GNUnet - Transports and Transport Selection

Matthias Wachs

Technische Universität München
Department of Computer Science
Network Architectures and Services

July, 25 2010

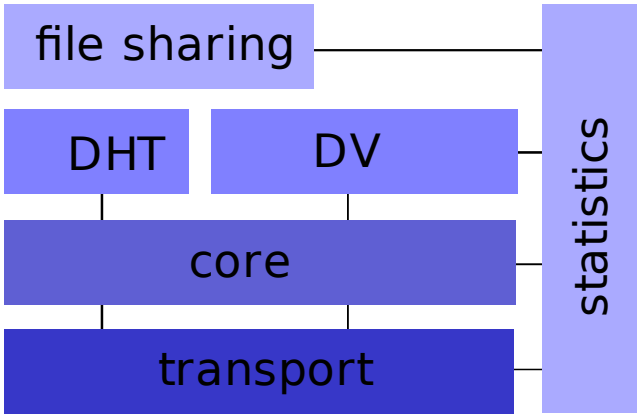
GNU Hacker Meeting - The Hague

Overview

- Services in GNUnet
- Transport Service
- Transport Service Architecture
- Transport Service Plugins
  - TCP
  - UDP
  - HTTP
  - Distance Vector
  - Wifi
  - NAT traversal in GNUnet
- DIY: Create a Transport Plugin
- Transport Selection
  - Availability
  - Cost
  - Quality
- Summary and Conclusion

Services in GNUnet

- Functionality in GNUnet is split in parts
- Every part is realised as a separate **GNUnet service**
- Each GNUnet service is a process with own address space
- Services are communicating via sockets:
  TCP/IP or UNIX domain sockets
- Some of the services are:



Benefits of separated functionality

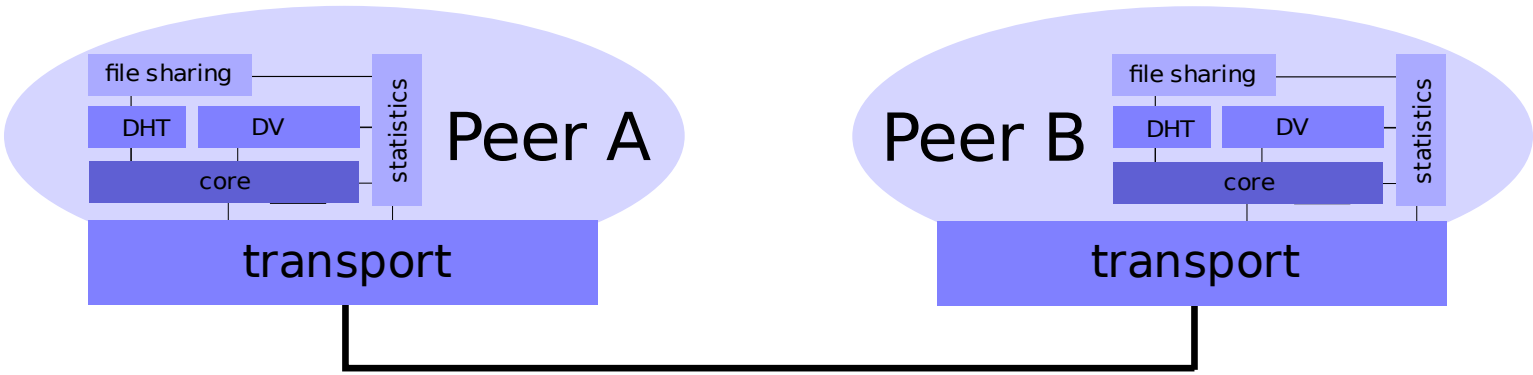
Separating program functionality leads to:

- improved project structure
- improved usage of multiprocessing systems
- simplified concurrency
  $\Rightarrow$ no threads, no thread synchronisation code
- less code inter-dependencies
- independent parts of the code are located in separate address space
  $\Rightarrow$ preventing hard to debug memory corruption
- better analysis of runtime behaviour and crash detection

Transport Service Architecture

Transport services manages connectivity to other peers:

- accepts messages from services to send to other peers
- interacts with other services using well-defined API
- abstracts from underlying transport mechanisms used
- provides traffic management
- provides extensible architecture
- supports IPv4 and IPv6
  peer can be connected over IPv4 and IPv6 at the same time

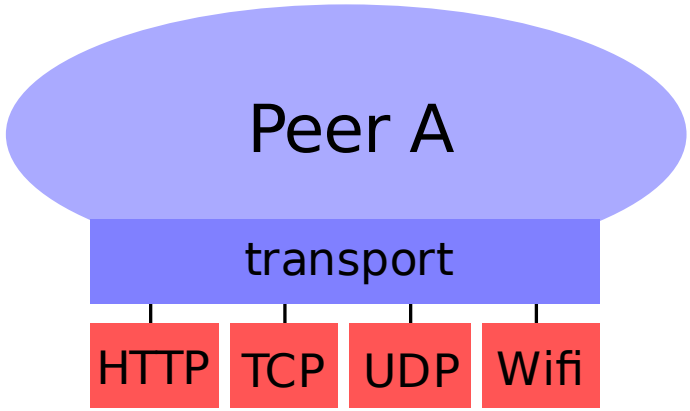


Transport Service Architecture

GNUnet transport service has a flexible and extensible architecture:

- extensible with plugins
- abstracts from used transport mechanism
- separated by API from other services
- well-defined API to load transport plugins dynamically
- different plugins can work on different ISO/OSI layers

- application layer: HTTP
- transport layer: TCP, UDP
- network layer: IPv4, IPv6
- physical layer: Wifi, Ethernet



Transport Plugins

- transport plugins are dynamically loadable C libraries
- loaded by transport service during startup
- can be loaded during operation

Benefits

- simple to add new transport mechanisms
- implementation of transport mechanism is hidden
  $\Rightarrow$ used mechanism is exchangeable
- functionality of multiple transports separated from service
  $\Rightarrow$ no interference between different plugins

Transport Plugins

Currently existing plugins:

- HTTP
  experimental
- TCP with NAT traversal support
  working
- UDP with NAT traversal support
  experimental
- Distance Vector
  working
- Wifi
  work in progress

TCP Transport Plugin

The TCP transport plugin provides

- connection oriented
- reliable
- bi-directional

transport mechanism.

TCP itself provides:

- built-in congestion and flow control
- error control

- Two peers can send and receive over one TCP connection

UDP Transport Plugin

The UDP transport plugin provides

- connection-less
- unreliable
- uni-directional

transport mechanism

Every peer sends packets when required:

- UDP packets can be dropped due to congestion, overload or transmission errors.
- GNUnet has to prevent network overload $\Rightarrow$ traffic engineering with quotas

HTTP Transport Plugin

HTTP is an application layer protocol using TCP:

- all benefits of TCP (reliability, congestion and flow control)

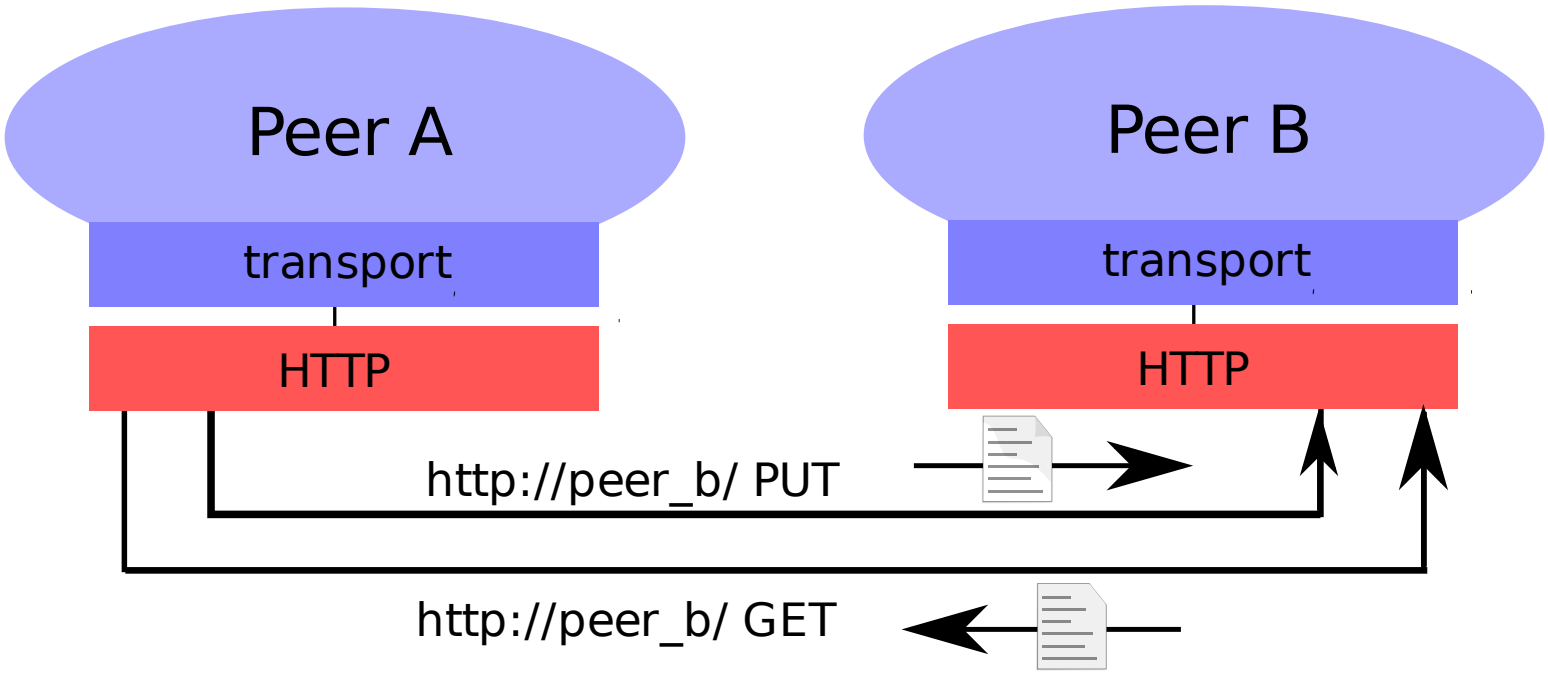
The HTTP transport plugin:

- uses the HTTP protocol to transfer data between peers
- based on libcurl and GNU libmicrohttpd
- plugins complies with HTTP protocol
- combines PUT and GET requests to transfer data between peers
- little overhead after connection is established
- simultaneous connections to same peer at a time: IPv6 and IPv4
- connections stay established:
  chunked encoding and reusing connections
  $\Rightarrow$not a single connection for every message

HTTP Transport Plugin

When Peer A connects to Peer B, it establishes:

- a PUT request to send data to Peer A
- a GET request to receive data from Peer B



Distance Vector

- Implements onion routing
- Nate will tell you more!

Wifi transport plugin

Connect peers directly using wifi

- build local ad-hoc networks
- no need to configure network
- different layer than other plugins:
  works on physical layer

new challenges in traffic engineering:

- measure and control network load
- control signal strength and energy consumption

Work in progress, deadline: end of 2010

NAT Traversal

GNUnet provides NAT traversal techniques:

- Network Address Translation is often described is security feature
- but: NAT prevents incoming connections

Some transports can be configured to use NAT traversal techniques:

- ICMP messages
  $\Rightarrow$ requires raw sockets $\Rightarrow$ suid
- functionality separated in client and server parts
- client punches hole in NAT sending "fake" ICMP messages
- server keeps hole open by sending periodically ICMP messages

NAT Traversal can be used with TCP, UDP and HTTP

Detailed information:
Autonomous NAT Traversal https://gnunet.org/pwnat

DIY: Create a Transport Plugin

Creating your own transport plugin is quite simple (in theory):

Steps to do:

- Step 1: Copy plugin template and implement template functions
- Step 2: Call transport functions
- Step 3: Implement tests and run given testcases
- Step 4: Configure transport to load plugin

DIY: Create a Transport Plugin

Step 1: Implement the template functions:

- Start up:
  `libgnunet_plugin_transport_<your_plugin>_init`
- Shutdown:
  `libgnunet_plugin_transport_<your_plugin>_done`
- Send message:
  `<your_plugin>_plugin_send`
- Disconnect from peer:
  `<your_plugin>_plugin_disconnect`
- Print address in a fancy way:
  `<your_plugin>_plugin_address_pretty_printer`
- Check if address is valid:
  `<your_plugin>_plugin_address_suggested`
- Print address as a string:
  `<your_plugin>_plugin_address_to_string`

DIY: Create a Transport Plugin

Step 2: Call transport service functions from time to time:

- Forward received message:
  `plugin_env_receive`
- Tell transport which addresses you use:
  `notify_address`
- Tell transport a session to a peer ends:
  `plugin_env_session_end`

Add all the functionality your plugin needs!

DIY: Create a Transport Plugin

Step 3: Implement tests and run given testcases

- implement your own testcases to check functionality

GNUnet has predefined testcases:

- testcases to check plugin functionality and reliability
- add your plugin to be loaded by testcase
  minor changes to existing testcases are needed
- add your plugin to Makefile.am to build testcases:
  `TESTS = test_transport_api_<your_plugin>`
  `check_PROGRAMS = test_transport_api_<your_plugin>`
- run testcase suite to check your plugin

DIY: Create a Transport Plugin

Step 4: Configure transport service to load plugin

- Configure your plugin:

```
[transport-<your-plugin>]
PORT = 12389
USE_IPv6 = NO
USE_IPv4 = YES
```

- Load plugin:

```
PLUGINS = http udp <your-plugin>
```

Transport Selection

Providing a lot of different transport mechanisms is fancy

But:

- Which transport mechanism should GNUnet use?
- When should it switch between mechanisms?
- How good is a transport mechanism at all?
- How can we measure a transport mechanism?

A mechanism for transport evaluation and selection is needed!

$\Rightarrow$ My research topic!

Quality of a Service

A transport mechanism can be weighted by:

- Availability
- Cost
- Quality

Availability of a Transport

Availability as an aspect to select transports:

- Is the peer available?
- Can we connect to the peer?
- Do we speak the same protocols?

$\Rightarrow$ Check periodically for availability and determinate currently available transport mechanisms

Cost of a Transport

Cost should be considered when selecting a transport

- Cost in financial units:
  volume charged vs. flatrate
  LAN vs. free of charge wifi vs. charged 3G
- Cost of bandwidth overhead:
  protocol overhead of different transports
- Computational cost:
  HTTP vs. HTTPS
- Energy consumption in wifi and mobile usage:
  More energy is consumed if you have to send with full power
- ...

$\Rightarrow$ Prefer transport mechanisms with low costs

Quality of a Transport

A transport mechanism can be measured by the quality of the service

On physical layer:

- Signal strength
- Signal noise ratio
- ...

On network layer:

- Throughput
- Delay (round trip time)
- Loss rate
- Current network load
- ...

$\Rightarrow$ Prefer transport mechanisms with better quality

Transport Selection

Aspects to consider during transport selection and switching:

- Availability
- Cost
- Quality

More decisions to make:

- How often to check transport availability?
  $\Rightarrow$ More often when using expensive or bad quality transports
- Is it worth changing transports?
  $\Rightarrow$ New connections have to be opened, overhead occurs

Transport Selection

Transport selection today:

- Measurements to evaluate transport quality
- RTT
- Select transport with lowest delay

Future plans:

- Examine transports for aspects to evaluate transports by
- Develop a metric for transports selection

Summary and Conclusion

GNUnet is an extensible framework with a flexible transport service:

- provides traffic management
- transport service hides transport mechanisms
- easy to implement transport plugins
- transport plugins can work on every ISO/OSI layer

Questions?

wachs@net.in.tum.de