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

April 30, 2010

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 0.9.0pre0 is an alpha release
- GNUnet 0.9.0pre0 works on GNU/Linux, OS X, likely Solaris
- GNUnet 0.9.0pre0 has known bugs (see TODO, Mantis)
- GNUnet 0.9.0pre0 lacks documentation
- GNUnet 0.9.0pre0 has a somewhat steep learning curve

GNUnet 0.9.x Release Status

- GNUnet 0.9.0pre0 is an alpha release
- GNUnet 0.9.0pre0 works on GNU/Linux, OS X, likely Solaris
- GNUnet 0.9.0pre0 has known bugs (see TODO, Mantis)
- GNUnet 0.9.0pre0 lacks documentation
- GNUnet 0.9.0pre0 has a somewhat steep learning curve
- APIs will still change for 0.9.0
- Protocol will still change 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

- Command-line interface (GET/PUT)
- Client-library (C API)
- Skeleton service
- Integration with datacache

⇒ GET/PUT on loopback already works, just add routing!

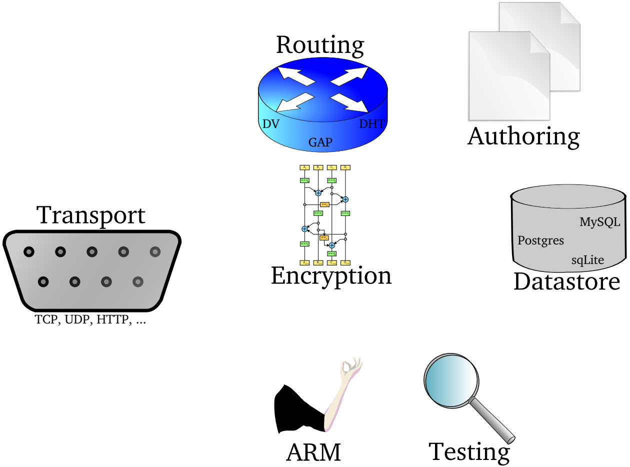
Agenda

- 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://ng.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 $\geq$ 0.4.6!
- libextractor $\geq$ 0.6.x!!
- sqlite
- mysql (soon)
- postgres (soon)

GNUnet System Overview: Getting started

```
configure --prefix=$HOME
make
make install
export GNUNET_PREFIX=$HOME
export PATH=$HOME/bin
make check
mkdir .gnunet/
touch .gnunet/gnunet.conf
gnunet-arm -s
```

GNUnet System Overview: Baby Steps

```
gnunet-arm -i datacache
gnunet-arm -i dht
gnunet-dht-put KEY VALUE
gnunet-dht-get KEY
gnunet-statistics
gnunet-statistics -s dht
```

GNUnet System Overview: Debugging

```
gnunet-arm -k dht
CFG=~/.gnunet/gnunet.conf
echo -e "[dht]\n" >> $CFG
echo -e "PREFIX=xterm -e gdb --args\n" >> $CFG
gnunet-arm -i dht
gnunet-arm -k dht
gdb --args gnunet-service-dht -L DEBUG
valgrind gnunet-service-dht -L DEBUG
```

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: Starting a service

```
typedef void (*GNUNET_SERVICE_Main) (void *cls,
                                     struct GNUNET_SCHEDULER_Handle * sched,
                                     struct GNUNET_SERVER_Handle * server,
                                     const struct GNUNET_CONFIGURATION_Handle *cfg);
int GNUNET_SERVICE_run (int argc,
                        char *const *argv,
                        const char *serviceName,
                        enum GNUNET_SERVICE_Options opt,
                        GNUNET_SERVICE_Main task,
                        void *task_cls);
```

APIs: Example invocation for GNUNET_SERVICE_run

```c
static void my_main (void *cls,
                     struct GNUNET_SCHEDULER_Handle * sched,
                     struct GNUNET_SERVER_Handle * server,
                     const struct GNUNET_CONFIGURATION_Handle *cfg)
{
  /* do work */
}

int main (int argc, char *const*argv)
{
  if (GNUNET_OK !=
      GNUNET_SERVICE_run (argc, argv, "my",
                          GNUNET_SERVICE_OPTION_NONE,
                          &my_main, NULL);
    return 1;
  return 0;
}
```

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: 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 the DHT

APIs: gnunet_service_datacache.h

- Simple API for (temporarily) storing blocks
- Datacache has finite size and all is lost on shutdown!
- Blocks have a type (defined in gnunet_block_lib.h)

One of the first things any service that extends the P2P protocol typically does is connect to the CORE:

```
struct GNUNET_CORE_Handle *
GNUNET_CORE_connect (struct GNUNET_SCHEDULER_Handle *sched,
                     const struct GNUNET_CONFIGURATION_Handle *cfg,
                     struct GNUNET_TIME_Relative timeout,
                     void *cls,
                     GNUNET_CORE_StartupCallback init,
                     GNUNET_CORE_ConnectEventHandler connects,
                     GNUNET_CORE_DisconnectEventHandler disconnects,
                     GNUNET_CORE_MessageCallback inbound_notify,
                     int inbound_hdr_only,
                     GNUNET_CORE_MessageCallback outbound_notify,
                     int outbound_hdr_only,
                     const struct GNUNET_CORE_MessageHandler *handlers);
```

In response to events (connect, disconnect, inbound messages, timing, etc.) services can then use this API to transmit messages:

```
typedef size_t
(*GNUNET_CONNECTION_TransmitReadyNotify) (void *cls,
                                          size_t size,
                                          void *buf);

struct GNUNET_CORE_TransmitHandle *
GNUNET_CORE_notify_transmit_ready (struct GNUNET_CORE_Handle *handle,
                                   uint32_t priority,
                                   struct GNUNET_TIME_Relative maxdelay,
                                   const struct GNUNET_PeerIdentity *target,
                                   size_t notify_size,
                                   GNUNET_CONNECTION_TransmitReadyNotify notify,
                                   void *notify_cls);
```

APIs: gnunet_service_peerinfo.h

The PEERINFO API can be used to obtain information about all known peers (and to be notified about changes to that set):

```
typedef void
(*GNUNET_PEERINFO_Processor) (void *cls,
                              const struct GNUNET_PeerIdentity *peer,
                              const struct GNUNET_HELLO_Message *hello,
                              uint32_t trust);
struct GNUNET_PEERINFO_NotifyContext *
GNUNET_PEERINFO_notify (const struct GNUNET_CONFIGURATION_Handle *cfg,
                        struct GNUNET_SCHEDULER_Handle *sched,
                        GNUNET_PEERINFO_Processor callback,
                        void *callback_cls);
```

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);
  GNUNET_PEERINFO_notify_cancel (nc);
}
static void
my_run (struct GNUNET_SCHEDULER_Handle *sched, ...)
{
  GNUNET_SCHEDULER_add_delayed (sched,
                                GNUNET_TIME_UNIT_FOREVER_REL,
                                &my_shutdown, NULL);
}
```

APIs: gnunet_service_statistics.h

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

```
struct GNUNET_STATISTICS_Handle *
GNUNET_STATISTICS_create (struct GNUNET_SCHEDULER_Handle *sched,
                          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_Testbed *
GNUNET_TESTING_testbed_start (struct GNUNET_SCHEDULER_Handle *sched,
                              const struct GNUNET_CONFIGURATION_Handle *cfg,
                              unsigned int count,
                              enum GNUNET_TESTING_Topology topology,
                              GNUNET_TESTING_NotifyDaemonRunning cb,
                              void *cb_cls,
                              const char *hostname,
                              ...);
void
GNUNET_TESTING_testbed_churn (struct GNUNET_TESTING_Testbed *tb,
                              unsigned int voff,
                              unsigned int von,
                              GNUNET_TESTING_NotifyCompletion cb,
                              void *cb_cls);
```

The DHT Project

- DHTs are a key building block for P2P networks
- We've provided most of what a DHT needs in GNUnet for you:
  - Local storage (DATACACHE)
  - (Encrypted, authenticated) message exchange (CORE)
  - Initial peer discovery (HOSTLIST/PEERINFO)
  - Command-line tools (`gnunet-dht-get`, `gnunet-dht-put`)
  - Peer identifiers (`struct GNUNET_PeerIdentity` in a key space (`GNUNET_HashCode`)
  - Distance metrics (`GNUNET_CRYPTO_hash_cmp` and `GNUNET_CRYPTO_hash_xorcmp`)
- You need to implement:
  - Routing table data structure, population, handling of churn
  - Routing decision procedure
  - Documentation
  - Correctness tests
  - Performance evaluation

The DHT Project

- Start by checking out a current revision of the project:

```
svn checkout https://ng.gnunet.org/svn/libmicrohttpd/
svn checkout -r 11111 https://ng.gnunet.org/svn/gnunet/
```

- We will tell you if and when it is safe (and a good idea) to update to a more recent version (bugfixes!)
- After installing dependencies (see webpage), run

```
. bootstrap
export GNUNET_PREFIX=SOMEPATH
export PATH=$PATH:$GNUNET_PREFIX/bin
export LD_LIBRARY_PATH=$GNUNET_PREFIX/lib
./configure --prefix=$GNUNET_PREFIX
make
make install
make check
```