obtain a NAMESTORE handle, they can use for operations, or NULL is returned if the connection failed.

To disconnect from NAMESTORE, clients use `GNUNET_NAMESTORE_disconnect` and specify the handle to disconnect.

NAMESTORE internally uses the ECDSA private key to refer to zones. These private keys can be obtained from the IDENTITY subsytem. Here *egos can be used to refer to zones or the default ego assigned to the GNS subsystem can be used to obtained the master zone's private key.*

5.28.1.1 Editing Zone Information

NAMESTORE provides functions to lookup records stored under a label in a zone and to store records under a label in a zone.

To store (and delete) records, the client uses the `GNUNET_NAMESTORE_records_store` function and has to provide namestore handle to use, the private key of the zone, the label to store the records under, the records and number of records plus an callback function. After the operation is performed NAMESTORE will call the provided callback function with the result GNUNET_SYSERR on failure (including timeout/queue drop/failure to validate), GNUNET_NO if content was already there or not found GNUNET_YES (or other positive value) on success plus an additional error message.

Records are deleted by using the store command with 0 records to store. It is important to note, that records are not merged when records exist with the label. So a client has first to retrieve records, merge with existing records and then store the result.

To perform a lookup operation, the client uses the `GNUNET_NAMESTORE_records_store` function. Here he has to pass the namestore handle, the private key of the zone and the label. He also has to provide a callback function which will be called with the result of the lookup operation: the zone for the records, the label, and the records including the number of records included.

A special operation is used to set the preferred nickname for a zone. This nickname is stored with the zone and is automatically merged with all labels and records stored in a zone. Here the client uses the `GNUNET_NAMESTORE_set_nick` function and passes the private key of the zone, the nickname as string plus a the callback with the result of the operation.

5.28.1.2 Iterating Zone Information

A client can iterate over all information in a zone or all zones managed by NAMESTORE. Here a client uses the `GNUNET_NAMESTORE_zone_iteration_start` function and passes the namestore handle, the zone to iterate over and a callback function to call with the result. If the client wants to iterate over all the, he passes NULL for the zone. A `GNUNET_NAMESTORE_ZoneIterator` handle is returned to be used to continue iteration.

NAMESTORE calls the callback for every result and expects the client to call `GNUNET_NAMESTORE_zone_iterator_next` to continue to iterate or `GNUNET_NAMESTORE_zone_iterator_stop` to interrupt the iteration. When NAMESTORE reached the last item it will call the callback with a NULL value to indicate.

5.28.1.3 Monitoring Zone Information

Clients can also monitor zones to be notified about changes. Here the clients uses the `GNUNET_NAMESTORE_zone_monitor_start` function and passes the private key of the zone and and a callback function to call with updates for a zone. The client can specify to obtain zone information first by iterating over the zone and specify a synchronization callback to be called when the client and the namestore are synced.

On an update, NAMESTORE will call the callback with the private key of the zone, the label and the records and their number.

To stop monitoring, the client calls `GNUNET_NAMESTORE_zone_monitor_stop` and passes the handle obtained from the function to start the monitoring.

5.29 PEERINFO Subsystem

The PEERINFO subsystem is used to store verified (validated) information about known peers in a persistent way. It obtains these addresses for example from TRANSPORT service which is in charge of address validation. Validation means that the information in the HELLO message are checked by connecting to the addresses and performing a cryptographic handshake to authenticate the peer instance stating to be reachable with these addresses. Peerinfo does not validate the HELLO messages itself but only stores them and gives them to interested clients.

As future work, we think about moving from storing just HELLO messages to providing a generic persistent per-peer information store. More and more subsystems tend to need to store per-peer information in persistent way. To not duplicate this functionality we plan to provide a PEERSTORE service providing this functionality.

5.29.1 PEERINFO - Features

- Persistent storage
- Client notification mechanism on update
- Periodic clean up for expired information
- Differentiation between public and friend-only HELLO

5.29.2 PEERINFO - Limitations

- Does not perform HELLO validation

5.29.3 DeveloperPeer Information

The PEERINFO subsystem stores these information in the form of HELLO messages you can think of as business cards. These HELLO messages contain the public key of a peer and the addresses a peer can be reached under. The addresses include an expiration date describing how long they are valid. This information is updated regularly by the TRANSPORT service by revalidating the address. If an address is expired and not renewed, it can be removed from the HELLO message.

Some peer do not want to have their HELLO messages distributed to other peers, especially when GNUnet's friend-to-friend modus is enabled. To prevent this undesired distribution. PEERINFO distinguishes between *public* and *friend-only* HELLO messages. Public

HELLO messages can be freely distributed to other (possibly unknown) peers (for example using the hostlist, gossiping, broadcasting), whereas friend-only HELLO messages may not be distributed to other peers. Friend-only HELLO messages have an additional flag `friend_only` set internally. For public HELLO message this flag is not set. PEERINFO does and cannot not check if a client is allowed to obtain a specific HELLO type.

The HELLO messages can be managed using the GNUnet HELLO library. Other GNUnet systems can obtain these information from PEERINFO and use it for their purposes. Clients are for example the HOSTLIST component providing these information to other peers in form of a hostlist or the TRANSPORT subsystem using these information to maintain connections to other peers.

5.29.4 Startup

During startup the PEERINFO services loads persistent HELLOs from disk. First PEERINFO parses the directory configured in the HOSTS value of the `PEERINFO` configuration section to store PEERINFO information. For all files found in this directory valid HELLO messages are extracted. In addition it loads HELLO messages shipped with the GNUnet distribution. These HELLOs are used to simplify network bootstrapping by providing valid peer information with the distribution. The use of these HELLOs can be prevented by setting the `USE_INCLUDED_HELLOS` in the `PEERINFO` configuration section to `NO`. Files containing invalid information are removed.

5.29.5 Managing Information

The PEERINFO services stores information about known PEERS and a single HELLO message for every peer. A peer does not need to have a HELLO if no information are available. HELLO information from different sources, for example a HELLO obtained from a remote HOSTLIST and a second HELLO stored on disk, are combined and merged into one single HELLO message per peer which will be given to clients. During this merge process the HELLO is immediately written to disk to ensure persistence.

PEERINFO in addition periodically scans the directory where information are stored for empty HELLO messages with expired TRANSPORT addresses. This periodic task scans all files in the directory and recreates the HELLO messages it finds. Expired TRANSPORT addresses are removed from the HELLO and if the HELLO does not contain any valid addresses, it is discarded and removed from the disk.

5.29.6 Obtaining Information

When a client requests information from PEERINFO, PEERINFO performs a lookup for the respective peer or all peers if desired and transmits this information to the client. The client can specify if friend-only HELLOs have to be included or not and PEERINFO filters the respective HELLO messages before transmitting information.

To notify clients about changes to PEERINFO information, PEERINFO maintains a list of clients interested in this notifications. Such a notification occurs if a HELLO for a peer was updated (due to a merge for example) or a new peer was added.

5.29.7 The PEERINFO Client-Service Protocol

To connect and disconnect to and from the PEERINFO Service PEERINFO utilizes the util client/server infrastructure, so no special messages types are used here.

To add information for a peer, the plain HELLO message is transmitted to the service without any wrapping. All pieces of information required are stored within the HELLO message. The PEERINFO service provides a message handler accepting and processing these HELLO messages.

When obtaining PEERINFO information using the iterate functionality specific messages are used. To obtain information for all peers, a `struct ListAllPeersMessage` with message type `GNUNET_MESSAGE_TYPE_PEERINFO_GET_ALL` and a flag include_friend_only to indicate if friend-only HELLO messages should be included are transmitted. If information for a specific peer is required a `struct ListAllPeersMessage` with `GNUNET_MESSAGE_TYPE_PEERINFO_GET` containing the peer identity is used.

For both variants the PEERINFO service replies for each HELLO message it wants to transmit with a `struct ListAllPeersMessage` with type `GNUNET_MESSAGE_TYPE_PEERINFO_INFO` containing the plain HELLO. The final message is `struct GNUNET_MessageHeader` with type `GNUNET_MESSAGE_TYPE_PEERINFO_INFO`. If the client receives this message, it can proceed with the next request if any is pending.

5.29.8 libgnunetpeerinfo

The PEERINFO API consists mainly of three different functionalities:

- maintaining a connection to the service
- adding new information to the PEERINFO service
- retrieving information from the PEERINFO service

5.29.8.1 Connecting to the PEERINFO Service

To connect to the PEERINFO service the function `GNUNET_PEERINFO_connect` is used, taking a configuration handle as an argument, and to disconnect from PEERINFO the function `GNUNET_PEERINFO_disconnect`, taking the PEERINFO handle returned from the connect function has to be called.

5.29.8.2 Adding Information to the PEERINFO Service

`GNUNET_PEERINFO_add_peer` adds a new peer to the PEERINFO subsystem storage. This function takes the PEERINFO handle as an argument, the HELLO message to store and a continuation with a closure to be called with the result of the operation. The `GNUNET_PEERINFO_add_peer` returns a handle to this operation allowing to cancel the operation with the respective cancel function `GNUNET_PEERINFO_add_peer_cancel`. To retrieve information from PEERINFO you can iterate over all information stored with PEERINFO or you can tell PEERINFO to notify if new peer information are available.

5.29.8.3 Obtaining Information from the PEERINFO Service

To iterate over information in PEERINFO you use `GNUNET_PEERINFO_iterate`. This function expects the PEERINFO handle, a flag if HELLO messages intended for friend only mode should be included, a timeout how long the operation should take and a callback with a callback closure to be called for the results. If you want to obtain information for a specific peer, you can specify the peer identity, if this identity is NULL, information for all peers are returned. The function returns a handle to allow to cancel the operation using `GNUNET_PEERINFO_iterate_cancel`.

To get notified when peer information changes, you can use `GNUNET_PEERINFO_notify`. This function expects a configuration handle and a flag if friend-only HELLO messages should be included. The PEERINFO service will notify you about every change and the callback function will be called to notify you about changes. The function returns a handle to cancel notifications with `GNUNET_PEERINFO_notify_cancel`.

5.30 PEERSTORE Subsystem

GNUnet's PEERSTORE subsystem offers persistent per-peer storage for other GNUnet subsystems. GNUnet subsystems can use PEERSTORE to persistently store and retrieve arbitrary data. Each data record stored with PEERSTORE contains the following fields:

- subsystem: Name of the subsystem responsible for the record.
- peerid: Identity of the peer this record is related to.
- key: a key string identifying the record.
- value: binary record value.
- expiry: record expiry date.

5.30.1 Functionality

Subsystems can store any type of value under a (subsystem, peerid, key) combination. A "replace" flag set during store operations forces the PEERSTORE to replace any old values stored under the same (subsystem, peerid, key) combination with the new value. Additionally, an expiry date is set after which the record is *possibly* deleted by PEERSTORE.

Subsystems can iterate over all values stored under any of the following combination of fields:

- (subsystem)
- (subsystem, peerid)
- (subsystem, key)
- (subsystem, peerid, key)

Subsystems can also request to be notified about any new values stored under a (subsystem, peerid, key) combination by sending a "watch" request to PEERSTORE.

5.30.2 Architecture

PEERSTORE implements the following components:

- PEERSTORE service: Handles store, iterate and watch operations.
- PEERSTORE API: API to be used by other subsystems to communicate and issue commands to the PEERSTORE service.
- PEERSTORE plugins: Handles the persistent storage. At the moment, only an "sqlite" plugin is implemented.

5.30.3 libgnunetpeerstore

libgnunetpeerstore is the library containing the PEERSTORE API. Subsystems wishing to communicate with the PEERSTORE service use this API to open a connection to PEERSTORE. This is done by calling `GNUNET_PEERSTORE_connect` which returns a handle to the newly created connection. This handle has to be used with any further calls to the API.

To store a new record, the function `GNUNET_PEERSTORE_store` is to be used which requires the record fields and a continuation function that will be called by the API after the STORE request is sent to the PEERSTORE service. Note that calling the continuation function does not mean that the record is successfully stored, only that the STORE request has been successfully sent to the PEERSTORE service. `GNUNET_PEERSTORE_store_cancel` can be called to cancel the STORE request only before the continuation function has been called.

To iterate over stored records, the function `GNUNET_PEERSTORE_iterate` is to be used. *peerid* and *key* can be set to NULL. An iterator callback function will be called with each matching record found and a NULL record at the end to signal the end of result set. `GNUNET_PEERSTORE_iterate_cancel` can be used to cancel the ITERATE request before the iterator callback is called with a NULL record.

To be notified with new values stored under a (subsystem, peerid, key) combination, the function `GNUNET_PEERSTORE_watch` is to be used. This will register the watcher with the PEERSTORE service, any new records matching the given combination will trigger the callback function passed to `GNUNET_PEERSTORE_watch`. This continues until `GNUNET_PEERSTORE_watch_cancel` is called or the connection to the service is destroyed.

After the connection is no longer needed, the function `GNUNET_PEERSTORE_disconnect` can be called to disconnect from the PEERSTORE service. Any pending ITERATE or WATCH requests will be destroyed. If the `sync_first` flag is set to `GNUNET_YES`, the API will delay the disconnection until all pending STORE requests are sent to the PEERSTORE service, otherwise, the pending STORE requests will be destroyed as well.

5.31 SET Subsystem

The SET service implements efficient set operations between two peers over a mesh tunnel. Currently, set union and set intersection are the only supported operations. Elements of a set consist of an *element type* and arbitrary binary *data*. The size of an element's data is limited to around 62 KB.

5.31.1 Local Sets

Sets created by a local client can be modified and reused for multiple operations. As each set operation requires potentially expensive special auxilliary data to be computed for each element of a set, a set can only participate in one type of set operation (i.e. union or intersection). The type of a set is determined upon its creation. If a the elements of a set are needed for an operation of a different type, all of the set's element must be copied to a new set of appropriate type.

5.31.2 Set Modifications

Even when set operations are active, one can add to and remove elements from a set. However, these changes will only be visible to operations that have been created after the changes have taken place. That is, every set operation only sees a snapshot of the set from the time the operation was started. This mechanism is *not* implemented by copying the whole set, but by attaching *generation information* to each element and operation.

5.31.3 Set Operations

Set operations can be started in two ways: Either by accepting an operation request from a remote peer, or by requesting a set operation from a remote peer. Set operations are uniquely identified by the involved *peers*, an *application id* and the *operation type*.

The client is notified of incoming set operations by *set listeners*. A set listener listens for incoming operations of a specific operation type and application id. Once notified of an incoming set request, the client can accept the set request (providing a local set for the operation) or reject it.

5.31.4 Result Elements

The SET service has three *result modes* that determine how an operation's result set is delivered to the client:

- **Full Result Set.** All elements of set resulting from the set operation are returned to the client.
- **Added Elements.** Only elements that result from the operation and are not already in the local peer's set are returned. Note that for some operations (like set intersection) this result mode will never return any elements. This can be useful if only the remove peer is actually interested in the result of the set operation.
- **Removed Elements.** Only elements that are in the local peer's initial set but not in the operation's result set are returned. Note that for some operations (like set union) this result mode will never return any elements. This can be useful if only the remove peer is actually interested in the result of the set operation.

5.31.5 libgnunetset

5.31.5.1 Sets

New sets are created with `GNUNET_SET_create`. Both the local peer's configuration (as each set has its own client connection) and the operation type must be specified. The set exists until either the client calls `GNUNET_SET_destroy` or the client's connection to the service is disrupted. In the latter case, the client is notified by the return value of functions dealing with sets. This return value must always be checked.

Elements are added and removed with `GNUNET_SET_add_element` and `GNUNET_SET_remove_element`.

5.31.5.2 Listeners

Listeners are created with `GNUNET_SET_listen`. Each time time a remote peer suggests a set operation with an application id and operation type matching a listener, the listener's callback is invoked. The client then must synchronously call either `GNUNET_SET_accept` or `GNUNET_SET_reject`. Note that the operation will not be started until the client calls `GNUNET_SET_commit` (see Section "Supplying a Set").

5.31.5.3 Operations

Operations to be initiated by the local peer are created with `GNUNET_SET_prepare`. Note that the operation will not be started until the client calls `GNUNET_SET_commit` (see Section "Supplying a Set").

5.31.5.4 Supplying a Set

To create symmetry between the two ways of starting a set operation (accepting and nitiating it), the operation handles returned by GNUNET_SET_accept and GNUNET_SET_prepare do not yet have a set to operate on, thus they can not do any work yet.

The client must call GNUNET_SET_commit to specify a set to use for an operation. GNUNET_SET_commit may only be called once per set operation.

5.31.5.5 The Result Callback

Clients must specify both a result mode and a result callback with GNUNET_SET_accept and GNUNET_SET_prepare. The result callback with a status indicating either that an element was received, or the operation failed or succeeded. The interpretation of the received element depends on the result mode. The callback needs to know which result mode it is used in, as the arguments do not indicate if an element is part of the full result set, or if it is in the difference between the original set and the final set.

5.31.6 The SET Client-Service Protocol

5.31.6.1 Creating Sets

For each set of a client, there exists a client connection to the service. Sets are created by sending the GNUNET_SERVICE_SET_CREATE message over a new client connection. Multiple operations for one set are multiplexed over one client connection, using a request id supplied by the client.

5.31.6.2 Listeners2

Each listener also requires a seperate client connection. By sending the GNUNET_SERVICE_SET_LISTEN message, the client notifies the service of the application id and operation type it is interested in. A client rejects an incoming request by sending GNUNET_SERVICE_SET_REJECT on the listener's client connection. In contrast, when accepting an incoming request, a GNUNET_SERVICE_SET_ACCEPT message must be sent over the set that is supplied for the set operation.

5.31.6.3 Initiating Operations

Operations with remote peers are initiated by sending a GNUNET_SERVICE_SET_EVALUATE message to the service. The client connection that this message is sent by determines the set to use.

5.31.6.4 Modifying Sets

Sets are modified with the GNUNET_SERVICE_SET_ADD and GNUNET_SERVICE_SET_REMOVE messages.

5.31.6.5 Results and Operation Status

The service notifies the client of result elements and success/failure of a set operation with the GNUNET_SERVICE_SET_RESULT message.

5.31.6.6 Iterating Sets

All elements of a set can be requested by sending `GNUNET_SERVICE_SET_ITER_REQUEST`. The server responds with `GNUNET_SERVICE_SET_ITER_ELEMENT` and eventually terminates the iteration with `GNUNET_SERVICE_SET_ITER_DONE`. After each received element, the client must send `GNUNET_SERVICE_SET_ITER_ACK`. Note that only one set iteration may be active for a set at any given time.

5.31.7 The SET Intersection Peer-to-Peer Protocol

The intersection protocol operates over CADET and starts with a GNUNET_MESSAGE_TYPE_SET_P2P_OP being sent by the peer initiating the operation to the peer listening for inbound requests. It includes the number of elements of the initiating peer, which is used to decide which side will send a Bloom filter first.

The listening peer checks if the operation type and application identifier are acceptable for its current state. If not, it responds with a GNUNET_MESSAGE_TYPE_SET_RESULT and a status of GNUNET_SET_STATUS_FAILURE (and terminates the CADET channel).

If the application accepts the request, the listener sends back a `GNUNET_MESSAGE_TYPE_SET_INTERSECTION_P2P_ELEMENT_INFO` if it has more elements in the set than the client. Otherwise, it immediately starts with the Bloom filter exchange. If the initiator receives a `GNUNET_MESSAGE_TYPE_SET_INTERSECTION_P2P_ELEMENT_INFO` response, it beings the Bloom filter exchange, unless the set size is indicated to be zero, in which case the intersection is considered finished after just the initial handshake.

5.31.7.1 The Bloom filter exchange

In this phase, each peer transmits a Bloom filter over the remaining keys of the local set to the other peer using a `GNUNET_MESSAGE_TYPE_SET_INTERSECTION_P2P_BF` message. This message additionally includes the number of elements left in the sender's set, as well as the XOR over all of the keys in that set.

The number of bits 'k' set per element in the Bloom filter is calculated based on the relative size of the two sets. Furthermore, the size of the Bloom filter is calculated based on 'k' and the number of elements in the set to maximize the amount of data filtered per byte transmitted on the wire (while avoiding an excessively high number of iterations).

The receiver of the message removes all elements from its local set that do not pass the Bloom filter test. It then checks if the set size of the sender and the XOR over the keys match what is left of his own set. If they do, he sends a `GNUNET_MESSAGE_TYPE_SET_INTERSECTION_P2P_DONE` back to indicate that the latest set is the final result. Otherwise, the receiver starts another Bloom fitler exchange, except this time as the sender.

5.31.7.2 Salt

Bloomfilter operations are probablistic: With some non-zero probability the test may incorrectly say an element is in the set, even though it is not.

To mitigate this problem, the intersection protocol iterates exchanging Bloom filters using a different random 32-bit salt in each iteration (the salt is also included in the message). With different salts, set operations may fail for different elements. Merging the results from the executions, the probability of failure drops to zero.

The iterations terminate once both peers have established that they have sets of the same size, and where the XOR over all keys computes the same 512-bit value (leaving a failure probability of 2-511).

5.31.8 The SET Union Peer-to-Peer Protocol

The SET union protocol is based on Eppstein's efficient set reconciliation without prior context. You should read this paper first if you want to understand the protocol.

The union protocol operates over CADET and starts with a GNUNET_MESSAGE_TYPE_SET_P2P_OPER being sent by the peer initiating the operation to the peer listening for inbound requests. It includes the number of elements of the initiating peer, which is currently not used.

The listening peer checks if the operation type and application identifier are acceptable for its current state. If not, it responds with a `GNUNET_MESSAGE_TYPE_SET_RESULT` and a status of `GNUNET_SET_STATUS_FAILURE` (and terminates the CADET channel).

If the application accepts the request, it sends back a strata estimator using a message of type GNUNET_MESSAGE_TYPE_SET_UNION_P2P_SE. The initiator evaluates the strata estimator and initiates the exchange of invertible Bloom filters, sending a GNUNET_MESSAGE_TYPE_SET_UNION_P2P_IBF.

During the IBF exchange, if the receiver cannot invert the Bloom filter or detects a cycle, it sends a larger IBF in response (up to a defined maximum limit; if that limit is reached, the operation fails). Elements decoded while processing the IBF are transmitted to the other peer using GNUNET_MESSAGE_TYPE_SET_P2P_ELEMENTS, or requested from the other peer using GNUNET_MESSAGE_TYPE_SET_P2P_ELEMENT_REQUESTS messages, depending on the sign observed during decoding of the IBF. Peers respond to a GNUNET_MESSAGE_TYPE_SET_P2P_ELEMENT_REQUESTS message with the respective element in a GNUNET_MESSAGE_TYPE_SET_P2P_ELEMENTS message. If the IBF fully decodes, the peer responds with a GNUNET_MESSAGE_TYPE_SET_UNION_P2P_DONE message instead of another GNUNET_MESSAGE_TYPE_SET_UNION_P2P_IBF.

All Bloom filter operations use a salt to mingle keys before hasing them into buckets, such that future iterations have a fresh chance of succeeding if they failed due to collisions before.

5.32 STATISTICS Subsystem

In GNUnet, the STATISTICS subsystem offers a central place for all subsystems to publish unsigned 64-bit integer run-time statistics. Keeping this information centrally means that there is a unified way for the user to obtain data on all subsystems, and individual subsystems do not have to always include a custom data export method for performance metrics and other statistics. For example, the TRANSPORT system uses STATISTICS to update information about the number of directly connected peers and the bandwidth that has been consumed by the various plugins. This information is valuable for diagnosing connectivity and performance issues.

Following the GNUnet service architecture, the STATISTICS subsystem is divided into an API which is exposed through the header **gnunet_statistics_service.h** and the STATISTICS service **gnunet-service-statistics**. The **gnunet-statistics** command-line tool can be used to obtain (and change) information about the values stored by the STATISTICS service. The STATISTICS service does not communicate with other peers.

Data is stored in the STATISTICS service in the form of tuples **(subsystem, name, value, persistence)**. The subsystem determines to which other GNUnet's subsystem the data belongs. name is the name through which value is associated. It uniquely identifies the record from among other records belonging to the same subsystem. In some parts of the code, the pair **(subsystem, name)** is called a **statistic** as it identifies the values stored in the STATISTCS service.The persistence flag determines if the record has to be preserved across service restarts. A record is said to be persistent if this flag is set for it; if not, the record is treated as a non-persistent record and it is lost after service restart. Persistent records are written to and read from the file **statistics.data** before shutdown and upon startup. The file is located in the HOME directory of the peer.

An anomaly of the STATISTICS service is that it does not terminate immediately upon receiving a shutdown signal if it has any clients connected to it. It waits for all the clients that are not monitors to close their connections before terminating itself. This is to prevent the loss of data during peer shutdown — delaying the STATISTICS service shutdown helps other services to store important data to STATISTICS during shutdown.

5.32.1 libgnunetstatistics

libgnunetstatistics is the library containing the API for the STATISTICS subsystem. Any process requiring to use STATISTICS should use this API by to open a connection to the STATISTICS service. This is done by calling the function `GNUNET_STATISTICS_create()`. This function takes the subsystem's name which is trying to use STATISTICS and a configuration. All values written to STATISTICS with this connection will be placed in the section corresponding to the given subsystem's name. The connection to STATISTICS can be destroyed with the function `GNUNET_STATISTICS_destroy()`. This function allows for the connection to be destroyed immediately or upon transferring all pending write requests to the service.

Note: STATISTICS subsystem can be disabled by setting `DISABLE = YES` under the `[STATISTICS]` section in the configuration. With such a configuration all calls to `GNUNET_STATISTICS_create()` return `NULL` as the STATISTICS subsystem is unavailable and no other functions from the API can be used.

5.32.1.1 Statistics retrieval

Once a connection to the statistics service is obtained, information about any other system which uses statistics can be retrieved with the function GNUNET_STATISTICS_get(). This function takes the connection handle, the name of the subsystem whose information we are interested in (a `NULL` value will retrieve information of all available subsystems using STATISTICS), the name of the statistic we are interested in (a `NULL` value will retrieve all available statistics), a continuation callback which is called when all of requested information is retrieved, an iterator callback which is called for each parameter in the retrieved information and a closure for the aforementioned callbacks. The library then invokes the iterator callback for each value matching the request.

Call to `GNUNET_STATISTICS_get()` is asynchronous and can be canceled with the function `GNUNET_STATISTICS_get_cancel()`. This is helpful when retrieving statistics takes too long and especially when we want to shutdown and cleanup everything.

5.32.1.2 Setting statistics and updating them

So far we have seen how to retrieve statistics, here we will learn how we can set statistics and update them so that other subsystems can retrieve them.

A new statistic can be set using the function `GNUNET_STATISTICS_set()`. This function takes the name of the statistic and its value and a flag to make the statistic persistent. The value of the statistic should be of the type `uint64_t`. The function does not take the name of the subsystem; it is determined from the previous `GNUNET_STATISTICS_create()` invocation. If the given statistic is already present, its value is overwritten.

An existing statistics can be updated, i.e its value can be increased or decreased by an amount with the function `GNUNET_STATISTICS_update()`. The parameters to this function are similar to `GNUNET_STATISTICS_set()`, except that it takes the amount to be changed as a type `int64_t` instead of the value.

The library will combine multiple set or update operations into one message if the client performs requests at a rate that is faster than the available IPC with the STATISTICS service. Thus, the client does not have to worry about sending requests too quickly.

5.32.1.3 Watches

As interesting feature of STATISTICS lies in serving notifications whenever a statistic of our interest is modified. This is achieved by registering a watch through the function `GNUNET_STATISTICS_watch()`. The parameters of this function are similar to those of `GNUNET_STATISTICS_get()`. Changes to the respective statistic's value will then cause the given iterator callback to be called. Note: A watch can only be registered for a specific statistic. Hence the subsystem name and the parameter name cannot be `NULL` in a call to `GNUNET_STATISTICS_watch()`.

A registered watch will keep notifying any value changes until `GNUNET_STATISTICS_watch_cancel()` is called with the same parameters that are used for registering the watch.

5.32.2 The STATISTICS Client-Service Protocol

5.32.2.1 Statistics retrieval2

To retrieve statistics, the client transmits a message of type `GNUNET_MESSAGE_TYPE_STATISTICS_GET` containing the given subsystem name and statistic parameter to the STATISTICS service. The service responds with a message of type `GNUNET_MESSAGE_TYPE_STATISTICS_VALUE` for each of the statistics parameters that match the client request for the client. The end of information retrieved is signaled by the service by sending a message of type `GNUNET_MESSAGE_TYPE_STATISTICS_END`.

5.32.2.2 Setting and updating statistics

The subsystem name, parameter name, its value and the persistence flag are communicated to the service through the message `GNUNET_MESSAGE_TYPE_STATISTICS_SET`.

When the service receives a message of type `GNUNET_MESSAGE_TYPE_STATISTICS_SET`, it retrieves the subsystem name and checks for a statistic parameter with matching the name given in the message. If a statistic parameter is found, the value is overwritten by the new value from the message; if not found then a new statistic parameter is created with the given name and value.

In addition to just setting an absolute value, it is possible to perform a relative update by sending a message of type `GNUNET_MESSAGE_TYPE_STATISTICS_SET` with an update flag (`GNUNET_STATISTICS_SETFLAG_RELATIVE`) signifying that the value in the message should be treated as an update value.

5.32.2.3 Watching for updates

The function registers the watch at the service by sending a message of type `GNUNET_MESSAGE_TYPE_STATISTICS_WATCH`. The service then sends notifications through messages of type `GNUNET_MESSAGE_TYPE_STATISTICS_WATCH_VALUE` whenever the statistic parameter's value is changed.

5.33 Distributed Hash Table (DHT)

GNUnet includes a generic distributed hash table that can be used by developers building P2P applications in the framework. This section documents high-level features and how developers are expected to use the DHT. We have a research paper detailing how the DHT works. Also, Nate's thesis includes a detailed description and performance analysis (in chapter 6).

Key features of GNUnet's DHT include:

- stores key-value pairs with values up to (approximately) 63k in size
- works with many underlay network topologies (small-world, random graph), underlay does not need to be a full mesh / clique
- support for extended queries (more than just a simple 'key'), filtering duplicate replies within the network (bloomfilter) and content validation (for details, please read the subsection on the block library)
- can (optionally) return paths taken by the PUT and GET operations to the application
- provides content replication to handle churn

GNUnet's DHT is randomized and unreliable. Unreliable means that there is no strict guarantee that a value stored in the DHT is always found — values are only found with high probability. While this is somewhat true in all P2P DHTs, GNUnet developers should be particularly wary of this fact (this will help you write secure, fault-tolerant code). Thus, when writing any application using the DHT, you should always consider the possibility that a value stored in the DHT by you or some other peer might simply not be returned, or returned with a significant delay. Your application logic must be written to tolerate this (naturally, some loss of performance or quality of service is expected in this case).

5.33.1 Block library and plugins

5.33.1.1 What is a Block?

Blocks are small (< 63k) pieces of data stored under a key (struct GNUNET_HashCode). Blocks have a type (enum GNUNET_BlockType) which defines their data format. Blocks are used in GNUnet as units of static data exchanged between peers and stored (or cached) locally. Uses of blocks include file-sharing (the files are broken up into blocks), the VPN (DNS information is stored in blocks) and the DHT (all information in the DHT and meta-information for the maintenance of the DHT are both stored using blocks). The block subsystem provides a few common functions that must be available for any type of block.

5.33.1.2 The API of libgnunetblock

The block library requires for each (family of) block type(s) a block plugin (implementing `gnunet_block_plugin.h`) that provides basic functions that are needed by the DHT (and possibly other subsystems) to manage the block. These block plugins are typically implemented within their respective subsystems. The main block library is then used to locate, load and query the appropriate block plugin. Which plugin is appropriate is determined by the block type (which is just a 32-bit integer). Block plugins contain code that specifies which block types are supported by a given plugin. The block library loads all block plugins that are installed at the local peer and forwards the application request to the respective plugin.

The central functions of the block APIs (plugin and main library) are to allow the mapping of blocks to their respective key (if possible) and the ability to check that a block is well-formed and matches a given request (again, if possible). This way, GNUnet can avoid storing invalid blocks, storing blocks under the wrong key and forwarding blocks in response to a query that they do not answer.

One key function of block plugins is that it allows GNUnet to detect duplicate replies (via the Bloom filter). All plugins MUST support detecting duplicate replies (by adding the current response to the Bloom filter and rejecting it if it is encountered again). If a plugin fails to do this, responses may loop in the network.

5.33.1.3 Queries

The query format for any block in GNUnet consists of four main components. First, the type of the desired block must be specified. Second, the query must contain a hash code. The hash code is used for lookups in hash tables and databases and must not be unique for the block (however, if possible a unique hash should be used as this would be best for performance). Third, an optional Bloom filter can be specified to exclude known results; replies that hash to the bits set in the Bloom filter are considered invalid. False-positives can be eliminated by sending the same query again with a different Bloom filter mutator value, which parameterizes the hash function that is used. Finally, an optional application-specific "eXtended query" (xquery) can be specified to further constrain the results. It is entirely up to the type-specific plugin to determine whether or not a given block matches a query (type, hash, Bloom filter, and xquery). Naturally, not all xquery's are valid and some types of blocks may not support Bloom filters either, so the plugin also needs to check if the query is valid in the first place.

Depending on the results from the plugin, the DHT will then discard the (invalid) query, forward the query, discard the (invalid) reply, cache the (valid) reply, and/or forward the (valid and non-duplicate) reply.

5.33.1.4 Sample Code

The source code in **plugin_block_test.c** is a good starting point for new block plugins — it does the minimal work by implementing a plugin that performs no validation at all. The respective **Makefile.am** shows how to build and install a block plugin.

5.33.1.5 Conclusion2

In conclusion, GNUnet subsystems that want to use the DHT need to define a block format and write a plugin to match queries and replies. For testing, the `GNUNET_BLOCK_TYPE_TEST`

block type can be used; it accepts any query as valid and any reply as matching any query. This type is also used for the DHT command line tools. However, it should NOT be used for normal applications due to the lack of error checking that results from this primitive implementation.

5.33.2 libgnunetdht

The DHT API itself is pretty simple and offers the usual GET and PUT functions that work as expected. The specified block type refers to the block library which allows the DHT to run application-specific logic for data stored in the network.

5.33.2.1 GET

When using GET, the main consideration for developers (other than the block library) should be that after issuing a GET, the DHT will continuously cause (small amounts of) network traffic until the operation is explicitly canceled. So GET does not simply send out a single network request once; instead, the DHT will continue to search for data. This is needed to achieve good success rates and also handles the case where the respective PUT operation happens after the GET operation was started. Developers should not cancel an existing GET operation and then explicitly re-start it to trigger a new round of network requests; this is simply inefficient, especially as the internal automated version can be more efficient, for example by filtering results in the network that have already been returned.

If an application that performs a GET request has a set of replies that it already knows and would like to filter, it can call `GNUNET_DHT_get_filter_known_results` with an array of hashes over the respective blocks to tell the DHT that these results are not desired (any more). This way, the DHT will filter the respective blocks using the block library in the network, which may result in a significant reduction in bandwidth consumption.

5.33.2.2 PUT

In contrast to GET operations, developers **must** manually re-run PUT operations periodically (if they intend the content to continue to be available). Content stored in the DHT expires or might be lost due to churn. Furthermore, GNUnet's DHT typically requires multiple rounds of PUT operations before a key-value pair is consistently available to all peers (the DHT randomizes paths and thus storage locations, and only after multiple rounds of PUTs there will be a sufficient number of replicas in large DHTs). An explicit PUT operation using the DHT API will only cause network traffic once, so in order to ensure basic availability and resistance to churn (and adversaries), PUTs must be repeated. While the exact frequency depends on the application, a rule of thumb is that there should be at least a dozen PUT operations within the content lifetime. Content in the DHT typically expires after one day, so DHT PUT operations should be repeated at least every 1-2 hours.

5.33.2.3 MONITOR

The DHT API also allows applications to monitor messages crossing the local DHT service. The types of messages used by the DHT are GET, PUT and RESULT messages. Using the monitoring API, applications can choose to monitor these requests, possibly limiting themselves to requests for a particular block type.

The monitoring API is not only useful for diagnostics, it can also be used to trigger application operations based on PUT operations. For example, an application may use

PUTs to distribute work requests to other peers. The workers would then monitor for PUTs that give them work, instead of looking for work using GET operations. This can be beneficial, especially if the workers have no good way to guess the keys under which work would be stored. Naturally, additional protocols might be needed to ensure that the desired number of workers will process the distributed workload.

5.33.2.4 DHT Routing Options

There are two important options for GET and PUT requests:

GNUNET_DHT_RO_DEMULITPLEX_EVERYWHERE This option means that all
: peers should process the request, even if their peer ID is not closest to the key. For a PUT request, this means that all peers that a request traverses may make a copy of the data. Similarly for a GET request, all peers will check their local database for a result. Setting this option can thus significantly improve caching and reduce bandwidth consumption — at the expense of a larger DHT database. If in doubt, we recommend that this option should be used.

GNUNET_DHT_RO_RECORD_ROUTE This option instructs the DHT to record
: the path that a GET or a PUT request is taking through the overlay network. The resulting paths are then returned to the application with the respective result. This allows the receiver of a result to construct a path to the originator of the data, which might then be used for routing. Naturally, setting this option requires additional bandwidth and disk space, so applications should only set this if the paths are needed by the application logic.

GNUNET_DHT_RO_FIND_PEER This option is an internal option used by
: the DHT's peer discovery mechanism and should not be used by applications.

GNUNET_DHT_RO_BART This option is currently not implemented. It may
: in the future offer performance improvements for clique topologies.

5.33.3 The DHT Client-Service Protocol

5.33.3.1 PUTting data into the DHT

To store (PUT) data into the DHT, the client sends a `struct GNUNET_DHT_ClientPutMessage` to the service. This message specifies the block type, routing options, the desired replication level, the expiration time, key, value and a 64-bit unique ID for the operation. The service responds with a `struct GNUNET_DHT_ClientPutConfirmationMessage` with the same 64-bit unique ID. Note that the service sends the confirmation as soon as it has locally processed the PUT request. The PUT may still be propagating through the network at this time.

In the future, we may want to change this to provide (limited) feedback to the client, for example if we detect that the PUT operation had no effect because the same key-value pair was already stored in the DHT. However, changing this would also require additional state and messages in the P2P interaction.

5.33.3.2 GETting data from the DHT

To retrieve (GET) data from the DHT, the client sends a `struct GNUNET_DHT_ClientGetMessage` to the service. The message specifies routing options, a replication

level (for replicating the GET, not the content), the desired block type, the key, the (optional) extended query and unique 64-bit request ID.

Additionally, the client may send any number of `struct GNUNET_DHT_ClientGetResultSeenMessages` to notify the service about results that the client is already aware of. These messages consist of the key, the unique 64-bit ID of the request, and an arbitrary number of hash codes over the blocks that the client is already aware of. As messages are restricted to 64k, a client that already knows more than about a thousand blocks may need to send several of these messages. Naturally, the client should transmit these messages as quickly as possible after the original GET request such that the DHT can filter those results in the network early on. Naturally, as these messages are sent after the original request, it is conceivalbe that the DHT service may return blocks that match those already known to the client anyway.

In response to a GET request, the service will send `struct GNUNET_DHT_ClientResultMessages` to the client. These messages contain the block type, expiration, key, unique ID of the request and of course the value (a block). Depending on the options set for the respective operations, the replies may also contain the path the GET and/or the PUT took through the network.

A client can stop receiving replies either by disconnecting or by sending a `struct GNUNET_DHT_ClientGetStopMessage` which must contain the key and the 64-bit unique ID of the original request. Using an explicit "stop" message is more common as this allows a client to run many concurrent GET operations over the same connection with the DHT service — and to stop them individually.

5.33.3.3 Monitoring the DHT

To begin monitoring, the client sends a `struct GNUNET_DHT_MonitorStartStop` message to the DHT service. In this message, flags can be set to enable (or disable) monitoring of GET, PUT and RESULT messages that pass through a peer. The message can also restrict monitoring to a particular block type or a particular key. Once monitoring is enabled, the DHT service will notify the client about any matching event using `struct GNUNET_DHT_MonitorGetMessages` for GET events, `struct GNUNET_DHT_MonitorPutMessage` for PUT events and `struct GNUNET_DHT_MonitorGetRespMessage` for RESULTs. Each of these messages contains all of the information about the event.

5.33.4 The DHT Peer-to-Peer Protocol

5.33.4.1 Routing GETs or PUTs

When routing GETs or PUTs, the DHT service selects a suitable subset of neighbours for forwarding. The exact number of neighbours can be zero or more and depends on the hop counter of the query (initially zero) in relation to the (log of) the network size estimate, the desired replication level and the peer's connectivity. Depending on the hop counter and our network size estimate, the selection of the peers maybe randomized or by proximity to the key. Furthermore, requests include a set of peers that a request has already traversed; those peers are also excluded from the selection.

5.33.4.2 PUTting data into the DHT2

To PUT data into the DHT, the service sends a `struct PeerPutMessage` of type `GNUNET_MESSAGE_TYPE_DHT_P2P_PUT` to the respective neighbour. In addition to the usual information about the content (type, routing options, desired replication level for the content, expiration time, key and value), the message contains a fixed-size Bloom filter with information about which peers (may) have already seen this request. This Bloom filter is used to ensure that DHT messages never loop back to a peer that has already processed the request. Additionally, the message includes the current hop counter and, depending on the routing options, the message may include the full path that the message has taken so far. The Bloom filter should already contain the identity of the previous hop; however, the path should not include the identity of the previous hop and the receiver should append the identity of the sender to the path, not its own identity (this is done to reduce bandwidth).

5.33.4.3 GETting data from the DHT2

A peer can search the DHT by sending `struct PeerGetMessage`s of type `GNUNET_MESSAGE_TYPE_DHT_P2P_GET` to other peers. In addition to the usual information about the request (type, routing options, desired replication level for the request, the key and the extended query), a GET request also contains a hop counter, a Bloom filter over the peers that have processed the request already and depending on the routing options the full path traversed by the GET. Finally, a GET request includes a variable-size second Bloom filter and a so-called Bloom filter mutator value which together indicate which replies the sender has already seen. During the lookup, each block that matches they block type, key and extended query is additionally subjected to a test against this Bloom filter. The block plugin is expected to take the hash of the block and combine it with the mutator value and check if the result is not yet in the Bloom filter. The originator of the query will from time to time modify the mutator to (eventually) allow false-positives filtered by the Bloom filter to be returned.

Peers that receive a GET request perform a local lookup (depending on their proximity to the key and the query options) and forward the request to other peers. They then remember the request (including the Bloom filter for blocking duplicate results) and when they obtain a matching, non-filtered response a `struct PeerResultMessage` of type `GNUNET_MESSAGE_TYPE_DHT_P2P_RESULT` is forwarded to the previous hop. Whenver a result is forwarded, the block plugin is used to update the Bloom filter accordingly, to ensure that the same result is never forwarded more than once. The DHT service may also cache forwarded results locally if the "CACHE_RESULTS" option is set to "YES" in the configuration.

5.34 GNU Name System (GNS)

The GNU Name System (GNS) is a decentralized database that enables users to securely resolve names to values. Names can be used to identify other users (for example, in social networking), or network services (for example, VPN services running at a peer in GNUnet, or purely IP-based services on the Internet). Users interact with GNS by typing in a hostname that ends in a top-level domain that is configured in the “GNS” section, matches an identity of the user or ends in a Base32-encoded public key.

Videos giving an overview of most of the GNS and the motivations behind it is available here and here. The remainder of this chapter targets developers that are familiar with high level concepts of GNS as presented in these talks.

GNS-aware applications should use the GNS resolver to obtain the respective records that are stored under that name in GNS. Each record consists of a type, value, expiration time and flags.

The type specifies the format of the value. Types below 65536 correspond to DNS record types, larger values are used for GNS-specific records. Applications can define new GNS record types by reserving a number and implementing a plugin (which mostly needs to convert the binary value representation to a human-readable text format and vice-versa). The expiration time specifies how long the record is to be valid. The GNS API ensures that applications are only given non-expired values. The flags are typically irrelevant for applications, as GNS uses them internally to control visibility and validity of records.

Records are stored along with a signature. The signature is generated using the private key of the authoritative zone. This allows any GNS resolver to verify the correctness of a name-value mapping.

Internally, GNS uses the NAMECACHE to cache information obtained from other users, the NAMESTORE to store information specific to the local users, and the DHT to exchange data between users. A plugin API is used to enable applications to define new GNS record types.

5.34.1 libgnunetgns

The GNS API itself is extremely simple. Clients first connec to the GNS service using `GNUNET_GNS_connect`. They can then perform lookups using `GNUNET_GNS_lookup` or cancel pending lookups using `GNUNET_GNS_lookup_cancel`. Once finished, clients disconnect using `GNUNET_GNS_disconnect`.

5.34.1.1 Looking up records

`GNUNET_GNS_lookup` takes a number of arguments:

handle This is simply the GNS connection handle from
: `GNUNET_GNS_connect`.

name The client needs to specify the name to
: be resolved. This can be any valid DNS or GNS hostname.

zone The client
: needs to specify the public key of the GNS zone against which the resolution should be done. Note that a key must be provided, the client should look up plausible values using its configuration, the identity service and by attempting to interpret the TLD as a base32-encoded public key.

type This is the desired GNS or DNS record type
: to look for. While all records for the given name will be returned, this can be important if the client wants to resolve record types that themselves delegate resolution, such as CNAME, PKEY or GNS2DNS. Resolving a record of any of these types will only work if the respective record type is specified in the request, as the GNS resolver will otherwise follow the delegation and return the records from the respective destination, instead of the delegating record.

only_cached This argument should typically be set to
: `GNUNET_NO`. Setting it to `GNUNET_YES` disables resolution via the overlay network.

shorten_zone_key If GNS encounters new names during resolution,
: their respective zones can automatically be learned and added to the "shorten zone". If this is desired, clients must pass the private key of the shorten zone. If NULL is passed, shortening is disabled.

proc This argument identifies
: the function to call with the result. It is given proc_cls, the number of records found (possilby zero) and the array of the records as arguments. proc will only be called once. After proc,> has been called, the lookup must no longer be cancelled.

proc_cls The closure for proc.

5.34.1.2 Accessing the records

The `libgnunetgnsrecord` library provides an API to manipulate the GNS record array that is given to proc. In particular, it offers functions such as converting record values to human-readable strings (and back). However, most `libgnunetgnsrecord` functions are not interesting to GNS client applications.

For DNS records, the `libgnunetdnsparser` library provides functions for parsing (and serializing) common types of DNS records.

5.34.1.3 Creating records

Creating GNS records is typically done by building the respective record information (possibly with the help of `libgnunetgnsrecord` and `libgnunetdnsparser`) and then using the `libgnunetnamestore` to publish the information. The GNS API is not involved in this operation.

5.34.1.4 Future work

In the future, we want to expand `libgnunetgns` to allow applications to observe shortening operations performed during GNS resolution, for example so that users can receive visual feedback when this happens.

5.34.2 libgnunetgnsrecord

The `libgnunetgnsrecord` library is used to manipulate GNS records (in plaintext or in their encrypted format). Applications mostly interact with `libgnunetgnsrecord` by using the functions to convert GNS record values to strings or vice-versa, or to lookup a GNS record type number by name (or vice-versa). The library also provides various other functions that are mostly used internally within GNS, such as converting keys to names, checking for expiration, encrypting GNS records to GNS blocks, verifying GNS block signatures and decrypting GNS records from GNS blocks.

We will now discuss the four commonly used functions of the API. `libgnunetgnsrecord` does not perform these operations itself, but instead uses plugins to perform the operation. GNUnet includes plugins to support common DNS record types as well as standard GNS record types.

5.34.2.1 Value handling

GNUNET_GNSRECORD_value_to_string can be used to convert the (binary) representation of a GNS record value to a human readable, 0-terminated UTF-8 string. NULL is returned if the specified record type is not supported by any available plugin.

GNUNET_GNSRECORD_string_to_value can be used to try to convert a human readable string to the respective (binary) representation of a GNS record value.

5.34.2.2 Type handling

GNUNET_GNSRECORD_typename_to_number can be used to obtain the numeric value associated with a given typename. For example, given the typename "A" (for DNS A reocrds), the function will return the number 1. A list of common DNS record types is here (http://en.wikipedia.org/wiki/List_of_DNS_record_types). Note that not all DNS record types are supported by GNUnet GNSRECORD plugins at this time.

GNUNET_GNSRECORD_number_to_typename can be used to obtain the typename associated with a given numeric value. For example, given the type number 1, the function will return the typename "A".

5.34.3 GNS plugins

Adding a new GNS record type typically involves writing (or extending) a GNSRECORD plugin. The plugin needs to implement the gnunet_gnsrecord_plugin.h API which provides basic functions that are needed by GNSRECORD to convert typenames and values of the respective record type to strings (and back). These gnsrecord plugins are typically implemented within their respective subsystems. Examples for such plugins can be found in the GNSRECORD, GNS and CONVERSATION subsystems.

The libgnunetgnsrecord library is then used to locate, load and query the appropriate gnsrecord plugin. Which plugin is appropriate is determined by the record type (which is just a 32-bit integer). The libgnunetgnsrecord library loads all block plugins that are installed at the local peer and forwards the application request to the plugins. If the record type is not supported by the plugin, it should simply return an error code.

The central functions of the block APIs (plugin and main library) are the same four functions for converting between values and strings, and typenames and numbers documented in the previous subsection.

5.34.4 The GNS Client-Service Protocol

The GNS client-service protocol consists of two simple messages, the LOOKUP message and the LOOKUP_RESULT. Each LOOKUP message contains a unique 32-bit identifier, which will be included in the corresponding response. Thus, clients can send many lookup requests in parallel and receive responses out-of-order. A LOOKUP request also includes the public key of the GNS zone, the desired record type and fields specifying whether shortening is enabled or networking is disabled. Finally, the LOOKUP message includes the name to be resolved.

The response includes the number of records and the records themselves in the format created by GNUNET_GNSRECORD_records_serialize. They can thus be deserialized using GNUNET_GNSRECORD_records_deserialize.

5.34.5 Hijacking the DNS-Traffic using gnunet-service-dns

This section documents how the gnunet-service-dns (and the gnunet-helper-dns) intercepts DNS queries from the local system. This is merely one method for how we can obtain GNS queries. It is also possible to change `resolv.conf` to point to a machine running `gnunet-dns2gns` or to modify libc's name system switch (NSS) configuration to include a GNS resolution plugin. The method described in this chaper is more of a last-ditch catch-all approach.

`gnunet-service-dns` enables intercepting DNS traffic using policy based routing. We MARK every outgoing DNS-packet if it was not sent by our application. Using a second routing table in the Linux kernel these marked packets are then routed through our virtual network interface and can thus be captured unchanged.

Our application then reads the query and decides how to handle it. If the query can be addressed via GNS, it is passed to `gnunet-service-gns` and resolved internally using GNS. In the future, a reverse query for an address of the configured virtual network could be answered with records kept about previous forward queries. Queries that are not hijacked by some application using the DNS service will be sent to the original recipient. The answer to the query will always be sent back through the virtual interface with the original nameserver as source address.

5.34.5.1 Network Setup Details

The DNS interceptor adds the following rules to the Linux kernel:

```
iptables -t mangle -I OUTPUT 1 -p udp --sport $LOCALPORT --dport 53 \
-j ACCEPT iptables -t mangle -I OUTPUT 2 -p udp --dport 53 -j MARK \
--set-mark 3 ip rule add fwmark 3 table2 ip route add default via \
$VIRTUALDNS table2
```

Line 1 makes sure that all packets coming from a port our application opened beforehand (`$LOCALPORT`) will be routed normally. Line 2 marks every other packet to a DNS-Server with mark 3 (chosen arbitrarily). The third line adds a routing policy based on this mark 3 via the routing table.

5.34.6 Serving DNS lookups via GNS on W32

This section documents how the libw32nsp (and gnunet-gns-helper-service-w32) do DNS resolutions of DNS queries on the local system. This only applies to GNUnet running on W32.

W32 has a concept of "Namespaces" and "Namespace providers". These are used to present various name systems to applications in a generic way. Namespaces include DNS, mDNS, NLA and others. For each namespace any number of providers could be registered, and they are queried in an order of priority (which is adjustable).

Applications can resolve names by using WSALookupService*() family of functions.

However, these are WSA-only facilities. Common BSD socket functions for namespace resolutions are gethostbyname and getaddrinfo (among others). These functions are implemented internally (by default - by mswsock, which also implements the default DNS provider) as wrappers around WSALookupService*() functions (see "Sample Code for a Service Provider" on MSDN).

On W32 GNUnet builds a libw32nsp - a namespace provider, which can then be installed into the system by using w32nsp-install (and uninstalled by w32nsp-uninstall), as described in "Installation Handbook".

libw32nsp is very simple and has almost no dependencies. As a response to NSPLookupServiceBegin(), it only checks that the provider GUID passed to it by the caller matches GNUnet DNS Provider GUID, then connects to gnunet-gns-helper-service-w32 at 127.0.0.1:5353 (hardcoded) and sends the name resolution request there, returning the connected socket to the caller.

When the caller invokes NSPLookupServiceNext(), libw32nsp reads a completely formed reply from that socket, unmarshalls it, then gives it back to the caller.

At the moment gnunet-gns-helper-service-w32 is implemented to ever give only one reply, and subsequent calls to NSPLookupServiceNext() will fail with WSA_NODATA (first call to NSPLookupServiceNext() might also fail if GNS failed to find the name, or there was an error connecting to it).

gnunet-gns-helper-service-w32 does most of the processing:

- Maintains a connection to GNS.
- Reads GNS config and loads appropriate keys.
- Checks service GUID and decides on the type of record to look up, refusing to make a lookup outright when unsupported service GUID is passed.
- Launches the lookup

When lookup result arrives, gnunet-gns-helper-service-w32 forms a complete reply (including filling a WSAQUERYSETW structure and, possibly, a binary blob with a hostent structure for gethostbyname() client), marshalls it, and sends it back to libw32nsp. If no records were found, it sends an empty header.

This works for most normal applications that use gethostbyname() or getaddrinfo() to resolve names, but fails to do anything with applications that use alternative means of resolving names (such as sending queries to a DNS server directly by themselves). This includes some of well known utilities, like "ping" and "nslookup".

5.35 GNS Namecache

The NAMECACHE subsystem is responsible for caching (encrypted) resolution results of the GNU Name System (GNS). GNS makes zone information available to other users via the DHT. However, as accessing the DHT for every lookup is expensive (and as the DHT's local cache is lost whenever the peer is restarted), GNS uses the NAMECACHE as a more persistent cache for DHT lookups. Thus, instead of always looking up every name in the DHT, GNS first checks if the result is already available locally in the NAMECACHE. Only if there is no result in the NAMECACHE, GNS queries the DHT. The NAMECACHE stores data in the same (encrypted) format as the DHT. It thus makes no sense to iterate over all items in the NAMECACHE — the NAMECACHE does not have a way to provide the keys required to decrypt the entries.

Blocks in the NAMECACHE share the same expiration mechanism as blocks in the DHT — the block expires wheneever any of the records in the (encrypted) block expires. The expiration time of the block is the only information stored in plaintext. The NAMECACHE

service internally performs all of the required work to expire blocks, clients do not have to worry about this. Also, given that NAMECACHE stores only GNS blocks that local users requested, there is no configuration option to limit the size of the NAMECACHE. It is assumed to be always small enough (a few MB) to fit on the drive.

The NAMECACHE supports the use of different database backends via a plugin API.

5.35.1 libgnunetnamecache

The NAMECACHE API consists of five simple functions. First, there is `GNUNET_NAMECACHE_connect` to connect to the NAMECACHE service. This returns the handle required for all other operations on the NAMECACHE. Using `GNUNET_NAMECACHE_block_cache` clients can insert a block into the cache. `GNUNET_NAMECACHE_lookup_block` can be used to lookup blocks that were stored in the NAMECACHE. Both operations can be cancelled using `GNUNET_NAMECACHE_cancel`. Note that cancelling a `GNUNET_NAMECACHE_block_cache` operation can result in the block being stored in the NAMECACHE — or not. Cancellation primarily ensures that the continuation function with the result of the operation will no longer be invoked. Finally, `GNUNET_NAMECACHE_disconnect` closes the connection to the NAMECACHE.

The maximum size of a block that can be stored in the NAMECACHE is `GNUNET_NAMECACHE_MAX_VALUE_SIZE`, which is defined to be 63 kB.

5.35.2 The NAMECACHE Client-Service Protocol

All messages in the NAMECACHE IPC protocol start with the `struct GNUNET_NAMECACHE_Header` which adds a request ID (32-bit integer) to the standard message header. The request ID is used to match requests with the respective responses from the NAMECACHE, as they are allowed to happen out-of-order.

5.35.2.1 Lookup

The `struct LookupBlockMessage` is used to lookup a block stored in the cache. It contains the query hash. The NAMECACHE always responds with a `struct LookupBlockResponseMessage`. If the NAMECACHE has no response, it sets the expiration time in the response to zero. Otherwise, the response is expected to contain the expiration time, the ECDSA signature, the derived key and the (variable-size) encrypted data of the block.

5.35.2.2 Store

The `struct BlockCacheMessage` is used to cache a block in the NAMECACHE. It has the same structure as the `struct LookupBlockResponseMessage`. The service responds with a `struct BlockCacheResponseMessage` which contains the result of the operation (success or failure). In the future, we might want to make it possible to provide an error message as well.

5.35.3 The NAMECACHE Plugin API

The NAMECACHE plugin API consists of two functions, `cache_block` to store a block in the database, and `lookup_block` to lookup a block in the database.

5.35.3.1 Lookup2

The `lookup_block` function is expected to return at most one block to the iterator, and return `GNUNET_NO` if there were no non-expired results. If there are multiple non-expired results in the cache, the lookup is supposed to return the result with the largest expiration time.

5.35.3.2 Store2

The `cache_block` function is expected to try to store the block in the database, and return `GNUNET_SYSERR` if this was not possible for any reason. Furthermore, `cache_block` is expected to implicitly perform cache maintenance and purge blocks from the cache that have expired. Note that `cache_block` might encounter the case where the database already has another block stored under the same key. In this case, the plugin must ensure that the block with the larger expiration time is preserved. Obviously, this can done either by simply adding new blocks and selecting for the most recent expiration time during lookup, or by checking which block is more recent during the store operation.

5.36 REVOCATION Subsystem

The REVOCATION subsystem is responsible for key revocation of Egos. If a user learns that theis private key has been compromised or has lost it, they can use the REVOCATION system to inform all of the other users that their private key is no longer valid. The subsystem thus includes ways to query for the validity of keys and to propagate revocation messages.

5.36.1 Dissemination

When a revocation is performed, the revocation is first of all disseminated by flooding the overlay network. The goal is to reach every peer, so that when a peer needs to check if a key has been revoked, this will be purely a local operation where the peer looks at his local revocation list. Flooding the network is also the most robust form of key revocation — an adversary would have to control a separator of the overlay graph to restrict the propagation of the revocation message. Flooding is also very easy to implement — peers that receive a revocation message for a key that they have never seen before simply pass the message to all of their neighbours.

Flooding can only distribute the revocation message to peers that are online. In order to notify peers that join the network later, the revocation service performs efficient set reconciliation over the sets of known revocation messages whenever two peers (that both support REVOCATION dissemination) connect. The SET service is used to perform this operation efficiently.

5.36.2 Revocation Message Design Requirements

However, flooding is also quite costly, creating O(`|E|`) messages on a network with `|E|` edges. Thus, revocation messages are required to contain a proof-of-work, the result of an expensive computation (which, however, is cheap to verify). Only peers that have expended the CPU time necessary to provide this proof will be able to flood the network with the revocation message. This ensures that an attacker cannot simply flood the network with millions of revocation messages. The proof-of-work required by GNUnet is set to take days

on a typical PC to compute; if the ability to quickly revoke a key is needed, users have the option to pre-compute revocation messages to store off-line and use instantly after their key has expired.

Revocation messages must also be signed by the private key that is being revoked. Thus, they can only be created while the private key is in the possession of the respective user. This is another reason to create a revocation message ahead of time and store it in a secure location.

5.36.3 libgnunetrevocation

The REVOCATION API consists of two parts, to query and to issue revocations.

5.36.3.1 Querying for revoked keys

`GNUNET_REVOCATION_query` is used to check if a given ECDSA public key has been revoked. The given callback will be invoked with the result of the check. The query can be cancelled using `GNUNET_REVOCATION_query_cancel` on the return value.

5.36.3.2 Preparing revocations

It is often desirable to create a revocation record ahead-of-time and store it in an off-line location to be used later in an emergency. This is particularly true for GNUnet revocations, where performing the revocation operation itself is computationally expensive and thus is likely to take some time. Thus, if users want the ability to perform revocations quickly in an emergency, they must pre-compute the revocation message. The revocation API enables this with two functions that are used to compute the revocation message, but not trigger the actual revocation operation.

`GNUNET_REVOCATION_check_pow` should be used to calculate the proof-of-work required in the revocation message. This function takes the public key, the required number of bits for the proof of work (which in GNUnet is a network-wide constant) and finally a proof-of-work number as arguments. The function then checks if the given proof-of-work number is a valid proof of work for the given public key. Clients preparing a revocation are expected to call this function repeatedly (typically with a monotonically increasing sequence of numbers of the proof-of-work number) until a given number satisfies the check. That number should then be saved for later use in the revocation operation.

`GNUNET_REVOCATION_sign_revocation` is used to generate the signature that is required in a revocation message. It takes the private key that (possibly in the future) is to be revoked and returns the signature. The signature can again be saved to disk for later use, which will then allow performing a revocation even without access to the private key.

5.36.3.3 Issuing revocations

Given a ECDSA public key, the signature from `GNUNET_REVOCATION_sign` and the proof-of-work, `GNUNET_REVOCATION_revoke` can be used to perform the actual revocation. The given callback is called upon completion of the operation. `GNUNET_REVOCATION_revoke_cancel` can be used to stop the library from calling the continuation; however, in that case it is undefined whether or not the revocation operation will be executed.

5.36.4 The REVOCATION Client-Service Protocol

The REVOCATION protocol consists of four simple messages.

A `QueryMessage` containing a public ECDSA key is used to check if a particular key has been revoked. The service responds with a `QueryResponseMessage` which simply contains a bit that says if the given public key is still valid, or if it has been revoked.

The second possible interaction is for a client to revoke a key by passing a `RevokeMessage` to the service. The `RevokeMessage` contains the ECDSA public key to be revoked, a signature by the corresponding private key and the proof-of-work, The service responds with a `RevocationResponseMessage` which can be used to indicate that the `RevokeMessage` was invalid (i.e. proof of work incorrect), or otherwise indicates that the revocation has been processed successfully.

5.36.5 The REVOCATION Peer-to-Peer Protocol

Revocation uses two disjoint ways to spread revocation information among peers. First of all, P2P gossip exchanged via CORE-level neighbours is used to quickly spread revocations to all connected peers. Second, whenever two peers (that both support revocations) connect, the SET service is used to compute the union of the respective revocation sets.

In both cases, the exchanged messages are `RevokeMessage`s which contain the public key that is being revoked, a matching ECDSA signature, and a proof-of-work. Whenever a peer learns about a new revocation this way, it first validates the signature and the proof-of-work, then stores it to disk (typically to a file $GNUNET_DATA_HOME/revocation.dat) and finally spreads the information to all directly connected neighbours.

For computing the union using the SET service, the peer with the smaller hashed peer identity will connect (as a "client" in the two-party set protocol) to the other peer after one second (to reduce traffic spikes on connect) and initiate the computation of the set union. All revocation services use a common hash to identify the SET operation over revocation sets.

The current implementation accepts revocation set union operations from all peers at any time; however, well-behaved peers should only initiate this operation once after establishing a connection to a peer with a larger hashed peer identity.

5.37 File-sharing (FS) Subsystem

This chapter describes the details of how the file-sharing service works. As with all services, it is split into an API (libgnunetfs), the service process (gnunet-service-fs) and user interface(s). The file-sharing service uses the datastore service to store blocks and the DHT (and indirectly datacache) for lookups for non-anonymous file-sharing. Furthermore, the file-sharing service uses the block library (and the block fs plugin) for validation of DHT operations.

In contrast to many other services, libgnunetfs is rather complex since the client library includes a large number of high-level abstractions; this is necessary since the Fs service itself largely only operates on the block level. The FS library is responsible for providing a file-based abstraction to applications, including directories, meta data, keyword search, verification, and so on.

The method used by GNUnet to break large files into blocks and to use keyword search is called the "Encoding for Censorship Resistant Sharing" (ECRS). ECRS is largely implemented in the fs library; block validation is also reflected in the block FS plugin and the FS service. ECRS on-demand encoding is implemented in the FS service.

NOTE: The documentation in this chapter is quite incomplete.

5.37.1 Encoding for Censorship-Resistant Sharing (ECRS)

When GNUnet shares files, it uses a content encoding that is called ECRS, the Encoding for Censorship-Resistant Sharing. Most of ECRS is described in the (so far unpublished) research paper attached to this page. ECRS obsoletes the previous ESED and ESED II encodings which were used in GNUnet before version 0.7.0. The rest of this page assumes that the reader is familiar with the attached paper. What follows is a description of some minor extensions that GNUnet makes over what is described in the paper. The reason why these extensions are not in the paper is that we felt that they were obvious or trivial extensions to the original scheme and thus did not warrant space in the research report.

5.37.1.1 Namespace Advertisements

An `SBlock` with identifier all zeros is a signed advertisement for a namespace. This special `SBlock` contains metadata describing the content of the namespace. Instead of the name of the identifier for a potential update, it contains the identifier for the root of the namespace. The URI should always be empty. The `SBlock` is signed with the content provder's RSA private key (just like any other SBlock). Peers can search for `SBlock`s in order to find out more about a namespace.

5.37.1.2 KSBlocks

GNUnet implements `KSBlocks` which are `KBlocks` that, instead of encrypting a CHK and metadata, encrypt an `SBlock` instead. In other words, `KSBlocks` enable GNUnet to find `SBlocks` using the global keyword search. Usually the encrypted `SBlock` is a namespace advertisement. The rationale behind `KSBlock`s and `SBlock`s is to enable peers to discover namespaces via keyword searches, and, to associate useful information with namespaces. When GNUnet finds `KSBlocks` during a normal keyword search, it adds the information to an internal list of discovered namespaces. Users looking for interesting namespaces can then inspect this list, reducing the need for out-of-band discovery of namespaces. Naturally, namespaces (or more specifically, namespace advertisements) can also be referenced from directories, but `KSBlock`s should make it easier to advertise namespaces for the owner of the pseudonym since they eliminate the need to first create a directory.

Collections are also advertised using `KSBlock`s.

5.37.2 File-sharing persistence directory structure

This section documents how the file-sharing library implements persistence of file-sharing operations and specifically the resulting directory structure. This code is only active if the `GNUNET_FS_FLAGS_PERSISTENCE` flag was set when calling `GNUNET_FS_start`. In this case, the file-sharing library will try hard to ensure that all major operations (searching, downloading, publishing, unindexing) are persistent, that is, can live longer than the process itself. More specifically, an operation is supposed to live until it is explicitly stopped.

If `GNUNET_FS_stop` is called before an operation has been stopped, a `SUSPEND` event is generated and then when the process calls `GNUNET_FS_start` next time, a `RESUME` event is generated. Additionally, even if an application crashes (segfault, SIGKILL, system crash) and hence `GNUNET_FS_stop` is never called and no `SUSPEND` events are generated, operations are still resumed (with `RESUME` events). This is implemented by constantly writing the

current state of the file-sharing operations to disk. Specifically, the current state is always written to disk whenever anything significant changes (the exception are block-wise progress in publishing and unindexing, since those operations would be slowed down significantly and can be resumed cheaply even without detailed accounting). Note that if the process crashes (or is killed) during a serialization operation, FS does not guarantee that this specific operation is recoverable (no strict transactional semantics, again for performance reasons). However, all other unrelated operations should resume nicely.

Since we need to serialize the state continuously and want to recover as much as possible even after crashing during a serialization operation, we do not use one large file for serialization. Instead, several directories are used for the various operations. When `GNUNET_FS_start` executes, the master directories are scanned for files describing operations to resume. Sometimes, these operations can refer to related operations in child directories which may also be resumed at this point. Note that corrupted files are cleaned up automatically. However, dangling files in child directories (those that are not referenced by files from the master directories) are not automatically removed.

Persistence data is kept in a directory that begins with the "STATE_DIR" prefix from the configuration file (by default, "$SERVICEHOME/persistence/") followed by the name of the client as given to `GNUNET_FS_start` (for example, "gnunet-gtk") followed by the actual name of the master or child directory.

The names for the master directories follow the names of the operations:

- "search"
- "download"
- "publish"
- "unindex"

Each of the master directories contains names (chosen at random) for each active top-level (master) operation. Note that a download that is associated with a search result is not a top-level operation.

In contrast to the master directories, the child directories are only consulted when another operation refers to them. For each search, a subdirectory (named after the master search synchronization file) contains the search results. Search results can have an associated download, which is then stored in the general "download-child" directory. Downloads can be recursive, in which case children are stored in subdirectories mirroring the structure of the recursive download (either starting in the master "download" directory or in the "download-child" directory depending on how the download was initiated). For publishing operations, the "publish-file" directory contains information about the individual files and directories that are part of the publication. However, this directory structure is flat and does not mirror the structure of the publishing operation. Note that unindex operations cannot have associated child operations.

5.38 REGEX Subsystem

Using the REGEX subsystem, you can discover peers that offer a particular service using regular expressions. The peers that offer a service specify it using a regular expressions. Peers that want to patronize a service search using a string. The REGEX subsystem will then use the DHT to return a set of matching offerers to the patrons.

For the technical details, we have Max's defense talk and Max's Master's thesis.

5.38.1 How to run the regex profiler

The gnunet-regex-profiler can be used to profile the usage of mesh/regex for a given set of regular expressions and strings. Mesh/regex allows you to announce your peer ID under a certain regex and search for peers matching a particular regex using a string. See szengel2012ms (`https://gnunet.org/szengel2012ms`) for a full introduction.

First of all, the regex profiler uses GNUnet testbed, thus all the implications for testbed also apply to the regex profiler (for example you need password-less ssh login to the machines listed in your hosts file).

Configuration

Moreover, an appropriate configuration file is needed. Generally you can refer to the `contrib/regex_profiler_infiniband.conf` file in the sourcecode of GNUnet for an example configuration. In the following paragraph the important details are highlighted.

Announcing of the regular expressions is done by the gnunet-daemon-regexprofiler, therefore you have to make sure it is started, by adding it to the AUTOSTART set of ARM:

```
[regexprofiler]
AUTOSTART = YES
```

Furthermore you have to specify the location of the binary:

```
[regexprofiler]
# Location of the gnunet-daemon-regexprofiler binary.
BINARY = /home/szengel/gnunet/src/mesh/.libs/gnunet-daemon-regexprofiler
# Regex prefix that will be applied to all regular expressions and
# search string.
REGEX_PREFIX = "GNVPN-0001-PAD"
```

When running the profiler with a large scale deployment, you probably want to reduce the workload of each peer. Use the following options to do this.

```
[dht]
# Force network size estimation
FORCE_NSE = 1

[dhtcache]
DATABASE = heap
# Disable RC-file for Bloom filter? (for benchmarking with limited IO
# availability)
DISABLE_BF_RC = YES
# Disable Bloom filter entirely
DISABLE_BF = YES

[nse]
# Minimize proof-of-work CPU consumption by NSE
WORKBITS = 1
```

Options

To finally run the profiler some options and the input data need to be specified on the command line.

```
gnunet-regex-profiler -c config-file -d log-file -n num-links \
-p path-compression-length -s search-delay -t matching-timeout \
-a num-search-strings hosts-file policy-dir search-strings-file
```

Where...

- ... `config-file` means the configuration file created earlier.
- ... `log-file` is the file where to write statistics output.
- ... `num-links` indicates the number of random links between started peers.
- ... `path-compression-length` is the maximum path compression length in the DFA.
- ... `search-delay` time to wait between peers finished linking and starting to match strings.
- ... `matching-timeout` timeout after which to cancel the searching.
- ... `num-search-strings` number of strings in the search-strings-file.
- ... the `hosts-file` should contain a list of hosts for the testbed, one per line in the following format:
  - `user@host_ip:port`
- ... the `policy-dir` is a folder containing text files containing one or more regular expressions. A peer is started for each file in that folder and the regular expressions in the corresponding file are announced by this peer.
- ... the `search-strings-file` is a text file containing search strings, one in each line.

You can create regular expressions and search strings for every AS in the Internet using the attached scripts. You need one of the CAIDA routeviews prefix2as (`http://data.caida.org/datasets/routing/routeviews-prefix2as/`) data files for this. Run

```
create_regex.py <filename> <output path>
```

to create the regular expressions and

```
create_strings.py <input path> <outfile>
```

to create a search strings file from the previously created regular expressions.

5.39 REST Subsystem

Using the REST subsystem, you can expose REST-based APIs or services. The REST service is designed as a pluggable architecture. To create a new REST endpoint, simply add a library in the form "plugin_rest_*". The REST service will automatically load all REST plugins on startup.

Configuration

The REST service can be configured in various ways. The reference config file can be found in `src/rest/rest.conf`:

```
[rest]
REST_PORT=7776
REST_ALLOW_HEADERS=Authorization,Accept,Content-Type
REST_ALLOW_ORIGIN=*
REST_ALLOW_CREDENTIALS=true
```

The port as well as

`(CORS)` [cross-origin resource sharing]
headers that are supposed to be advertised by the rest service are configurable.

5.39.1 Namespace considerations

The `gnunet-rest-service` will load all plugins that are installed. As such it is important that the endpoint namespaces do not clash.

For example, plugin X might expose the endpoint "/xxx" while plugin Y exposes endpoint "/xxx/yyy". This is a problem if plugin X is also supposed to handle a call to "/xxx/yyy". Currently the REST service will not complain or warn about such clashes, so please make sure that endpoints are unambiguous.

5.39.2 Endpoint documentation

This is WIP. Endpoints should be documented appropriately. Perferably using annotations.

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with... Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B GNU General Public License

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.

The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

   Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
   c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.

However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so

available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

Concept Index

A

Accounting to Encourage Resource Sharing 6
Anonymity 7
ARM 109
ATS Subsystem 124
Authentication 5
Automatic Restart Manager 109

B

Bluetooth plugin 118
Building GNUnet 76

C

CADET Subsystem 130
chk-uri 25
Coding style 72
compiling libgnurl 77
Confidentiality 7
CONTAINER_MDLL API 108
copyright assignment 59
core clinet-service protocol 127
CORE Peer-to-Peer Protocol 128
CORE Subsystem 124
core subsystem limitations 125
Cryptography API 102

D

Deniability 8
Design Goals 4
DHT 155
Distributed Hash Table 155

E

ECRS 170
Egos 9
Encoding for Censorship-Resistant Sharing 170
EphemeralKeyMessage creation 128

F

FS 169
FS Subsystem 169

G

GNS 160
GNS Namecache 165
GNU Name System 160
GNUnet GTK 10
gnunet-ext 75
GTK 10
GTK user interface 10

H

HOSTLIST client 139
HOSTLIST daemon 138
HOSTLIST learning 139
HOSTLIST server 138
HOSTLIST Subsystem 136
How file-sharing achieves Anonymity 7

I

ICP 97
IDENTITY Subsystem 140
Interprocess communication API 97

K

Key Concepts 5
ksk-uri 26

L

libgnunetblock API 155
libgnunetcadet 131
libgnunetcore 126
libgnunetdht 157
libgnunetidentity 140
libgnunetnamestore 142
libgnunetnse 134
libgnunetpeerstore 147
libgnunetset 149
libgnunetstatistics 153
libgnunetutil 91
libgnurl 77
license, GNU Free Documentation License 175
license, GNU General Public License 183
loc-uri 26
Log files 95
log levels 92
Logging 92

M

Message Queue API 103

N

NAMESTORE Subsystem 142
NAT library 113
NSE Peer-to-Peer Protocol.................... 135
nse principle.................................. 132
NSE principle.................................. 132
nse security................................... 132
NSE security 132
NSE Subsystem............................... 132

P

Peer Identities 8
PEERINFO Subsystem 144
PEERSTORE Subsystem..................... 147
Philosophy...................................... 3
PlanetLab testbed 89
Practicality 5

R

REGEX subsystem 171
REST subsystem 173
REVOCATION Subsystem 167

S

Security and Privacy 4
Service API.................................... 105
SET Subsystem............................... 148
sks-uri 26
SMTP plugin................................. 115
STATISTICS Subsystem...................... 152

T

TESTBED Caveats 90
TESTBED Subsystem 83
TESTING API 80
TESTING library.............................. 80
TRANSPORT Subsystem..................... 112

V

Versatility 4

W

when is a peer connected...................... 125
writing testcases 75

Z

Zones in the GNU Name System (GNS Zones)... 9

Programming Index

(

(CORS) 174

G

GNUNET_SERVICE_Options 105