Peer-to-Peer Networking

Project Work: Gnutella protocol v0.4 (project draft v0.9)

1 Project Description

The target of this project work is to develop a working peer-to-peer client based on Gnutella protocol specification version 0.4 (document revision 1.2). You only need to implement a subset of the full protocol in a way, that your P2P client, from now on called *servent*, is able to provide the minimum functionality involved in file sharing networks. This requires the following:

- *bootstrapping* (connecting the network),
- *routing* messages,
- *searching* for files, and
- *transferring* the files

You can develop your servent in any common programming language, that can be run on a Linux platform. If you desire to use some exotic language, please contact staff before starting to ensure a proper testing environment. Giving the freedom to choose the language is intended to let you use the most suitable one for you, and demonstrate the fact that a protocol specification should not be bound to any specific implementation technique. You are not required to provide any fancy user interface, minimum requirement is a terminal based UI.

Why Gnutella protocol version 0.4? Gnutella is one of the simpler P2P protocols, therefore making it a bit more easier to implement it as an exercise. However, there are several reasons not to go with the *de facto* v0.6. The first one is that v0.6 is still a *draft* and may be subject to changes. Furthermore, it introduces several extensions of messages (e.g., vendor-specific fields provided by different implementations) and more advanced and complex network (hierarchical overlay topology built upon ultra-peers). This would be out of our project implementation scope. Nevertheless, you are invited to study and try the newer v0.6 specification, but our project will only concern with the standard v0.4 specification.

Learning aim for this exercise is to familiarize yourself with the practical side of P2P development by implementing a file sharing network. This requires you to carefully study and understand the protocol specifications. Correct implementation is critical for interoperability among the servents. Therefore your implementation must be compatible with the test environment provided in computer class.

All related documents, including this project specification, are provided on the project web page.

2 Gnutella Overview

As defined in the protocol:

> Gnutella is a protocol for distributed search. Although the Gnutella protocol supports a traditional client/centralized server search paradigm, Gnutella's distinction is its peer-to-peer, decentralized model. In this model, every client is a server, and vice versa. These so-called Gnutella servents perform tasks normally associated with both clients and servers. They provide client-side interfaces through which users can issue queries and view search results, while at the same time they also accept queries from other servents, check for matches against their local data set, and respond with applicable results. Due to its distributed nature, a network of servents that implements the Gnutella protocol is highly fault-tolerant, as operation of the network will not be interrupted if a subset of servents goes offline.

3 Bootstrapping

The minimum requirement for servent to join the network is via command `*open localhost port*' where *localhost* is the address of a Gnutella servent listening in port number *port.* A commonly used method for bootstrapping is the Gnutella Web Caching System (GWCS). In GWCS, a well known server is queried for a list of active peers to connect. GWCS protocol specification details for version 1 are provided on the project work web pages, follow them. For minimum functionality, you only need to implement the hostfile request/response. Following a HTTP redirect can be omitted here. **Note:** CWCS support is not mandatory, but you will be credited extra points if your servent can utilise the cache server provided in the computer class.

4 Protocol Overview

The protocol specification document is provided on the project web page. It defines the messages and required functionality. In our exercise, you need to implement the following message types:

- Ping
- Pong
- Query
- Query Hit

You are not required to implement Push message for bypassing firewalls. You can also omit specifications for “firewalled” (behind a proxy or NAT) servent and appendix 1 (protocol extensions). It's still recommended to think in terms of TCP/IP how the file transfer can bypass a firewall, or what happens when both servents are being firewalled.

Once a servent has connected successfully to the network, it communicates with other servents by sending and receiving Gnutella protocol messages. You should note that:

- One IP packet may contain several Gnutella messages, and one Gnutella message may be split up on multiple IP-packets. This means one can never assume a Gnutella message ends when the chunk of data read from the socket ends.
- All fields in the following structures are in *little-endian byte order* unless otherwise specified.
- All IP addresses in the following structures are in IPv4 format. For example, the following IPv4 byte array represents the dotted address 192.0.2.133:

byte 0	byte 1	byte 2	byte 3
0xC0	0x00	0x02	0x85

The specification should be quite clear. However, you need to pay attention, that messages are constructed properly. As the documentation states, data fields are measured in *bytes* and you must obey the given formats. Another issues is *byte endianness*, in other words, byte ordering. *Big-endian* and *little-endian* define how multi-byte data types are stored in the computer’s memory. The protocol defines which format is used when data is transferred over the network.

5 Searching and File Downloads

You are required to implement the Query and Query Hit messages. Query complexity (case sensitivity, partial matches, wild card, etc.) is you to decide, but the general principle (and minimum requirement) is that an exact match should produce a hit. Provide the search functionality with command syntax `*search string*', where *string* is the search string. Search results should present the necessary information for retrieving the file. For terminal based UI Shared files should be located in the same directory as the executable binary. **Note:** implementing file transfer is not mandatory, but recommended. Again, you will be credited for some extra points for working file transfer (both ways). Syntax of the command for downloading a file can be `*get filename servent_identifier*', but you can also think something more convenient for typing, like indexing the search results `*get result_index*'.

6 Default Parameters

The protocol does not specify certain parameters, for example, the rate in which a servent sends Ping descriptors. Therefore, you should use the following parameters for this project:

- Maximum number of connected neighbours: 7. Beyond that, use reply GNUTELLA BUSY\n\n
- Maximum Time To Live (TTL) value: 7
- Maximum size of descriptor cache: 1024
- Rate for sending Ping descriptor: 30 seconds

7 Other implementation issues

Plagiarism is forbidden. While it's recommended to use web and other resources to help you on your assignment, you cannot use code written by someone else. When using resources, mark them in the start of your source code with a short explanation where and how it's applied. Failing to provide used resources may seem a bit suspicious, so it's always better to mark them. If you are in any doubt, please ask the teaching personnel.

Tips and tricks. You should start by creating a servent stub which can listen incoming TCP connection and connect to another servent. Probably the easiest way to start building the communication is creating a successful handshake and Ping/Pong messaging. Troubleshooting with *netcat* is always useful when communicating with string messages. You should also note, that GWCS and file transfer *do use* HTTP protocol. Either utilise some ready HTTP library in your programming language or write proper HTTP headers yourself.

Computing environment and testing. Your code must compile and run in the laboratory (room 6218), which computers are currently Debian GNU/Linux 4.0 (Etch) or similar based. There will be present a GWCS and possibly a servent holding files for testing and evaluation purposes. You will need to give a demonstration, in where the servent is tested and implementation is explained. A collective testing will also be held at the same time to see how the servents interoperate. Dates will be announced later.

8 Grading

Your work is graded 0-5. See *Table 1* below for grading details. You should concentrate on the first three issues; your implementation *must* work according to specifications in order to pass the assignment. Getting a grade of 3 should really not be so difficult. Several additional features are listed for obtaining higher grades.

Requirement	Explanation	Grade
Minimum specifications	**Must meet the minimum requirements specified above**	**1**
Good programming practices	**Clean code, follows general programming guidelines (e.g., no global variables in C code)**	**2**
Commenting code	**Readability, maintenance and intelligibility of the code**	**3**
GWCS bootstrapping	Implementing hostfile request and update request (inform server about your node)	+1
File transfer	Able to retrieve and upload files based on search results	+1
Push message	Able retrieve and upload files when behind a firewall	+1
Protocol extension	Create a *justified, useful* protocol extension, which can be omitted if not supported by the other end	+1

Table 1: Grading evaluation chart