Implementation of a Transaction-based Peer-to-Peer Protocol

Sofia Nygård[1] and Nils Kullberg[2]

[1]Department of Information Technologies, Åbo Akademi University, Turku Finland
sonygard@abo.fi
[2]Laboratory of Polymer Technology, Åbo Akademi University, Turku Finland
nils.kullberg@abo.fi

Abstract

The paper describes a lightweight implementation of a peer-to-peer protocol. The implementation is intended as a testbed for ideas concerning the utilization of the peer-to-peer paradigm. The protocol takes the Gnutella protocol as a starting point, although the present implementation differs from the Gnutella protocol regarding intent as well as routing solutions.

The protocol is implemented in Python since various data types and modules offered by Python make it possible to limit the code to a manageable size. The GUI is implemented with Tkinter.

The design has two substantial features. Firstly, the protocol is transaction-based. As a consequence, the pong messages are no longer required since every successfully transferred message, independent of type, is a sign that a node is active. An adaptive ping algorithm further reduces the load on bandwidth. Secondly, there is only one entry point to the router. The way in which the router communicates with the GUI is symmetrical to the way in which it communicates with the network. This means that the connections between the router and the GUI are characterized by socket communication and protocol messages. This results in a fairly independent user interface, which creates a basis for further utilization of peer-to-peer systems.

1 Introduction

A basic notion underlying all peer-to-peer (p2p) systems is that ordinary PC-users have something valuable to share [20]. This may be network capacity, CPU-cycles or storing capacity. The notion of nodes acting as peers is not new, even if the actual term peer-to-peer is of a fairly recent date. The Arpanet was created in 1969, and the nodes in the Arpanet were peers. These could locate other nodes, share files, and take part in distributed computing [28, 30].

The term p2p came into being as a way to describe new applications for file sharing [20, 22, 39] and distributed computing [4]; hence, the definitions often deal with features related to these networks, such as topology [27], connectivity [39] and performance [18]. Nodes in a p2p network have a limited awareness of the structure as well as of the resources of the network. This awareness varies according to whether the network is based on a structured or on an unstructured design.

The conceptual ancestor of the present design is to be found in the Gnutella protocol [13, 14]. This means, that the design adopts broadcasting as a means of communication. However, the design differs from the Gnutella protocol in several respects: the design is transaction-based, the pong message is eliminated from the protocol, the ping message carries information load and; furthermore, the ping message is sent according to an adaptive algorithm. The intent of the design also differs from the intentions behind the Gnutella protocol. Various versions of Gnutella are used for locating (often copyrighted) material. The present design is intended as a testbed for ideas concerning the development and utilization of p2p protocols. One conclusion drawn from the implementation is that the protocol would benefit from being rewritten into a pure ASCII protocol. Furthermore, the fairly independent user interface combined with the transaction-based design suggest the utilization of a web-

based interface. One possible consequence is to employ HTTP as the means of communication between p2p-nodes.

In the next section, we present the peer-to-peer paradigm, its definitions and main research areas. We then describe the Gnutella protocol messages. These constitute the comparison material for the algorithms and solutions presented in this paper. Next, we describe a transaction-based p2p protocol design. The design entails opportunities for further developing the protocol. One recommendation is to utilize HTTP as a means of exchanging p2p messages. Finally, we present an illustration of the transition from a local p2p node to a web based node.

Throughout the paper the terms “node”, “peer” and “host” are used interchangeably. The terms refer to the participating entities in a p2p network. The term “leaf node” is used to denote a node that is connected to a central node that controls at least a part of the operations of a network.

2 The peer-to-peer paradigm

2.1 Definitions

The p2p concept has been delimitated against client/server concepts [37], as well as against grid computing [11]. Explicit discussions on definitions [5, 16, 37] and frameworks for analyzing and classifying p2p applications have been offered [3, 5, 21].

Although there is an abundance of p2p definitions, the majority can be distilled into three concepts: decentralization, self-organization and resource-sharing. For an analysis of how these three concepts are interrelated, see Aberer and Hauswirth [1]. There is also a line of thought that emphasizes the locality and connectivity of the nodes participating in the investigated network [39]. This results in two sets of p2p criteria for classifying protocols and applications [11]. One emphasizes locality and connectivity of the nodes. This view does not consider the direct communication between nodes to be a constituting feature of the concept of p2p. The other gives prominence to decentralization, self-organization and direct communication between nodes.

2.1.1 Definitions based on locality and connectivity of nodes

A frequently cited definition of p2p is formulated by Shirky [39], who states that “peer-to-peer is a class of applications that takes advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer nodes must operate outside the DNS and have significant or total autonomy from central servers.” An application meets the p2p criteria if it allows for variable connectivity and temporary network addresses, and if it gives the nodes at the edges of the network significant autonomy.

Shirky points out that the novelty does not lie in the direct communication between nodes. The breakthrough consists of the chances to utilize previously unused resources offered by devices that bypass the Domain Name System (DNS). This definition encompasses systems such as Napster [29] and Seti@home [36], which rely on centralized servers for their operations.

2.1.2 Definitions based on decentralization and self-organization

In the present context, decentralization refers to topological properties. A decentralized network develops as a consequence of local decisions made by the nodes of the network on the basis of local information. This means that these networks have routing mechanisms of their own. Gnutella [13] and Freenet [22] are examples of networks that are designed to function in a decentralized way. In practice, many p2p network architectures in use represent a hybrid version which includes both decentralized and centralized features.

Self-organizing nodes frequently have the ability to build an application level network [32]. The nodes cooperate for the benefit of the whole network, and the result is an application that could not be provided by a single node. The management functions of the nodes are indispensable for the self-organization of the nodes. Graham explains that "the more management power we take out of a peer […] the more we degrade the p2p architecture [16]."

2.2 Areas of development and research

Unstructured p2p systems such as Gnutella [13, 14] do not engage in any book-keeping activities that would keep the nodes informed of the resources of the neighboring peers. Content is queried using flooding or random walks. These choices have caused a strain on bandwidth load. In order to solve the bandwidth problem, structured p2p systems such as CAN [31], Chord [40] and Pastry [35] were developed. These systems build distributed data structures based on peer identifiers. This facilitates query routing but render self-organizing more difficult. The self-organizing properties of the nodes give them the ability to build a virtual network on the application level [32]. Hybrid systems such as YAPPERS [12], P-Grid [2] and Structella [6] were developed in order to support self-organization while keeping the advantages of structured systems.

An important line of p2p research has focused on the development of algorithms and structures for efficient search in structured systems. One reason for the concentration on structured overlay networks is that unstructured systems such as Gnutella are associated with several problems concerning scalability and performance. These problems rise from the unstructured design and the resulting algorithms for message passing. Furthermore, the traffic on the Gnutella network does not map well to the physical network infrastructure [32]. Paradoxically, the weaknesses make the network interesting. They have given rise to a multitude of research projects concerning many issues related to decentralized networks, e.g. [7, 23]. One result of the investigations of the Gnutella network is the conclusion that in order for a network to function well, it cannot be fully decentralized. A hybrid which consists of both centralized and decentralized features is preferable [7, 10]. Another result concerns message routing. Many researchers, e.g. [19, 20, 26, 32] mention an unnecessary network traffic which has to be reduced. Massey et al. [26] recommend a restraint on ping, pong and push messages.

Another prominent line of research addresses the lack of interoperability between p2p networks. This stems from the fact that the majority of the protocols and interfaces are application specific. As a consequence, there is an extensive research aimed at creating standardized interfaces for p2p applications. The aim of JXTA [44] is to provide interoperability by offering a programming platform for p2p applications. Dabek et al. [8] offer a common API for structured p2p overlays, and Aberer et al. [3] propose a general reference architecture for overlay networks.

3 The Gnutella protocol

3.1 Gnutella messages

The Gnutella protocol consists of messages and rules. There are five different types of messages. Two of them, ping and pong, are used in order to register a node's presence in the network. These messages are required in case of a disconnection in the network. A node that has access to the information offered by the pong messages is later able to contact the registered peers. Two of the message types, query and query hit, relate to the search function of the protocol. The last message type, push, concerns the transmission of files [13].

The nodes also have a routing function. This is expressed through the rules of the protocol. The rules govern the way in which the nodes exchange messages. Ping and query messages are sent to all nodes that are situated on a certain distance from the source node. These messages are answered with a pong or a query hit message respectively. The answers are propagated back to the source node over the same nodes that forwarded the incoming message. The expression "search horizon" is used to illustrate the farthest set of hosts reachable by a search request. A Gnutella node's standard horizon is seven steps, usually referred to as hops [20].

In early implementations of the Gnutella protocol [13], ping messages were broadcasted over the network, which caused a considerable strain on network performance. Modern Gnutella servents cache pong messages [14], which means that a node does not forward ping messages to the nodes whose addresses are contained in the pong cache. This diminishes the number of recipients reached by a ping message. The frequency of the ping messages seems to pose a problem. The developers of Gnutellium suggest that ping messages comprise of approximately 50 % of the network traffic. They further state that one of the problems is that the investigated nodes seemed to send ping messages on a periodical basis. The protocol does not seem to contain any regulations as to the frequency with which to send pings. The specification draft for version 0.6 gives a somewhat alarming indication of the ping traffic. It states that a node must, if possible, respond with a number of pong messages, if it was at least one second since another ping was received on that connection. However, the specification draft also discusses the possibility to reduce the number of pongs by altering the protocol to support ping messages which include pong data.

The version 0.6 protocol suggests, that if ping messages containing pong data were to be used, the nodes do not have to discover neighbors, they can wait for the peers to announce themselves. The principle of waiting for messages from neighboring nodes has been applied by [6, 35]. In Structella, the Gnutella-like implementation of Castro et al. [6], each node expects a message from each neighbor in each 30 seconds period. If the node does not receive a message, it probes the silent node in order to investigate whether it still is present in the network. Furthermore, Structella as well as Pastry use every message sent, not only the ping and pong messages to register the presence of the network nodes. However, these solutions still expect the ping messages to be sent on a periodical basis.

3.2 The Gnutella network

3.2.1 Performance

The network performance is among other things jeopardized by the ping and pong messages. Group membership messages and search messages require a very large percentage of the bandwidth that is available to the net-

work [32]. Performance is also affected by the number of connections of each node and by the number of hops a message can propagate through the network. Each hop adds to the total bandwidth load.

3.2.2 Scalability

Gnutella was designed to support a few hundred to a few thousand users [41]. This is one of the explanations to the commonly held view that the Gnutella network does not meet the demands on scalability [19, 33, 43]. According to Ritter [34], the Gnutella network is unable to grow into an extensive network. These conclusions have been questioned by e.g. R. and G. Schollmeier [38].

One solution to the scalability problem is to modify Gnutella into a partly hierarchical network by using super peers [18]. Clip2 introduced the concept and realized it in the Clip2 Reflector program. The next step is a version of a Gnutella node that can operate in three different modes: peer, client and server. BearShare has implemented this idea [42].

4 Implementation of a transaction-based protocol

4.1 Intended use of the implementation

The application is intended as a prototype for testing and developing p2p systems. The node can be contained in separate computers that together form a network. It can also be used for simulations employing nodes that are all running on the same computer.

In order to facilitate testing, the messages are complemented with information about the nodes that send and forward messages. It is possible to monitor the individual nodes' activities. The implementation makes it possible to trace the routes of the messages. Thus, it is possible to discover at what point the forwarding or back propagating of a message has failed.

4.2 Elements of the local peer-to-peer node

The p2p node consists of the following structural entities: the router, the user interface (GUI) and three data structures that support the internal book keeping regarding messages, files and peers. The entities are illustrated in Figure 1. The hexagons represent the above mentioned data structures, which are referred to as tables. The broken lines between the router and the three tables connote the uses that the router makes of these data structures.

The router communicates with the user interface and the network in a symmetrical way. In both cases the communication is characterized by protocol messages and socket communication. The arrows between the router and the user interface and between the router and the network illustrate this communication. The oval named the Net connotes the peers of the node.

Four threads[1] maintain the activities of the node. Two are assigned to the GUI. One runs the user interface, and one maintains a socket listener for the user interface. A third thread contains the router listening for messages from the user interface as well as from the network. The fourth keeps track on when to send ping messages to the nodes that are registered in the peer table.

[1] The threads are encapsulated by a thread module created by Peter Nylund. The thread module utlizes the Future Class by David Perry[17].

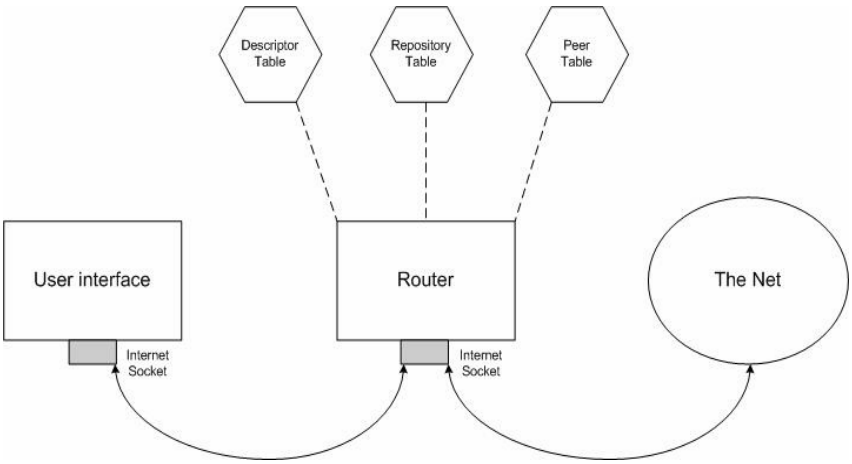

Figure1. Local node architecture

4.2.1 The graphical user interface

The graphical user interface is situated on the same local host as the router. It communicates with the router by sending queries and receiving query hit messages. The user interface is implemented with Tkinter. The GUI consists of a super class, which serves as a basis for any GUI application. The super class is responsible for opening and closing the windows as well as for the geometry management involved in placing the windows in their right position. The main program creates the root window, imports all the available modules and sets up the menus of the root window.

4.2.2 The router

The router receives messages both from the surrounding network and from the user interface. Regardless of the type of communication, the router is always involved as one party. There is only one entry point to the router, which means, that the user interface communicates with the router utilizing the same type of protocol messages that are exchanged between the router and the network. It is not necessary to define the messages coming from the user interface as a special message type. The messages from the interface are recognized as such, because the GUI has an identification number of its own, and the router and the GUI are aware of their respective identification numbers.

The router applies a symmetrical format and method for proceeding when replying to the messages that come from the network and to the messages that arrive from the user interface. A match to a search query coming from the interface can be found locally and/or in the network. When the router sends a query hit to the user interface, it uses the same format as for a query hit to the network.

4.2.3 Data structures

The file repository is an ordinary directory that contains the files and further directories that the person running the node wishes to share. The router utilizes local function calls to obtain the number of files and their total size as well as the name and size of every entry matching the search string in a query. Aside from the file repository, the node requires two additional data structures, a descriptor table and a peer table. The descriptor table contains the descriptor id, which is a 16–byte string uniquely identifying a message on the network. The value is preserved when messages are forwarded between nodes. When a node forwards a message, it adds the descriptor id of the message to the descriptor table together with information on the message type and the immediate address from which it was received. The peer table contains the addresses of the peers with which the node communicates directly.

4.2.4 The socket interfaces

The socket interfaces constitute the foundation for the p2p node. The user interface listens on a socket and the router listens on a socket. The router socket is the sole entry point to the router. The only communication between the threads of the node occurs through a socket interface.

4.3 Transaction-based connections

4.3.1 Connections between peers

A peer has the capability to function as a server, a client and a router, but these functionalities do not as such determine the type of network in which the node is located. The context of the node is the determining factor. The manner in which individual members of a set of nodes, NODE, are related to each other, can be described by the relation $connections \in \mathrm{P}(NODE \times NODE)$. A pair of nodes is connected if they are related by the closure of the relation $(connections \cup connections^{-1})$. The existence of a peer table, $peer\_table \in NODE \rightarrow \mathrm{P}(NODE)$, determines the routing capabilities and network awareness of a node. The power set includes the alternatives that the peer table contains the empty set, or that the node is only able to search in its own database when receiving a query.

4.3.2 A transaction-based design

According to Drucker and Kanane [9], a transaction is “a single conceptual transfer of high-level data or control. It is defined by its begin time, end time, and all the relevant information associated with the transaction.” A session is a durable relationship between application end points.

The Gnutella protocol is session-based. A Gnutella node connects to the network by establishing a connection with another node currently on the network. A node’s connections to other nodes are treated as open connections during the session. Once a connection is established, the Gnutella protocol maintains the connection by sending control signals in the form of ping and pong messages.

This paper introduces a transaction-based p2p node. It is to be noted that this design does neither presume nor involve UDP-communication. Every transaction is conducted via TCP-sockets. The router continuously listens for incoming messages, either from the user interface or from the network. The outgoing messages; however, are

sent on a transactional basis. The router opens a connection, it sends a message and closes the connection. One reason for applying session-based protocols is that they are well suited for large messages. In consequence, a transaction-based router entails a restriction in the size of the packets that are transferred between the nodes on the network. The length of user data in one Ethernet packet has to be less than 1500 bytes. In the IP layer of the OSI model, messages that are longer than 1500 bytes are fragmented during transmission.

A weakness of the session-based design is the risk that the receiver of a message becomes out of synchronization with the input stream. The descriptor header includes a field for payload length. This is the only way for a Gnutella node to find the beginning of the next message in the input stream. Since the messages consist of fixed-field binary packets, there is no guarantee that the node is able to register the loss of synchronization.

The transaction-based design also provides an opportunity to alter the way in which the ping and pong messages are utilized. This work does not implement the pong message. The pong data are added to the ping messages. Furthermore, an adaptive ping algorithm is implemented.

4.4 Signaling traffic

4.4.1 Ping payload

The present transaction-based node offers an opportunity to reduce the network traffic regarding pings and pongs. Furthermore, it alters the significance of the ping message as it is implemented on the basis of the Gnutella protocol. In a session-based context, the ping message extracts information about the presence of other nodes. In a transaction-based environment, every message constitutes a separate connection. Every successfully transferred message is a sign of life, whether it be a ping, pong, query or query hit that produces this sign. Furthermore, both the sender and the receiver benefit from this information. The sender can update its peer table regarding the receiver and, correspondingly, the receiver can update its information regarding the sender.

Since every successfully transferred message, independent of type, is a sign that the node is active, pong messages are no longer necessary for the sake of notifying nodes of active peers. This affects network performance because it reduces the load on bandwidth required. The information transferred by these messages is still needed. The present work adds this information to the ping message.

4.4.2 Adaptive ping timing

One of the problems regarding the session-based Gnutella node is that ping messages are sent frequently as well as periodically. The present paper presents an implementation of an adaptive algorithm for sending ping messages. The algorithm presumes the existence of a peer table containing the address of each node, a value for time and date, and an interval in minutes. The time interval is proposed to grow successively, e.g. in powers of two, starting with a one minute interval and ending with a 64 minutes interval. The interval in minutes indicates the length of the time elapsed between two pings sent to the same node.

A node that is connected to the network for long periods of time will be sent a ping at most every 64 minutes by the peers that have included this node in their tables. Since every successfully transferred message shows that the sending node is active, nodes that actively participate in the network seldom receive a ping. When a node does not accept a message, it is no longer active and; hence, it is removed from the peer table of the nodes that discover this fact.

4.5 Advantages of the Python programming language

The protocol presented in this paper is implemented with the Python programming language. Our intent in the following is not to compare the Python language to other languages. Furthermore, we do not claim to offer new programming code solutions or recipes. We make use of existing Python modules and functions in a way that is in accordance with established programming practice. However, we have chosen to conduct the implementation with Python, because various data types, modules and libraries offered by Python make it possible to limit the programming code to a manageable size. Thus, the following examples are presented in order to illustrate a few of the benefits of the language.

4.5.1 File operations

There is an inherent notion in the p2p concept that a node has to keep track on the files that it offers. In order to gather this information, the node has to explore the file system. This can be done by using the `os` and / or the `os.path` module. `os.listdir(path)` returns a list containing the names of all files and subdirectories found in the path directory. An added benefit of using this module is that we do not have to allocate memory for the list returned by the module function. The example in Figure 2 uses the `os.path` module, which contains functions for the analysis and transformation of path strings.

```
def amountFunction(arg, directory, names):
    global DIRsize, nrDIRfiles
    for name in names:
        tmp = os.path.join(directory, name)
        if os.path.isfile(tmp):
            DIRsize += os.path.getsize(tmp)
            nrDIRfiles += 1
```

Figure 2. Implementation using os.path

This function searches the File Repository and gives the number and total size of the files in the repository. The example is not meant as a new recipe. Equivalent examples are to be found in e.g. [17, 24, 25]. The example is intended to illustrate the statement that Python programming code can be kept at a manageable size.

4.5.2 Network protocol modules

Figure 3 is also intended as an illustration of the short and concise code made possible by the Python language. In order to send a ping message; a global variable is used, a ping object is created, a string is produced and subsequently sent to the address supplied.

```
def sendPing(address):
    global sessionid
    ping = PingMsg.Ping(SERVNR, LocalPORTasHex, LocalIPasHex, sessionid)
    msg = ping.constructMsg()       # the message string of a Ping object
    sendMsg(address, msg)
```

Figure 3. Implementation of the method concerned with the sending of ping messages

4.5.3 Sequence operations

When the router decides to forward a ping message that has arrived to the router, some of the values in the message string have to be substituted. This can be carried out by slicing, as in Figure 4. By referencing segments of the received message string, sections of the message sequence are extracted from the message and substituted by values that reflect the forwarding router and the message's remaining time to live.

```
def reactToPing(data, descrID, ttl, hops, retPort, retAddr):
    ttl = checkTTL(ttl, hops)
    if ttl >= 1:
   forwMsg = (data[0] + LocalPORTasHex + LocalIPasHex + data[7:24] +
                    chr(ttl) + chr(hops) + data[26:36])
        for key in Connections.keys():
            if Connections[key].hops == 1 and (retAddr, retPort) != key:
                sendMsg(key, forwMsg)
```

Figure 4. Example implementation that benefits from slicing

4.5.4 Data types

The data types tuples, dictionaries and lists substantially facilitate the handling of data structures. An added benefit is that the programmer does not have to allocate memory for the data structures. The Descriptor Table and Peer Table are both dictionaries that make use of the opportunity to gather objects of different types.

4.5.5 Conclusions based on the implementation of the local p2p node

The implementation has followed the praxis of the Gnutella protocol, which is characterized by a mixture of ASCII text and binary code. Gnutella opens every session with an ASCII encoded handshake. Once the node has connected to the network, it sends and receives packets that consist almost entirely of binary code. It is to be noted that the search string in a query message is assumed to be pure ASCII. Files are downloaded with the help of the HTTP protocol, which is ASCII encoded. A first proposal would be to rewrite the protocol to a pure ASCII (UTF-8) protocol. The program code would be shorter, easier to read and easier to debug.

The second proposal concerns the development of a web-based p2p node. It is inherent in this design, that since the user interface communicates with the router through sockets, it could connect to any router as long as it is aware of the router's address and listening port. This could be developed either into a network consisting of super peers and leaf nodes, or into a design that consists of fairly independent user interfaces that contact nodes for requests and download without providing any material of their own. This is one step from a web-based node.

4.6 HTTP as a means of peer-to-peer communication

4.6.1 The web-based peer-to-peer node

In the previously mentioned implementation of a transaction-based p2p protocol, the connections between the router and the GUI are characterized by socket communication and protocol messages. The next evolutional step would be to develop a web-based node, employing HTTP as a means of communication between p2p nodes. This conclusion has resulted in a web-based implementation, which is illustrated in Figure 5.

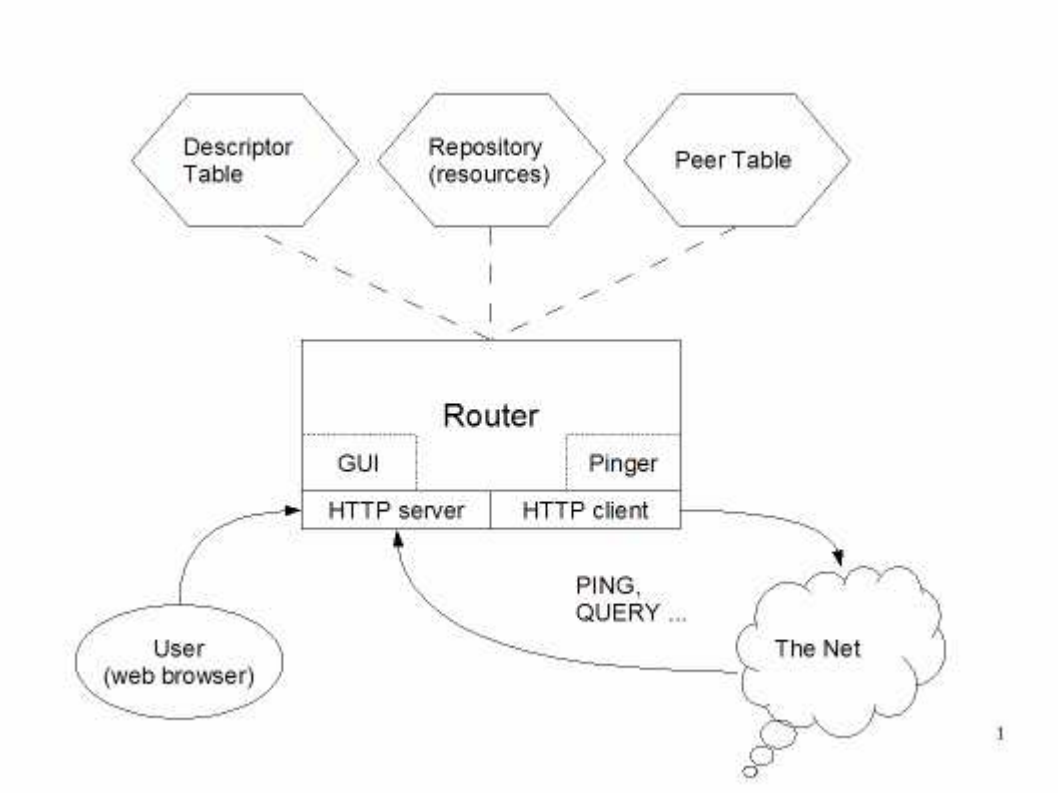

Figure 5. Web based node architecture

The hexagons in Figure 5 represent the data structures that facilitate the management of information concerning messages, resources and peers. As in the previous local node, the broken lines between the router and the three tables connote the uses that the router makes of these data structures. There is only one entry point to the router. The router examines the requested URL in order to respond to the received p2p message in an appropriate fashion.

The user communicates with the web server. The requests from the user enter the server and are handled by the router. The Net consists of the peers of the node. The peers are registered in the Peer Table. The unbroken arrows connote HTTP-communication. They also connote the direction of the communication messages that are sent.

4.6.2 Peer-to-peer messages contained in HTTP methods

The p2p messages are transferred by the HTTP protocol. The design employs three request messages, PING, QUERY, and GUI. The PING payload is contained in the POST method. Correspondingly, a dictionary is included in the POST method when a QUERY is sent. GUI is a message directed to the graphical user interface. The request messages give rise to text-based response messages as well as response messages that contain HTML. The GUI is the only message that results in a response message containing HTML. The response messages to pings and queries contain plain text. In this case, one line represents one hit. It is also possible to return the URLs of the hits in the response message to a PING or a QUERY. A small part (ping, query, GUI) of the requested URL informs the router how to respond to a requested message.

Table 1 gives an overview of the p2p messages, the HTTP methods and the corresponding URLs. The term node used in Table 1 connotes the server address in the form `host_name: port_number`.

P2P Messages	Method	URL
PING	`HTTP POST`	`http://node/ping`
QUERY	`HTTP POST`	`http://node/query`
GUI	`HTTP POST/GET`	`http://node/GUI`

Table 1. P2P messages expressed as HTTP messages

4.6.3 Comparisons between the local node and the web-based node

The local node described in section 4 consists of a router, a graphical user interface and three data structures. The transition from a local node to a web-based one leaves the Descriptor Table and Peer Table unchanged. The Repository differs from the Repository Table in Figure 1. The local node offered information on files, while the web-based node offers information on / links to resources.

The Tkinter implementation of the graphical user interface in the local node is substituted with HTML. However, the same features regarding search arguments and results are preserved.

The socket-based messages in the implementation of the local node are sent to a specific socket. The same information is sent with the payload in the POST method. This method was designed to send input data to a server, although the current use is to support HTML forms [15]. Since the communication is conducted with HTTP, the message classes of the local node implementation are not required in the web-based solution. The number of threads is reduced to a minimum of two threads: one for the ping generator and one for the router.

As a summary of the changes can be stated, that the resulting programming code is remarkably shorter than the code that constitutes the local node.

5 Concluding remarks

This paper can be summarized as ”a peer-to-peer protocol design and its consequences.” The design refers to the implementation of a local p2p node. This design creates a basis for further development of peer-to-peer systems. As a consequence of this, a web-based p2p node is developed. Thus, the paper presents two p2p implementations. The first implementation takes the Gnutella protocol as its starting point. The reason for this is that the Gnutella protocol is a simple, well-documented open protocol. However, the presented implementation differs from the Gnutella protocol regarding intent as well as routing solutions. The main features of this implementation are as follows: the transaction-based design, the adaptive ping algorithm, and the router’s symmetric way of communicating with the GUI and the surrounding peers. The transaction-based design of the local node points in the direction of the HTTP protocol. Based on this conclusion, a web-based p2p node has been developed and presented. Both implementations are characterized by a transaction-based design, and both implementations make use of the adaptive ping algorithm. Needless to say, the swift transition from a local p2p node to a web-based node also serves as an illustration of the flexibility and simplicity of the Python programming language.

The adaptive ping algorithm is based on the assumption that some nodes are more stable than other nodes. There are nodes that connect to the network for short periods of time, and there are nodes that remain connected to the network for a substantial amount of time. The adaptive ping algorithm assumes, that an active node that is

considered stable will remain active in the near future. We have not yet explored these assumptions. We intend to test the adaptive ping algorithm in a laboratory environment.

When a stable node ceases to be reachable, it might still be included in the peer tables of its peers during the next hour. However, this is only the case as long as the other nodes do not see any reason to contact this node. A consequence of the transaction-based design is that every message constitutes a separate connection. The router opens a connection, it sends a message and closes the connection. If the addressee is not active, it is not possible to open the connection. The sending node is immediately informed of this fact and updates its peer table correspondingly.

It is to be noted that HTTP is a stateless protocol. Thus, the communication is stateless; however, the p2p nodes are not stateless. The nodes keep track on their environment and send pings to their peers according to the given algorithm. A consequence of utilizing HTTP is that a user does not have to be in the presence of the computer running the p2p node. This creates a basis for a further development of p2p systems.

It has not escaped our notice that the presented implementations might lead to the application of the p2p concept in more mundane surroundings than the Gnutella protocol offers. The implementation, which concerns the local node is intended as a testbed for ideas regarding the utilization of the p2p paradigm. The implementation based on HTTP has been developed in a laboratory environment. The design will be further tested in the same environment.

Acknowledgments This work has been supported by the Åbo Akademi Foundation.

References

[1] K. Aberer, M. Hauswirth. *An Overview on Peer-to-Peer Information Systems*. Workshop on Distributed Data and Structures (WDAS – 2002), Paris, France, 2002.

[2] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva and R. Schmidt. P-Grid: A Self-organizing Structured P2P System. *In SIGMOD Record, 32 (2)*, September 2003.

[3] K. Aberer, L. Onana Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi and M. Hauswirth. The essence of P2P: A reference architecture for overlay networks. In *Proceedings of the Fifth International Conference on Peer-to-Peer Computing* (P2P2005) 31 August – 2 September 2005, Konstanz, Germany.

[4] D. Anderson, SETI@home, In A. Oram, (ed.) *Peer-to-Peer. Harnessing the Power of Disruptive Technologies.* Cambridge 2001, pp 67 - 76.

[5] S. Androutsellis-Theotokis and D. Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies, *ACM Computing Surveys, Vol. 36, No. 4*, December 2004, pp. 335 – 371.

[6] M. Castro, M. Costa and A. Rowstron. Should we build Gnutella on a structured overlay? *ACM SIGCOMM Computer Communication Review, 34* (2004) 131-136.

[7] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *Proceedings of ADM SIGCOMM*, August, 2003.

[8] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, February 2003.

[9] L. Drucker and K. Karnane. How transaction-based verification works. *EEdesign.com*, 2002. http://www.eedesign.com/features/exclusive/oeg20020322S0101

[10] J. Epstein. *Peer-to-Peer is Changing the Internet.* Drexel University. http://www.jeffyjeffy.com/

[11] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, February 2003, Berkeley, CA.

[12] P. Ganesan, Q. Sun and H. Garcia-Molina. YAPPERS: A Peer-to-Peer Lookup Service over Arbitrary Topology. In *Proceedings of the IEEE INFOCOM 2003*, San Francisco, CA, USA, March 30 – April 3, 2003.

[13] The Gnutella Protocol Specification v0.4.
http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

[14] The Gnutella Protocol Draft Version 0.6
http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html

[15] D. Gourley and B. Totty; with M. Sayer, S. Reddy and A. Aggarwal. *HTTP. The Definitive Guide.* Sebastopol, CA, 2002.

[16] R.L. Graham. Peer-to-Peer. Toward a Definition. *Proceedings of the 2001 International Conference on Peer-to-Peer Computing (P2P'01)*, Linköping, Sweden, 27-29 August 2001.

[17] D. Harms and K. McDonald. *Quick Python Book*. Greenwich, CT, 2000.

[18] T. Hong, Performance, In A. Oram, (ed.) *Peer-to-Peer. Harnessing the Power of Disruptive Technologies.* Cambridge 2001, pp 203-241.

[19] I. Ivkovic. *Improving Gnutella Protocol. Protocol Analysis and Research Proposals.* Technical Report, LimeWire LLC, 2001. http://www9.limewire.com/download/ivcovic_paper.pdf

[20] G. Kan, Gnutella, In A. Oram, (ed.) *Peer-to-Peer. Harnessing the Power of Disruptive Technologies.* Cambridge 2001, pp 94 – 122.

[21] K. Kant, R. Iyer, and V. Tewari. A Framework for Classifying Peer-to-Peer Technologies. *2nd IEEE/ACM Intl. Symposium on Cluster Computing and the Grid*, May 21-26, 2002, Berlin, Germany.

[22] A. Langley, Freenet, In A. Oram, (ed.) *Peer-to-Peer. Harnessing the Power of Disruptive Technologies.* Cambridge 2001, pp 123 - 132.

[23] Q. Lv, S. Ratnasamy and S. Shenker. Can Heterogeneity Make Gnutella Scalable? In *Proceedings of IPTPS'02*, Cambridge, MA, USA, February 2002.

[24] A. Martelli. *Python in a Nutshell. A Desktop Quick Reference*. Sebastopol, CA, 2003.

[25] A. Martelli and D. Ascher, (eds.). *Python Cookbook*. Sebastopol, CA, 2002.

[26] R. Massey, S. Bharath and A. Jain. *Gnutella-Pro. What Bandwidth Barrier?* Course Project 2001. UCSD Jacobs School. Department of Computer Science and Engineering.
http://www.cs.ucsd.edu/classes/wi01/cse222/projects/reports/gnutellapro-5.pdf

[27] N. Minar. *Distributed Systems Topologies. Part 2,* Published 2002, http://www.openp2p.com/lpt/a/1461.

[28] A. Montresor. *Introduction to Peer-to-Peer.* Lecture Notes. Department of Computer Science, University of Bologna, 2003.

[29] The Napster web site, http://www.napster.com.

[30] A. Oram, (ed.) *Peer-to-Peer. Harnessing the Power of Disruptive Technologies.* Cambridge 2001.

[31] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM'01,* San Diego, August 2001, pp. 149-160.

[32] M. Ripeanu. Peer-to-Peer Architecture Case Study. Gnutella Network. *In Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*, Linköping, Sweden, 27-29 August 2001.

[33] M. Ripeanu, A. Iamnitchi and I. Foster. Mapping the Gnutella Network. *IEEE Internet Computing*. 2002. http://people.cs.uchicago.edu/~matei/PAPERS/ic.pdf

[34] J. Ritter. *Why Gnutella Can't Scale. No Really*. http://www.darkridge.com/~jpr5/doc/gnutella.html

[35] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001),* Heidelberg, Germany, November 2001.

[36] The Seti@home project web site, http://setiathome.ssl.berkeley.edu.

[37] R. Schollmeier. A Definition of Peer-to-Peer Networking towards a Delimitation Against Classical Client-Server Concepts. In *Proceedings of the 7th EUNICE Open European Summer School (EUNICE'01) and the IFIP Workshop on IP and ATM Traffic Management (WATM'01)*, Paris, France, September 3 – 5, 2001.

[38] R. Schollmeier and G. Schollmeier. Why Peer-to-Peer (P2P) Does Scale. An Analysis of P2P Traffic Patterns, In *Proceedings of the 2nd International Conference on Peer-to-Peer Computing (P2P'02),* 5-7 September 2002, Linköping, Sweden, pp. 112 - 119.

[39] C. Shirky. Listening to Napster, In A. Oram, (ed.) *Peer-to-Peer. Harnessing the Power of Disruptive Technologies.* Cambridge 2001, pp 21 - 39.

[40] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In Proceedings *of ACM SIGCOMM'01*, San Diego, August 2001, pp. 160-177.

[41] K. Truelove. Gnutella: Alive, Well, and Changing Fast. *The O'Reilly Network*, Published 2001. http://www.openp2p.com/lpt/a/573

[42] K. Truelove. Gnutella and the Transient Web. *The O'Reilly Network*, Published 2001. http://www.openp2p.com/lpt/a/705

[43] B. Wiley. Interoperability through Gateways. In In A. Oram, (ed). *Peer-to-Peer. Harnessing the Power of Disruptive Technologies.* Cambridge 2001, pp 381 – 392.

[44] B.J. Wilson. *JXTA.* Indianapolis 2002.