Peer-to-Peer Systems and Security

Introduction to GNUnet 0.9.x for Developers

Christian Grothoff

Lehrstuhl für Netzarchitekturen und Netzdienste
Institut für Informatik
Technische Universität München

May 5, 2011

Agenda

- GNUnet 0.9.x Release Status
- GNUnet 0.9.x Features
- GNUnet 0.9.x System Overview
- GNUnet 0.9.x APIs

GNUnet 0.9.x Release Status

- GNUnet 0.9.0pre3 is an alpha release
- GNUnet 0.9.0pre3 works on GNU/Linux, OS X, likely Solaris
- GNUnet 0.9.0pre3 has known bugs (see TODO, Mantis)
- GNUnet 0.9.0pre3 lacks documentation
- GNUnet 0.9.0pre3 has a somewhat steep learning curve

GNUnet 0.9.x Release Status

- GNUnet 0.9.0pre3 is an alpha release
- GNUnet 0.9.0pre3 works on GNU/Linux, OS X, likely Solaris
- GNUnet 0.9.0pre3 has known bugs (see TODO, Mantis)
- GNUnet 0.9.0pre3 lacks documentation
- GNUnet 0.9.0pre3 has a somewhat steep learning curve
- APIs may still change slightly for 0.9.0
- Protocols may still change slightly for 0.9.0

Agenda

- GNUnet 0.9.x Release Status
- **GNUnet 0.9.x Features**
- GNUnet 0.9.x System Overview
- GNUnet 0.9.x APIs

GNUnet 0.9.x Features

- OS abstraction layer
- Bandwidth management
- Transport abstraction (TCP, UDP, ...)
- Link encryption
- Peer discovery (hostlist, P2P gossip)
- Topology management

GNUnet 0.9.x Features

- Logging, configuration management, command-line parsing
- Cryptographic primitives
- Event loop, client-server IPC messaging infrastructure
- Binary I/O, asynchronous DNS resolution,
- Datastructures (Heap, HashMap, Bloomfilter)

GNUnet 0.9.x Features

- Datastore (for file-sharing)
- Datacache (for DHT)
- Statistics
- Testbed management (loopback & distributed testing)
- Automatic Restart Management

GNUnet 0.9.x DHT Features

- Randomized DHT based on Kademlia
- Command-line interface (GET/PUT)
- Client-library (C API)
- Should work pretty well, but unreliable as any P2P operation (!)

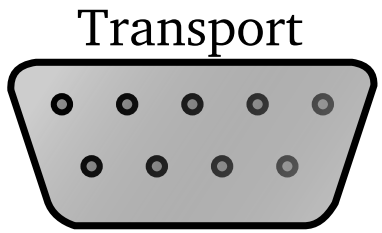
Agenda

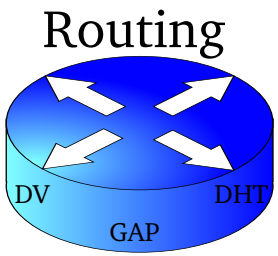
- GNUnet 0.9.x Release Status
- GNUnet 0.9.x Features
- **GNUnet 0.9.x System Overview**
- GNUnet 0.9.x APIs

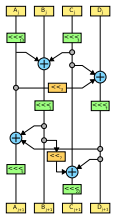
GNUnet System Overview: Help!

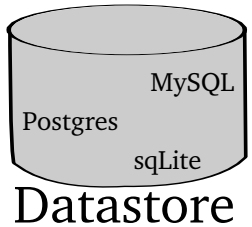
- https://gnunet.org/
  - How to build & run GNUnet
  - End-user and developer manuals, FAQ
  - Bug database
  - Doxygen source code documentation
  - Regression tests results
  - Code coverage analysis
  - Static analysis
- irc.freenode.net#gnunet

GNUnet System Overview









GNUnet System Overview: 0.9.x Philosophy

- `gnunetutil` library provides shared functions for services, daemons and user interfaces
- No (more) threads (no deadlocks, no races, no fun)
- Services are processes accessed via C API
- Daemons are processes without an API
- Service API use IPC (TCP/IP or UNIX Domain Sockets) to communicate with the respective service process
- Service processes are managed by `gnunet-service-arm`
- `gnunet-service-arm` is controlled with `gnunet-arm`

GNUnet System Overview: Dependencies

- libgcrypt
- libgmp
- libmicrohttpd $\geq$ 0.9.9!
- libextractor $\geq$ 0.6.x!!
- sqlite
- mysql
- postgres

Agenda

- GNUnet 0.9.x Release Status
- GNUnet 0.9.x Features
- GNUnet 0.9.x System Overview
- **GNUnet 0.9.x APIs**

APIs: Function Pointers

- C has first-class, higher-order functions
- GNUnet uses those

APIs: Inner Functions

- C has first-class, higher-order functions
- GNUnet uses those
- GNU GCC has inner functions
- GNUnet does **not** use inner functions

APIs: Function Pointers and Closures

- C has first-class, higher-order functions
- GNUnet uses those
- GNU GCC has inner functions
- GNUnet does **not** use inner functions
- GNUnet passes a `void *` closure (`cls`) as an explicit first argument to all higher-order functions

APIs: gnunet_util_lib.h

- Header includes many other headers
- Should be included after `platform.h`
- Provides OS independence / portability layer
- Provides higher-level IPC API (message-based)
- Provides some data structures (Bloom filter, hash map, heap, doubly-linked list)
- Provides configuration parsing
- Provides cryptographic primitives (AES-256, SHA-512, RSA, (P)RNG)
- Use: `GNUNET_malloc`, `GNUNET_free`, `GNUNET_strdup`, `GNUNET_snprintf`, `GNUNET_asprintf`, `GNUNET_log`, `GNUNET_assert`

APIs: GNUNET_assert and GNUNET_break

- GNUNET_assert aborts execution if the condition is false (0); use when internal invariants are seriously broken and continued execution is unsafe
- GNUNET_break logs an error message if the condition is false and then continues execution; use if you are certain that the error can be managed and if this has to be a programming error with the local peer
- GNUNET_break_op behaves just like GNUNET_break except that the error message blames it on other peers; use when checking that other peers are well-behaved
- GNUNET_log should be used where a specific message to the user is appropriate (not for logic bugs!); GNUNET_log_strerror and GNUNET_log_strerror_file should be used if the error message concerns a system call and errno

APIs: gnunet_scheduler_lib.h

- Part of libgnunetutil
- Main event loop
- Each *task* is supposed to never block (disk IO is considered OK)
- SCHEDULER can be used to schedule tasks based on IO being ready, timeouts or completion of other tasks
- Each task has a unique 64-bit GNUNET_SCHEDULER_TaskIdentifier that can be used to *cancel* it
- The event loop is typically started using the higher-level PROGRAM or SERVICE abstractions

APIs: SHUTDOWN

The scheduler provides a somewhat tricky way to install a function that will be run on shutdown:

```
static void
my_shutdown (void *cls,
             const struct GNUNET_SCHEDULER_TaskContext *tc)
{
  GNUNET_assert (0 != (tc->reason & GNUNET_SCHEDULER_REASON_SHUTDOWN));
  GNUNET_CORE_disconnect (core);
}
static void
my_run (...)
{
  GNUNET_SCHEDULER_add_delayed (GNUNET_TIME_UNIT_FOREVER_REL,
                                &my_shutdown, NULL);
}
```

APIs: gnunet_server_lib.h

- Part of libgnunetutil
- Used to receive requests from service APIs
- For example, GET/PUT requests from DHT API
- Main uses: register handler, transmit response to client

APIs: gnunet_protocols.h

- Used to define message types
- Each message in GNUnet begins with 4 bytes: type & size
- 64k message types, up to 64k of data per message
- You will need to define some message type(s) for your services

The STATISTICS service provides an easy way to track performance information:

```
struct GNUNET_STATISTICS_Handle *
GNUNET_STATISTICS_create (const char *subsystem,
                          const struct GNUNET_CONFIGURATION_Handle *cfg);
void
GNUNET_STATISTICS_set (struct GNUNET_STATISTICS_Handle *handle,
                       const char *name,
                       uint64_t value, int make_persistent);
void
GNUNET_STATISTICS_update (struct GNUNET_STATISTICS_Handle *handle,
                          const char *name,
                          int64_t delta, int make_persistent);
```

With this, you can then use `gnunet-statistics` to inspect the current value of the respective statistic.

The TESTING library provides an easy way to setup testbeds:

```
struct GNUNET_TESTING_PeerGroup *
GNUNET_TESTING_testbed_start (const struct GNUNET_CONFIGURATION_Handle *cfg,
                              unsigned int total,
                              struct GNUNET_TIME_Relative timeout,
                              GNUNET_TESTING_NotifyConnection connect_cb,
                              GNUNET_TESTING_NotifyCompletion peergroup_cb,
                              void *peergroup_cls,
                              const struct GNUNET_TESTING_Host *hostnames);
```