# Improving Gnutella Search Algorithms through Epidemic Dissemination

Holger Kampffmeyer

supervised by Cecilia Mascolo

Computer Science Department,
University College London

March 23, 2006

This report is submitted as part requirement for the MSci Degree in Computer Science at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

**Abstract**

Search algorithms in unstructured P2P networks such as Gnutella use flooding-based techniques for communication and as a consequence, they produce high message overhead. More dynamic algorithms such as Gnutella's Dynamic Query Protocol take into account the user's desired number of results and network topology properties to increase scalability. However, these algorithms only work well for popular files and often fail in locating rare content. Proposed structured approaches such as DHTs are good in finding rare files, but due to their significant overhead and problems with high network fluctuations, they are not very applicable for finding popular content.

In this report, we propose a search algorithm based on epidemic-style information dissemination techniques, that shows good performances in finding both rare and popular files. It exploits the structure of the underlying network and by that maximizes its search horizon and minimizes the number of needed search messages. The presented simulation results show that the search algorithm not only works well in Gnutella-like networks, but would be also applicable in a much broader context such as scale-free networks.

# Contents

# List of Figures

# 1 Introduction

Peer-to-peer (P2P) systems have attracted a great deal of attention in recent years. While users fancy the possibilities of unlimited sharing and exchange of files, P2P systems also attract the research community. Much research has been carried out in the area of topology characteristics and measurements, and the development of faster and better search algorithms. P2P networks take a totally new and different approach to network structure, in contrast to traditional client-server networks like the Internet. [7] describes a server as a software application that provides a service on behalf of other software applications called clients. In a client-server architecture, a high number of clients make use of the service of a single server. In contrast to conventional client-server architectures, a pure P2P network does not require a server. Peer nodes in a P2P architecture provide both server and client services and contribute resources such as sharable data, computing power, bandwidth and storage space to the network. The main design principle of a P2P network aims at being completely decentralized and self-organized. It is more robust than client-server networks in case of failures because of the replication of data over multiple peer nodes and the local decision-making of individual nodes. Although P2P networks can host a wide range of applications and services, they are mainly used for sharing files (audio, video, images, other digital data), and for transporting realtime data such as telephone traffic [6].

The lack of a centralized server comes at a high cost: the communication overhead needed for maintaining the network, for finding other peers, and the overhead produced by search messages distributed over the network for finding sharable data, are significant. Therefore, a second form of P2P systems has been developed: hybrid P2P networks. Hybrid P2P networks use several known central servers for network maintenance and search, and clients that provide the sharable content of the network. This form combines the best features of both architectures. The network overhead caused by inter-peer communication is reduced, while the robustness of a distributed

network and the resulting availability of files is still maintained. However, the maintenance of central servers is expensive and surely not scalable. We therefore propose a decentralized, unstructured search algorithm, which uses epidemic-style information dissemination techniques, to control the number of query messages and to fine-tune the dissemination process. The approach is based on recent results on complex networks theory and models of epidemics spreading. We present simulation results that show that our approach works efficiently not only in Gnutella-like P2P networks, but would be also applicable in a much broader context such as scale-free networks.

The thesis is structured as follows: In Section 2 we will describe several common search algorithms in modern P2P networks, which all have their advantages and shortcomings. Either these algorithms produce a high message overhead, which make them applicable only for very restricted use cases, or a query is not exhaustive and can not guarantee that a file that is in the network is actually found. We will introduce the concept of epidemic information dissemination, which is inspired by epidemiological research and based on recent results of complex network theory in Section 3. In Section 4 we will show how to develop an information spreading algorithm, which dynamically evaluates network properties to fine-tune the dissemination of the information. We will further develop this algorithm as a search algorithm in a Gnutella-style P2P network and present simulation results of both the original epidemic algorithm and the epidemic search algorithm in Section 6, We compare the Epidemic Search algorithm in this Section with Gnutella's current search algorithm and with Gnutella's old flooding-based search algorithm. The evaluation shows that our algorithm produces better search results for rare files, and produces less overhead in finding popular files compared to Gnutella.

# 2 Search Algorithms in Unstructured and Structured Networks

The location of items in a network is an important part in the operation of P2P networks. All popular P2P applications [9] work on `Unstructured` networks. Unstructured networks connect nodes in an ad-hoc fashion. The location of files is not controlled by the system. Therefore, there is no guarantee for the success of a search. `Structured` networks, on the other hand, use content-based routing of query messages. Every node is assigned an identifier and messages are forwarded only according to these keys. The network coordinates global agreement on the location of a file. A query labeled with a specific key is routed to a particular node in the network that stores the desired file. While being highly reliable in locating items in the network, the overhead produced of "publishing" files by keyword is significant [38].

In this section, we describe search algorithms in P2P networks, which have been either successfully used in popular P2P applications, or are related to our work. We describe the modern Gnutella system in Section 2.1, because our search algorithm and proposed protocol is highly influenced by this system. One of the most popular file sharing applications, eMule, is described in Section 2.3. As an example of an probabilistic search algorithm which tries to decrease the message overhead produced by flooding based techniques used in Gnutella, we introduce Random Walks in Section 2.2. Gnutella, eMule and Random Walks operate on unstructured networks. To complete this section, we describe `Distributed Hash Tables` (DHT), which operate on structured networks, in Section 2.4.

## 2.1 Gnutella

The Gnutella P2P network [4], with a user base of over 2.2 million users in March 2006 [9], is one of the top three popular P2P file sharing applications, primarily used to exchange music, movies and software. Much research has been done in characterizing its network topology [36, 37] and due to its open

protocol specification [3], a healthy developer base is still developing and maintaining Gnutella applications. We will describe the general information exchange mechanism between Gnutella peers in Section 2.1.1, the architecture of the two-tier topology in Section 2.1.2 and the actual search algorithm, `Dynamic Query Protocol` (DQP), in Section 2.1.3.

### 2.1.1 Protocol Services

The Gnutella protocol is a pure P2P protocol in the sense that its nodes operate both as servers and clients. These `Servants` offer a client-side interface which is used by users to issue queries and to receive search results. On the other hand, a Gnutella node acts as a server which accepts search queries from other servants. It checks these queries against its local data set and responds with corresponding search results. It also manages the network traffic for spreading the information which is needed to maintain network integrity. Like peers in other P2P networks, Gnutella servants operate through messages, defined in a special protocol. The protocol defines messages for discovering and maintaining connections (`PING, PONG`) to the network and messages for searching (`QUERY, QUERYHIT`). The structure of the messages is very similar to standard `HTTP` messages as specified by the W3C [5], and can be divided into two groups:

- Group Membership messages: A Gnutella servant that wants to join the network initiates a broadcasted PING message to announce its presence. Every Gnutella message contains a `Time To Live`(TTL) value that limits the lifetime of the message. Each time a node receives a PING message it decreases its TTL and forwards it to all known neighbours. As soon as the TTL reaches 0, the message expires and gets deleted from the network. The servant reacts to the PING message by answering with a PONG message that contains information like its IP address and number and size of shared files.

- Search messages: When a user wants to search for a file, a QUERY message containing the user specified search string is sent to every node the

peer is connected to. A node receiving the QUERY message matches the search string against the locally stored file name and broadcasts the message further. If there is a match, the node answers with a QUERYHIT message. A QUERYHIT message contains information necessary to download the file, like the number of hits, the IP address of the responding node and the download speed.

For actually downloading a file a standard HTTP connection is established (`GET, PUSH`). This means that the download process is not directly part of the Gnutella protocol. A file download must be initiated by the user, who inspects the list of search results delivered by a QUERYHIT and chooses the sources he wants to download from.

Before the introduction of the Gnutella protocol v0.6 in 2002 [4], every node in the network played the role of an equal peer. This resulted in the overloading of peers with low bandwidth connections and generally in an overhead of messages, because every node had to forward every message to every neighbour. These scalability problems were improved with the introduction of the `two-tier overlay architecture`.

### 2.1.2 The Two-tier Overlay Architecture

With the introduction of the two-tier overlay architecture, the Gnutella developer community [3] implemented specialized roles for peers in the network. Figure 1 shows the general architecture of this model. A small subset of nodes becomes `ultrapeers`, responsible for routing all messages and for shielding the `leaf peers` that are connected to them from the network traffic.

To become an ultrapeer, a node must provide sufficient bandwidth, must not be firewalled [1], and must provide sufficient uptime [2]. Ultrapeers are av-

[1] A firewalled node is a computer residing behind a firewall that blocks certain incoming traffic. By that the firewall hinders the ultrapeer from communicating with other peers in the network.

[2] Uptime is the time an ultrapeer is connected to the network and stays reachable for other peers.

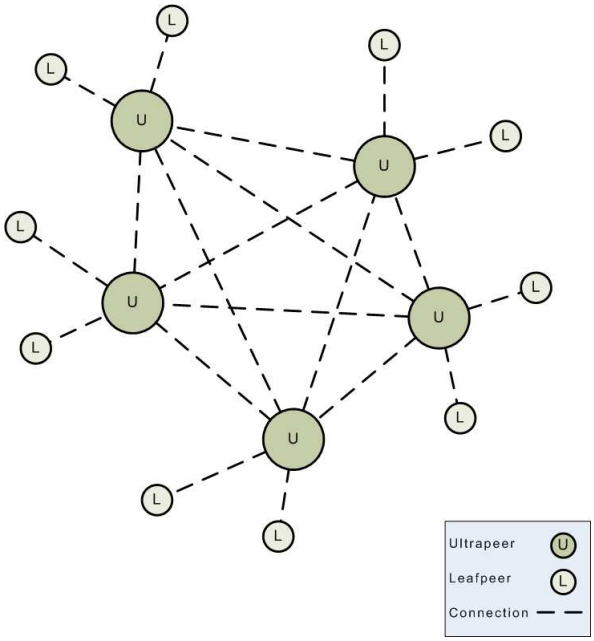



Figure 1: The two-tier topology in Gnutella

eragely connected to 30 other peers [35, 37], while leaf peers hold only a small number of connections ($<= 3$) to ultrapeers. Leaf peers, on the other hand, concentrate on providing files. Information about the files the nodes share are uploaded to the ultrapeer. An ultrapeer forwards only those QUERIES to its leaf peers that have matching files. The introduction of the two-tier topology was the first step to improve the scalability and performance of Gnutella by reducing search message overhead. Another reason, why Gnutella had these performance issues, was the simple flooding algorithm for propagating messages over the network. The `Dynamic Query Protocol` further reduced message overhead by considering the popularity of the queried file and reducing the used message TTL.

### 2.1.3 Dynamic Query Protocol

The traditional Gnutella query protocol has broadcasted every search query in a flooding based manner. In this algorithm, a message is sent to every neighbour of a node, until the message TTL reaches 0. Figure 2 shows an

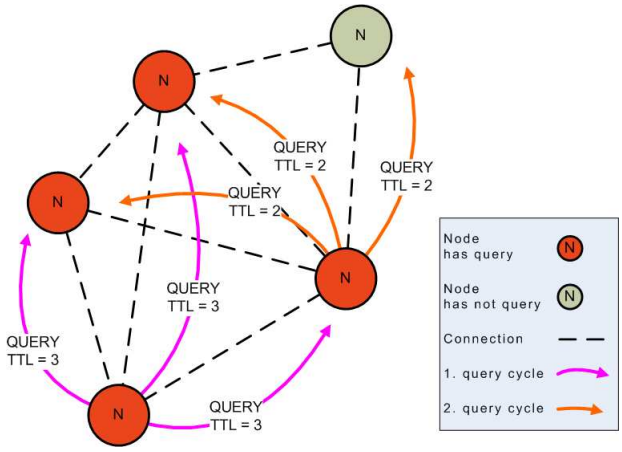



Figure 2: The flooding of messages in Gnutella

example, of how flooding works. In a first cycle, a message with an assumed TTL of 3 is flooded from the bottom node to all its neighbours, leaving only one node in this example without having seen the message. In the second cycle, all nodes but the "grey" node will see the same message a second time, which results in a huge message overhead.

In the old Gnutella network, a query did not adapt to different file popularities. The number of sent messages was the same for popular and rare items, resulting in far too many results for popular items. These issues were improved by the Dynamic Query Protocol [4, 37]. The goal in this approach is to only gather enough results to satisfy the user's needs (typically 50 to 200 results). The DQP delegates all search responsibilities to the ultrapeers. Every leaf peer commits its file list to the ultrapeer it is connected to. When an ultrapeer receives a search query from an attached leaf node, it starts a dynamic query which consists of two steps: first it sends a query message with a low TTL that is flooded outwards, decreasing the TTL each time it is forwarded until the TTL expires. This is called a probe query, meant to give some idea of the popularity of the file. Depending on the results the ultrapeer receives, a second query is started with a higher TTL. This process is repeated until the desired number of results is reached or the ultrapeer

gives up. [29] shows the effectiveness of the DQP in finding popular content and a poor performance in finding rare files. In Section 6 we confirm these results and compare them with our algorithm.

## 2.2 Random Walks

Random walks have been proposed for searching and construction of unstructured P2P networks [23, 38]. They have shown a better performance and scalability than flooding-based algorithms. We mention Random Walks in this section, because they are quite similar to our approach. Both use probabilistic means in reducing communication overhead. Random walks show constant network overhead, meaning they are independent on the size of the network. Queries, using random walks, are also known as walkers. Each node on receiving a query, forwards it to an equal number of randomly chosen neighbours and by that, the query forms a random path through the network. Random walks can be either TTL-based or check-based. TTL-based random walks stop the query after the number of steps specified by the TTL. A check-based walker contacts periodically the querying node and stops the query after the termination condition is satisfied. The most important advantage of random walk algorithms over flooding-based algorithms is the reduction of message overhead, by contacting only a subset of neighbours. [38] has found that only $k * TTL$ messages are sent in the worst case. The disadvantage, on the other hand, lies in the highly variable performance of random walks. Its success rate and number of query hits depend on the network topology and the hop distance, in the case of TTL-based random walks. Furthermore, random walk algorithms are not adaptable to different query loads. Whether a file is popular or not is not considered by these algorithms; all queries are treated the same.

## 2.3 eDonkey2K/eMule

eMule, based on the eDonkey protocol [28, 24], is the second popular P2P file sharing application [9]. Because it uses servers for centralized indexing services, eMule belongs to the group of hybrid P2P systems. A client connects

to a single known server to get information about files and other clients. It then connects to multiple other clients to upload and download files. eMule supports multi-source downloads, that is a file can be downloaded from several sources at the same time. This is possible due to the separation of the file into different parts that are individually downloaded. The different fragments are then joined together to the final file after the download is completed. A client is able to upload finished parts of a file, so the sharing of a file starts even if it is not yet completed. To further encourage the wide sharing of files, eMule has introduced a credit system that rewards users for sharing files. The more files a user uploads the more credits the user receives and the faster he advances in the waiting queue of other users. A waiting queue is needed because each client only has a restricted number of slots other clients can use to connect to this user. The quicker a user advances on a waiting queue, the faster he can start the download.

The search for files is initiated by the user. Each search request is sent to the server a client is connected to. In contrast to Gnutella, eMule files are not identified by their file name but by a MD4 hash value [34]. This makes it possible for the server to clearly identify a file even if it is shared under several different names. The central server based search in eMule is the main difference from Gnutella, where each query is forwarded from one Gnutella servant to another. This forwarding of search messages produces a considerable amount of the Gnutella message overhead. Because of the relatively low number of eMule servers the overhead produced by search messages is not a problem in eMule. [24] finds that there is a different drawback in the communication between eMule servers: the communication protocol between eMule servers is UDP [32], which is a stateless protocol. Therefore, an eMule server can not check if another server is still running. It keeps sending messages and by that causes traffic overhead. Finally, because any server can handle only a certain number of users, this approach is not scalable. New servers have to be introduced with an increase in user numbers. Furthermore, if a server goes down, the network is severely handicapped [2].

## 2.4 DHT and Hybrid Search Approaches

As an example of algorithms working on structured networks, we describe `Distributed Hash Tables` (DHT), a class of distributed lookup services for efficiently finding resources in a network [33, 19]. Each node of the network represents a bucket of a hashtable, where keys map unique identifier to nodes that contain a resource. DHTs consist of two components: a `keyspace partitioning` scheme that splits the ownerships of keys among nodes and a `overlay network` that connects nodes and allows to find the owner of a given key.

To store a file in the DHT, a node calculates the hash value $key$ of a file with content $data$ and sends a message $(key, data)$ to all its neighbors. The message gets forwarded until it reaches the single node that is responsible for the key, according to the keyspace partitioning. In order to find a file for a given hashvalue $key$, the search message is routed to the node responsible for that key, which responds with the stored data. The keyspace partitioning is a technique for a node with only local knowledge to decide which neighbour in some sense is closer to the destination of a file. The DHT network automatically adjusts the mapping of keys and neighbor tables when the set of nodes changes. Most DHTs guarantee that routing completes in $O(logN)$ hops for a network with $N$ nodes, while Gnutella requires $O(N)$ steps to reliably locate a specific file [17]. The first DHTs were designed to scale to large number of nodes and to overcome scalability problems of such systems as the old Gnutella. The `Content-Addressable Network` (CAN) developed by [33], maintains a $d$-dimensional coordinate routing table that holds the IP address and a virtual coordinate of each immediate neighbour of a node. Another DHT system is `CHORD` developed by [19], who use consistent hashing [27] to map nodes onto an $m$-bit circular identifier space.

[29] proposes a hybrid query processor, combining the Gnutella search algorithm for popular files with a DHT-based search for rare content. This is motivated by measurements which suggested that Gnutella is highly ef-

fective for locating popular files, but performs poorly in finding rare items. Due to the fact that DHT-based search is bandwith-expensive, a partial index for locating rare items has been proposed. To decide, which items are rare and need to be published, several schemes has been analyzed. Each ultrapeer node gathers information about results generated with certain keywords to decide if a file is rare and needs to be published via DHT. However, this approach lacks elegance. It needs two separate overlay networks: an unstructured Gnutella network for popular files and a structured DHT overlay for rare files. We introduce the concept of `Epidemic Information Dissemination` in the next section, which we use to design an universal search algorithm in Section 4 which works both for rare and popular files.

# 3 Epidemic Information Dissemination

Epidemic algorithms are a potentially effective mechanism to disseminate information in distributed systems, particularly in P2P networks. They exploit epidemiological research results on the spread of contagious diseases. There is a strong analogy between information dissemination in distributed systems and epidemic transmission in communities. Diseases are transmitted by contact between infected individuals and susceptible (i.e. potentially infectable) individuals. The spread of an epidemic can be used to model replication and dissemination of information in a P2P system. The highly resilient nature of epidemics can be used to ensure the reliability and robustness of an epidemic algorithm. In a community, only a few people need to spread the disease to make it almost impossible to wipe out the epidemic again. Even if many infected people die before they can transmit a virus, the epidemic will reliably propagate throughout the population. The analogy becomes even more evident when the content of the information is malicious as in the case of computer viruses [14, 18]. According to [13], epidemiologists have found that there is a critical threshold for the propagation of a disease throughout a population. Any epidemic less infectious than this threshold will inevitably die out, whereas those above the threshold will multiply exponentially. For Scale Free networks this threshold is zero. That is, all viruses, even those that are weakly contagious, will spread and persist in the system.

Epidemic-style algorithms and protocols have been subject of research in various areas such as distributed computing [40, 21], distributed databases [20], and ad-hoc networks [39, 12]. Not only are they highly scalable, but also easy to deploy, robust and resilient to failure. Through the adjustment of the parameters of an epidemic algorithm, even in systems with a dynamic network topology, process crashes, disconnections and packet losses, a high reliability can be achieved. Epidemic algorithms rely on probabilistic message replication and redundancy, to ensure reliable communication among nodes. However, these approaches are mostly based on empirical experiments, not analytical methods. The input parameters to control the dissemination pro-

cess are selected on experimental results, not on mathematical models. As a result, the information dissemination can not dynamically be tuned with accuracy. [21] give an overview on epidemic algorithms in distributed systems. They describe key problems such as membership maintenance, network awareness, buffer management and message filtering, where epidemic algorithms are applicable. [42] propose a protocol for disseminating information in ad hoc networks and p2p networks. However, their work is in an early stage. The future goal of their work is a hybrid approach of DHTs and epidemic dissemination.

Complex network theory is the means to describe social, biological and computer networks, were nodes represent individuals or hosts and links represent their interactions. For example, the brain is a network of nerve cells connected by axons, whereas societies can be seen as networks of people linked by friendships, familial relationships and professional ties. Complex networks can be characterized by their distribution of node linkages: `exponential networks` and `scale free networks`. The simplest example of an exponential network is the random graph model [15, 10]. The placement of links between nodes is completely random and most nodes will have approximately the same number of links. Therefore the connectivity distribution $P(k)$ follows a Poisson distribution with its characteristical bell shaped curve, which peaks at an average value $\langle k \rangle$ in this model. Random networks are called exponential, because the probability that a node is connected to $k$ other nodes decreases exponentially for large $k$ [13]. Exponential networks are characterized by very small fluctuations (i.e., the degree of every node can be approximated as $k \approx \langle k \rangle$); for this reason, they are also identified as `homogeneous` networks. This corresponds to the `homogeneous mixing` assumption that is usually made by epidemiologists in a large number of studies [11]: all individuals in the population have the same number of acquaintances that can be infected. On the other hand, for the inherent fluctuations of the degree of connectivity, scale-free networks are classified as `heterogeneous` networks. Examples of scale free networks are the World Wide Web and the Internet [13]. Scale free networks show a power-law con-

nectivity distribution $P(k) \sim k^{-2-\gamma}$ with $\gamma > 0$. In this model, nodes of the network are not considered equal. When deciding where to establish a link, a new node prefers to attach to an existing node that already has many other connections. This behavior leads to a network dominated by hubs, a low number of nodes having an enormous number of links, while the majority of nodes in the network has only a few connections to other nodes. We outline network topology characteristics of the Gnutella network and analyze the behavior of our algorithms in homogeneous and heterogeneous networks. In Section 4 we show that an epidemic algorithm based on epidemic models and recent results of complex network theory can be designed, which dynamically adapts to the underlying network topology and controlls the number of message replicas spread around the network.

## 3.1 The Infection Spreading Model

The Epidemic dissemination algorithm that we adopt as a basis for a search algorithm in a Gnutella-like network, is based on the work of [31], which they have applied in `Mobile Ad-hoc Networks` (MANETs). The core of the Epidemic dissemination algorithm is an infection spreading model, proposed by Kermack and McKendrick in 1927. A simplified model which is used by [31] is the Susceptible-Infective-Susceptible `SIS` model [1, 11]. An individual can be in two possible states; infected (i.e., an individual is infected with a disease), and susceptible (i.e., an individual can potentially get infected). The model is mapped onto P2P networks, by substituting the individual with a host (a node in the network). A host is considered infected, if it holds the message, and susceptible, if it does not. If the message is deleted from the host, the host becomes susceptible again. The assumptions which [31] make for their model in MANETs are also valid for P2P networks. The dynamics of the infectives and susceptibles in a scenario composed of $N(t)$ active hosts (i.e., not failed), can be described by means of a system of differential equations:

$$\begin{cases} \dfrac{dS(t)}{dt} = -\beta S(t)I(t) + \gamma(t)I(t) \\ \dfrac{dI(t)}{dt} = \beta S(t)I(t) - \gamma(t)I(t) \\ \dfrac{dN(t)}{dt} = -\phi N(t) \\ S(t) + I(t) = N(t) \end{cases} \tag{1}$$

where:

- $I(t)$ is the number of infected hosts at time $t$;
- $S(t)$ is the number of susceptible hosts at time $t$;
- $\beta$ is the average number of contacts with susceptible hosts that leads to a new infected host per unit of time per infective in the population;
- $\gamma$ is the average rate of removal of infectives from circulation per unit of time per infectives in the population;
- $\phi$ is the failure rate (i.e., the probability that one host fails per unit of time).

If we solve the system by using the initial condition $I(t) = I_0$ (where $I_0$ is the number of initial hosts infected), we obtain that the number of infectives at time $t$ is described by the following equation:

$$I(t) = \frac{I_0 e^{\alpha\beta t}}{1 + \dfrac{I_0}{\alpha}(e^{\alpha\beta t} - 1)} \tag{2}$$

with $\alpha = N(t) - \dfrac{\gamma}{\beta}$. $N(t)$ is considered approximately constant during the entire epidemic process described by the system 1, since we assume that the failure process is stationary considering the interval of time during which the epidemics spreading happens (i.e., we assume $N(t) \approx N^*$ with $N^*$ equal to the number of hosts present in the system at the beginning of the epidemics).

In our case the initial condition is $I_0 = 1$: this represents the first copy of the message that is inserted in its buffer by the sender. This result can be used to calculate the number of infectives at instant $t$ with a given infectivity $\beta$ and a given removal rate $\gamma$, or, more interestingly for our purposes, $\beta$ and $\gamma$ can be tuned in order to obtain a certain epidemics spreading, after a specific length of time has passed. The infectivity $\beta$ is the fundamental parameter of the message replication algorithm. In fact, a certain infectivity $\beta$ can be selected in order to obtain, at time $t^*$, a number of infectives (i.e., hosts that have received the message) equal to $I(t^*)$ or, in other words, a percentage of infectives[3] equal to $I(t^*)/N(t^*)$. The parameter $\gamma$ can be interpreted as the deletion rate of the messages from the buffer of the hosts. In fact, since the message buffers have limited size, it may be necessary to delete some messages according to a certain policy. Thus, from the average removal rate of messages from buffer, it is possible to derive the infectivity that is necessary and sufficient to spread the infection. In case the absence of overflow phenomena (i.e., in the case of sufficiently large buffers) can be assumed, the model can be simplified with $\gamma = 0$.

In homogeneous networks, such as random graphs[4], the node degree $k$ for each node can be approximated quite precisely with the average degree of connectivity $\langle k \rangle$ of the network. Therefore, in case of homogeneous networks, in order to take into account the effect of the connectivity, it is possible to rewrite the system (1), substituting $\beta$ with $\lambda \frac{\langle k \rangle}{N}$ as follows, as discussed

[3]Note that $\beta = f(I(t))$ is not defined for $I(t) = N(t)$. Therefore, from a practical point of view, in the case of a message sent to all the hosts of the system, we will use the approximation $I(t) = N(t) - \epsilon$, with $\epsilon > 0$, in the expression used to calculate $\beta$.

[4]The degree distribution of a random graph is a binomial distribution with a peak at $P(\langle k \rangle)$.

in [14]:

$$\begin{cases} \dfrac{dS(t)}{dt} = -\lambda \dfrac{\langle k \rangle}{N} S(t)I(t) + \gamma(t)I(t) \\ \dfrac{dI(t)}{dt} = \lambda \dfrac{\langle k \rangle}{N} S(t)I(t) - \gamma(t)I(t) \\ \dfrac{dN(t)}{dt} = -\phi N(t) \\ S(t) + I(t) = N(t) \end{cases} \tag{3}$$

The first equation states that the number of susceptible hosts is given by the sum of a term proportional to the spreading rate $\lambda$, the number of susceptible nodes that may become infected, $S(t)$, the number of infected individuals in contact with any susceptible node and a term, $\gamma I(t)$, that represents the number of infectives that recover and then become susceptible again, per unit of time. The second equation can be interpreted in a similar way. The third equation models the variation of the number of hosts; the fourth encapsulates the assumption that hosts that did not fail are either infected or susceptible. $\lambda$ represents the probability of infecting a neighbouring host. $\frac{\langle k \rangle}{N}$ gives the probability of being in contact with a certain host. In other words, in this model, by substituting $\beta$ with $\lambda \frac{\langle k \rangle}{N}$, we have separated, in a sense, the event of being connected to a certain host and the infective process [14].

The solution of this system is similar to (2) (i.e., it is sufficient to substitute $\beta$ with $\lambda \frac{\langle k \rangle}{N}$). Thus, it is possible to calculate $\lambda$ as function of $I(t^*)$ and $\langle k \rangle$. Finally, it is interesting to note that in homogeneous networks, every host knows its value of $k$ and, consequently, of $\langle k \rangle$. We will exploit this property to tune the spreading of message replicas in the system.

# 4 The Epidemic Search Algorithm

[31] designed a set of middleware primitives which allow a reliable and tunable probabilistic communication and information dissemination in `mobile ad hoc networks` (MANETs), consisting of about 100 nodes. The goal of our work is the evaluation of this algorithm in a P2P environment with a much higher number of nodes. We therefore design a search algorithm for Gnutella-like networks, which uses epidemic information dissemination techniques to control the dissemination process of queries. In a P2P network such as Gnutella (Section 2.1), messages are sent from source A to its neighbours in a flooding-based manner. To avoid the overhead produced by flooding the message to each neighbour, we use the results from Section 3.1 to only send a message from A to a randomly chosen neighbour B with a certain probability $\Psi$, and still make sure that at time $t^*$ a certain number of hosts have received the message. Thus, the parameter $\Psi$ can be used to control the reliability of the information dissemination mechanism. In other words, given an expected reliability (or percentage of hosts which has to be infected) equal to $\Psi$, it is possible to calculate the value of $\beta$ in order to obtain $I(t^*) = \Psi N$.

## 4.1 The Epidemic Search Algorithm in a Gnutella-like network

As mentioned in Section 2, several approaches have been undertaken to improve search algorithms in P2P networks. Search facilities in eDonkey/eMule rely on central indexing servers. This means potential scalability problems if too many clients try to query the same server. The Gnutella search was traditionally based on flooding search queries over the network, meaning a high overhead. Gnutella's Dynamic Search Protocol and the two-tier topology of modern Gnutella [4, 35, 37] fought that problem. [30] have measured a good performance of the DQP in the search for popular items, but a poor performance in locating rare items. This is due to the fact that the Gnutella search is not an exhaustive search. Its query horizon is restricted by the mes-

sage TTL, so only a small fraction of nodes in a network will be covered by a given query. As a result, there is no guarantee that a query returns matches that actually exist in the network. To overcome this problem, there has been approaches to combine Gnutella search for popular items using a DHT-based search for rare items in a hybrid query processor [39, 12]. DHT-based P2P applications use structured overlay networks to both control the data placement and the overlay topology itself. They can efficiently locate rare files, but at a high cost. Two main disadvantages of DHT can be observed. In a highly dynamic network with high fluctuation rates of nodes leaving and joining the network, a high overhead of replicating the content between the nodes is produced. Also, DHTs ability to implement keyword search, which is crucial for file sharing applications, is weak and still difficult to implement [30, 29]. Random walks also significantly reduce message overhead [38]. Because of the TTL they use, they have the same problems as the Gnutella approach in being not exhaustive. Furthermore, while the DQP dynamically adapts the sent messages on the popularity of the queried file, random walks do not learn anything from their previous success or failures.

We propose an `Epidemic Search Algorithm` based on epidemic information dissemination. The algorithm takes into account the desired number of search results and the popularity of the queried file. We exploit the two-step search process of Gnutella's Dynamic Query Protocol, to dynamically adapt our algorithm to different file popularities. A probe search is triggered in order to get an idea of the file popularity. A second query is designed by evaluating the number of responses from the probe search to generate just enough results to satisfy the users needs (typically 50 to 200 number of results). In case of popular files, where the probe query delivers a sufficient number of results, no second query is sent, the search process stops immediately. However, there is one major difference between Gnutella and our search approach. Instead of using a higher or lower TTL value to adapt to rare or popular files, we use the epidemic `reliability` parameter $\Psi$ to fine-tune the dissemination of the query. The Epidemic Search Algorithm furthermore analyzes network parameters like the number of nodes $N$ and

their connectivity $k$. In case of a poor connected overlay the algorithm is able to adapt the dissemination process to still make sure a sufficient number of nodes receive the query. A simple proportional equation (4) is used to evaluate the probe query and to choose a sufficient value of the reliability parameter for the second query:

$$\frac{Nr_{probe}}{Nr_{desired}} = \frac{P_{probe}}{P_{desired}} \tag{4}$$

$Nr_{probe}$ is the number of results that are returned from the probe query, and $Nr_{desired}$ is the number of results the user wants to receive. $P_{probe}$ is the reliability parameter for the probe query (i.e., the percentage of hosts that are reached by the probe query). $P_{desired}$ is the percentage of hosts that need to be reached and can be calculated out of the others to set the reliability for the standard query.

It is important for the algorithm to know in advance how popular search results will be, to adjust the parameters of the probe query precisely. If the probe query is sent using a too high reliability value and the queried file is popular, too many results are allready returned by the probe query, wasting bandwith and producing overhead. If the queried file is rare and the probe query is sent to an insufficient number of hosts, the resulting calculation of the reliability parameter is not reliable enough and statistically not valid. The following requirements are needed for the algorithm to work:

- The algorithm has global knowledge over the network topology. This includes information about the number of nodes in the network and their degree of connectivity.
- The algorithm has information about the rough popularity of keyword combinations to choose an appropriate starting value for the reliability parameter of the probe query.

In the Future Work section we describe, how this global knowledge can be achieved. We describe a scheme that uses a query results size threshold

to decide if a keyword combination is popular and a lower probe reliability parameter is sufficient. Finally, a periodical crawl of the network can gather needed properties of the network. In Section 5 we describe the architecture of the simulation framework and the implementation of our algorithms.

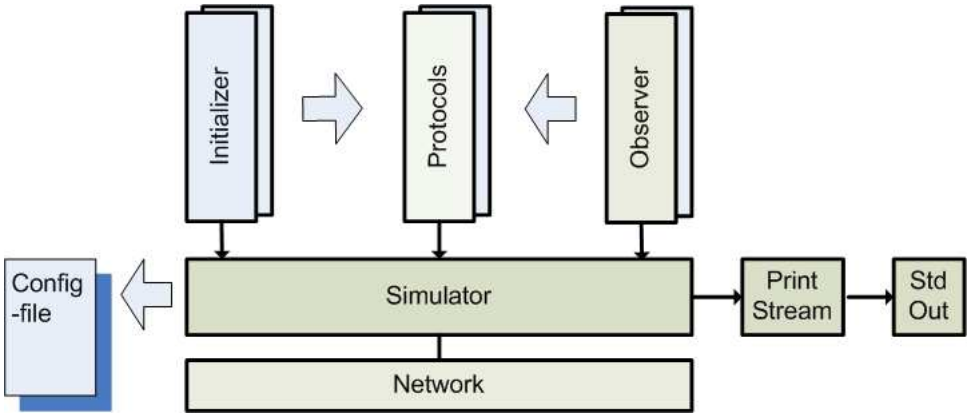



Figure 3: The component architecture of PeerSim

# 5 Architecture and Implementation

To implement the algorithms described in Section 4.1 and produce simulation results, we need a highly configurable framework that can easily be adapted to different needs. For instance, it should be possible to change the underlying network structure, or to exchange one algorithm with another. PeerSim [25, 26] is a Java framework designed to experiment with large scale P2P overlay networks. Its simulator architecture is highly scalable (i.e., it is able to support a high number of network nodes), and through its component structure fully configurable. This makes it easy to join together different pluggable building blocks. The simulator is based on the concept of `cycles`: in each cycle a protocol defined function is executed, in which a node of the network can communicate with its neighbours to perform a specific task (i.e., exchange messages).

Figure 3 shows the rough architecture of the simulation framework and our system. The configuration of PeerSim is done via a config file, in which the developer can set the number of nodes of the network, the number of cycles that make up one experiment, and the number of experiments a simulation shall consist of. The developer can set one or several `protocol` components. A protocol represents a node of the network, with a message buffer

to store query messages and the actual search algorithm which defines how the messages are forwarded to the node's neighbours. We use a separate protocol component for each search algorithm. The developer specifies one or more `initializer controls` which can be used to initialize protocols, including setting up the topology by connecting the nodes of the network. One or more `observer controls` are used to collect data about the current simulation state, like the number of sent messages, or the number of nodes that have received a message. To collect information from several experiments, the output of the observers can be sent to customized `PrintStream` objects, who parse the information and write it to textfiles or to the standard output.

## 5.1 From the Model to the Code

PeerSim nodes communicate by sending messages to each other. The architecture of the messages is shown in Figure 4. The messages are composed of a header, containing information for routing the message, and a message body, containing the actual data. The message header contains the sender, a unique identifier and a flag payload which characterizes the type of the message. Each message also contains an expiration time field, used to specify its validity. Every node can store a finite number of messages, which are inserted into the buffer of a node only if not already present. Additional fields are used to store parameters of the mathematical model, which is used for the replication mechanisms of the messages.

To implement both the Gnutella Search and the Epidemic Search, we adapt the Dynamic Query Protocol implementation of PHEX [8], a Gnutella application written in JAVA, into PeerSim. For Gnutella, we use the PHEX message implementation. Instead of using a TTL field as in the Gnutella message implementation to control the validity of the message, we can use the expiration time and delete the epidemic message at time $t^*$. We use a PeerSim protocol component to maintain a message buffer and control the dissemination of the messages in the buffer in each cycle $t$, until cycle $t^*$ is

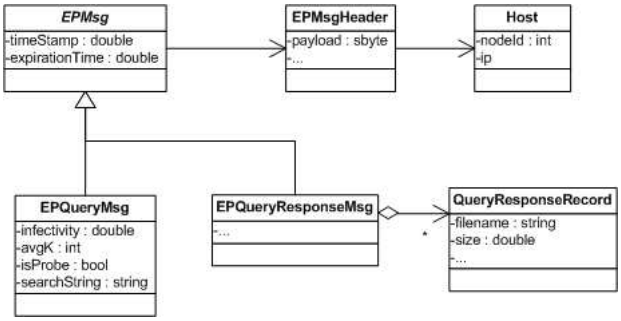



Figure 4: The epidemic message architecture

reached and the message dies. Source code 1 shows the message replication algorithm, which is executed periodically in each node. The algorithm runs through every message stored in the node's buffer and checks its expiration time. If the message has expired, the node removes the message and jumps to the next message in the buffer. For every valid message a random number between 0 and 1 is generated. The algorithm runs through every neighbour of the node and compares the generated random number with the message infectivity. The message is sent to the neighbour, if its random number is less or equal the infectivity of the message.

For the generation of the random numbers we use the library Colt v1.2.0 [16], an open source library for high performance scientific and technical computing in Java. Each node of the network obtains its own instance of the random number generator object. We use the `cern.jet.random.Uniform` implementation that generates statistically uniformly distributed random numbers. We encountered some difficulties using the built in Java Random class that is used in PeerSim. It seems that this implementation does not produce true uniformly distributed random numbers, which is crucial for the Epidemic Algorithm to work.

Each time a node wants to send a new message over the network, the system must calculate the infectivity $\lambda$ which is necessary and sufficient to spread the information with the desired reliability (that has to be chosen

```
// go through all messages in the buffer
foreach(msg in buffer) {
    if (msg.getExpirationTime() < CommonState.getTime()) {
        // time is up
        removeFromBuffer(msg);
        continue;
    }
    // should be between 0..1
    double inf = msg.getInfectivity();
    // go through all reachable nodes
    foreach(node in nodes) {
        // rValue, infectivity between 0..1
        double rValue = generateRandomNumber();
        // models the probability lambda
        if (rValue <= inf) {
            sendMsg(node,msg.clone());
        }
    }
}
```

**Source code 1:** Epidemic Spreading Algorithm.

in the range [0,1]), in a specified time interval, evaluating the average degree of connectivity and the number of nodes of the network. The message's unique identifier, the value of the calculated infectivity, the expiration time that indicates when the message needs to get deleted from the network are inserted into the header of the message and the message is stored in the node which created the message. Source code 2 shows the code used in a PeerSim init control component which creates a new message and stores all needed parameter in it.

In the simulation we create one message and inject it in a randomly chosen node of the network. The average k = avgK = $\langle k \rangle$ can be found by running through all nodes of the network and evaluating the number of connections each node has. It is also possible to use the parameter given in the config file that is used to construct the underlying network topology. The code for calculating $\beta$ and $\lambda$ (the infectivity) is given in Source code 3 and 4.

```
int avgK = System.getAvgK();
int n = Network.size();
double t = System.getMsgTimeValid();
double beta = calculateBeta(reliability, n, t);
double inf = calculateInfectivity(beta, avgK, n);
msg = new Message();
msg.setExpirationTime(System.getCurrentTime+t);
msg.setInfectivity(inf);
msg.setContent(messageContent);

System.addToBuffer(startNode,msg);
```

**Source code 2:** Creation of the Message.

To calculate $\beta$, the algorithm needs to know the reliability ($\Psi$), number of nodes in the network, and the number of rounds (i.e., the time period the message is valid), set in the config file by the developer. We use an adapted binary search algorithm (Source code 3) to calculate $\beta$. We calculate the number of hosts we need to infect and try different values for $I(t)$, until $I(t) == hostsToInfect$. The calculation of $\lambda$ is shown in Source code 4. The algorithm multiplies $\beta$ with the number of nodes in the network and divides through $\langle k \rangle$.

In contrast to [31], we do not consider restricted buffers and therefore we do not need to take into account the average removal rate $\gamma$. We assume that due to more available ressources such as memory capacity in P2P applications than in devices for MANETs, no removal of messages during the life time of a message is needed. In the simulation we are more interested in the behavior of one query at a time. However, in a real application it is easy to implement the $\gamma$ parameter, if the application developer has to cope with restricted resources that makes deletion of messages from the buffer inevitable.

The cycle of the search algorithm is the same for Gnutella and Epidemic Search. A method `processQuery()` as shown in source code 5 is invoked

```
private double calculateBeta(double reliability,
    int networkSize, double nrRounds) {
    double nrHostsToInfect = networkSize * reliability;
    double coeff = (reliability==1.0) ?
        ((99.9999 / 100) * nrHostsToInfect) :
        nrHostsToInfect;
    double low = 0.0;
    double high = 1;
    //adapted binary search
    while ((int)i != (int)coeff) {
        beta = (high + low)/2;
        double b = Math.exp(nrRounds * networkSize * beta);
        i = b / (double)(1 + ((b - 1)
            / (double)networkSize) );
        if(i >= coeff) {
            high = beta;
        }else {
            low = beta;
        }
     }
     return beta;
}
```

**Source code 3:** Calculation of $\beta$.

periodically and, depending on the state of the query process, calls either `processProbeQuery()` (Source code 6) or `processStandardQuery()` (Source code 7). The standard query is only invoked if the number of results of the probe query is not sufficient.

If a probe query has not yet been sent, `processProbeQuery()` as shown in Source code 6 is invoked. It sends a probe message with a default time the message is valid[5], and a value for the reliability (either set by the developer or calculated by the schemes described in Section 4.1). It then sets a flag indicating that the probe query was sent and a waiting time that is needed

[5]We empirically found that $t = 100$ rounds are statistically sufficient to ensure the dissemination process. However, a different value could be needed for different networksizes and connectivities.

```
private double calculateInfectivity(double beta,
    int avgK, int networkSize) {
        return (beta * networkSize) / (double) avgK;
}
```

**Source code 4:** Calculation of $\lambda$.

```
public void processQuery() {
    int currentTime = CDState.getTime();
    if (currentTime < nextProcessTime
        || isQueryFinished()) {
        // not our turn to query now...
    return;
    }
    if (!isDynamicQueryStarted) {
        isDynamicQueryStarted = true;
    }
    if (!isProbeQuerySent) {
        processProbeQuery();
    } else if(!isQueryFinished()) {
        processStandardQuery();
    }
}
```

**Source code 5:** processQuery.

until the results are returned to the searching node.

The method `processProbeQuery()` in Source code 7 evaluates the search results from the probe query by calling `calculateReliability()`. It sends the query message with the same time validity as `processProbeQuery()`. The method `sendMessage()` constructs a new search query similar to that shown in Source code 2, calculates all needed parameters and adds the message to the sender node's buffer. After sending the standard query, the search process stops by setting the `isDynamicQueryStopped` flag to true.

Source code 8 shows the calculation of the reliability parameter needed for the standard query which is derived from (4). It multiplies the reliability

```
private void processProbeQuery() {
    double reliability = DEFAULT_PROBE_RELIABILITY;
    int msgTimeValid = DEFAULT_MSG_TIME_VALID;
    ...
    sendMessage(node, reliability, msgTimeValid);
    nextProcessTime = CDState.getTime() + DEFAULT_MSG_TIME_VALID;
    isProbeQuerySent = true;
}
```

**Source code 6:** processProbeQuery.

```
private void processStandardQuery() {
    double reliability =
        calculateReliability(DEFAULT_PROBE_RELIABILITY);
    int msgTimeValid = DEFAULT_MSG_TIME_VALID;
    ...
    sendMessage(node, reliability, msgTimeValid);
    isDynamicQueryStopped = true;
}
```

**Source code 7:** processStandardQuery.

used in the probe query with the user defined number of desired results, and divides this through the number of results received by the probe query.

```
private double calculateReliability(double percHostsProbe) {
    return rel = percHostsProbe * DEFAULT_DESIRED_RESULTS
        / (double) nrProbeResults;
}
```

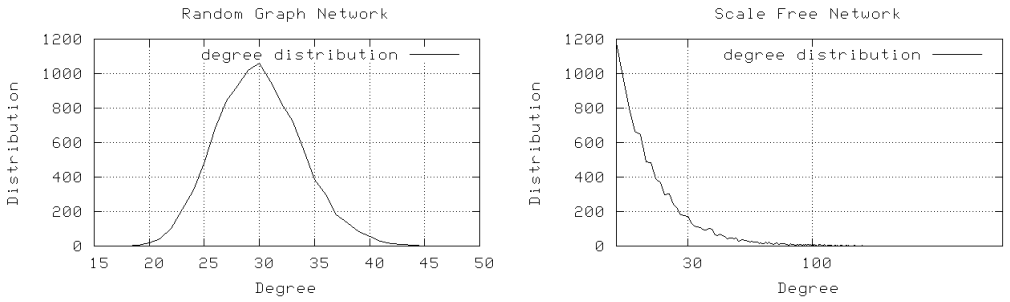
**Source code 8:** calculateReliability.

# 6 Evaluation

In order to evaluate the epidemic algorithm those characterizing properties were analyzed in [31], we present simulation results gathered using PeerSim. In Section 6.1 we evaluate the proposed system and model it using a Random Graph network topology, while later in that section we present results using a Scale Free network topology. We also show that the Epidemic Algorithm can be applied as an effective search algorithm which can compete against Gnutella's search engine. We compare the Epidemic Algorithm as a search algorithm in a Gnutella-like network with Gnutella's Dynamic Query Protocol. For a reference, we also show a comparison between the Epidemic Search and Flood Search, a flooding-based algorithm that was used in the early Gnutella implementations.

We choose a Random Graph network as the basic overlay network for our experiments with Gnutella. As described in Section 2.1, only ultrapeers are responsible for searching the Gnutella network. Therefore, we only consider the ultrapeer overlay in our simulations. [37, 36] have found that this ultrapeer-layer forms a stable core-layer, whose nodes are connected randomly. It also exhibits small-world properties [41], meaning that to reach a random node A from another node B only a few hops are needed. This top-level overlay forms a stable core, with a spike in its connectivity degree distribution around 30. This means, we can approximate the Gnutella topology through a Random Graph with an average degree of connectivity $\langle k \rangle = 30$. We give some evaluation results also for Scale Free networks, to continue and support the analytical work of [31]. We can also use these results to show the universal applicability of the Epidemic Algorithm.

**Analysis of PeerSim's Topology Generators** We evaluate the distribution of the number of links of the Random Graph and Scale Free network (Barabasi-Albert (BA), [10]) implementation of PeerSim. Figure 5.a) shows the link degree distribution of the Random Graph generator used in PeerSim.




(a) Random Graph Topology

(b) Scale Free Topology

Figure 5: The distribution of number of links of the network topology in PeerSim.

We used a network size of 10000 nodes and a connectivity parameter $k = 30$. The graph clearly shows the expected bell-shaped curve of the distribution of node linkages of random graph networks. A peak around 30 can be observed. Figure 5.b) shows the power law distribution of node linkages of the BA model. We also use a $k = 30$, which in this case is the number of edges added to each new node. It is interesting to notice, how steep the curve is. The steepness of the curve implies that in a sufficient big network the number of nodes with a medium or high connectivity is quite high. We empirically show that in this implementation of Barabasi-Albert networks our epidemic model also works for Scale Free networks.

## 6.1 The Plain Epidemic Algorithm

The emphasis of the evaluation of the epidemic algorithm in this work is to show that the algorithm can universally be applied using different network topologies with varying parameters and properties. We therefore use a system consisting of 100 to 10000 nodes, being connected using two different topologies with different degrees of connectivity.

PeerSim uses a cycle based model. We decided not to use the physical

time from the original work [31], but to use logical time (*trounds* ) instead. Apparently it is easy to convert one time model into the other. In order to gain statistically valid data we ran 20 experiments of each simulation and calculated the average of the different trials. Each experiment lasts for 100 rounds ($\tau = 100$). Clearly, the choice of the values of $\tau$ influences the accuracy of the model, since it relies on a probabilistic process. Our simulation showed that 100 rounds are sufficient to achieve accurate results. Developers could ensure the accuracy of the model by checking $\tau << t_{MIN}$, with $t_{MIN}$ being the minimum value of the timestamp of the message. For the Law of the Large Numbers, we obtain a better accuracy of the simulation as the number of rounds (i.e., from a probabilistic point of view, the number of trials) increases. We assume that in the infection phase the number of hosts remains constant and the buffers are well-dimensioned (i.e., big enough for all messages to be stored without the need to remove valid messages), so we can consider the removal rate $\gamma = 0$ and simplify the calculation of $\beta$ as shown in source code 3.

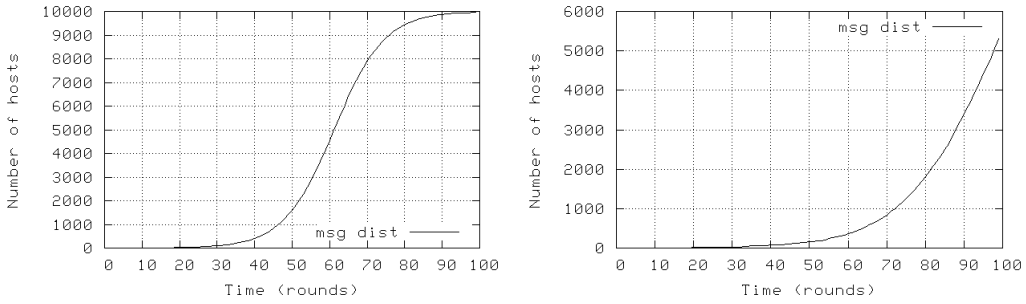
### 6.1.1 Random Networks

In this Section we analyze the results of our simulations, run on an exponential network (Random graph). We study the influence of the parameters `number of hosts` and `average connectivity` $\langle k \rangle$) on the following performance indicators:

- The `delivery ratio`: This is the $hosts_{infected}/network_{size} * 100$ that is is the number of hosts that have received a message divided by the number of hosts in the network. It is desirable that the value is close to the set `reliability`.
- The `number of messages`: This is the needed number of messages that are sent over the network after a period of time $t^*$.

Figure 6 shows the epidemic spreading process (i.e., the number of infectives $I(t^*)$ in a network consisting of 10000 hosts, with a desired reliability of 100 % and 50 %, in a time period of $t^* = 100$. The shape of the curve

and the time when the infection process starts can vary slightly in different trials. It is also important to notice that we use "global knowledge" on the network to calculate $\langle k \rangle$. In an application with only local knowledge were the approximation $k \approx \langle k \rangle$ can be used, the resulting $\beta$ can vary from the ideal value. For example, if a message is sent by a host that has a degree of connectivity $\overline{k} > \langle k \rangle$ the value of $\beta$ will be lower than the infectivity associated to the average degree of connectivity $\langle k \rangle$.



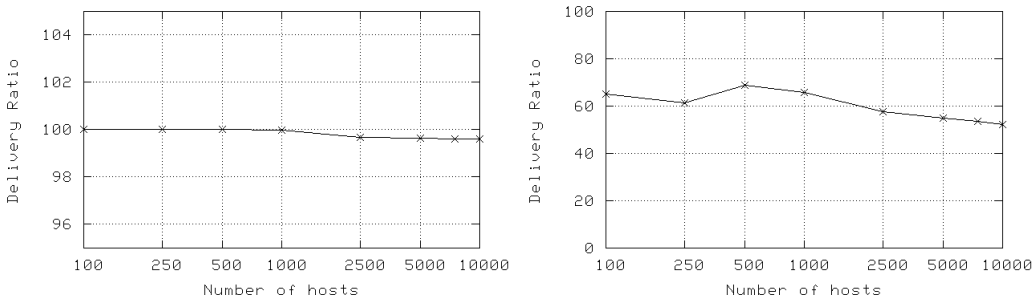

(a) with desired reliability = 100

(b) with desired reliability = 50

Figure 6: Simulation data infection spreading curve in Random Graph network with 10000 hosts, using an infection time of $t^* = 100$ rounds

Figure 7 shows the delivery ratio on a system with a high number of hosts, with $\langle k = 20 \rangle$, $t^* = 100$, and a desired reliability equal to 100 and 50, respectively. The number of infected hosts decreases slightly with an increasing network size. It seems that the chosen number of rounds $t^* = 100$ is statistically not always sufficient enough to ensure the infection process in a high-number network. Either the extension of the duration of the message validity, or an increase of $\langle k \rangle$ can solve this problem.

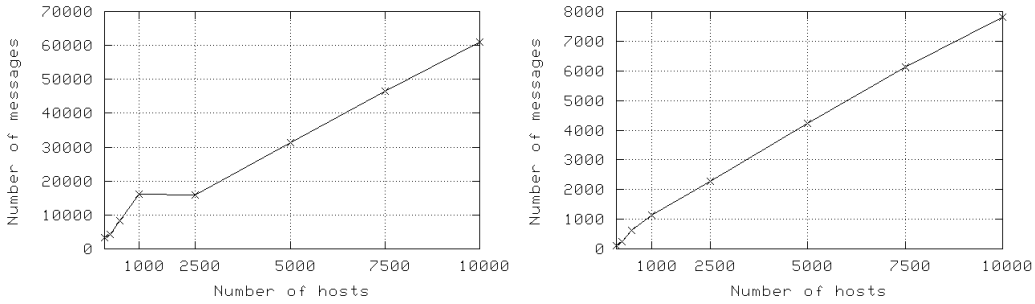
Figure 8 shows the overall number of sent messages as a function of the population density. A number of hosts from 100..10000 nodes is used, $k = 30$, $t^* = 100$, and 20 experiments per sample. It is interesting to see that the curve in both cases of desired reliability is approximately linear. This shows




(a) with desired reliability = 100 (b) with desired reliability = 50

Figure 7: Delivery ratio vs population density with $t^* = 100$ rounds

the scalability of our algorithm.




(a) with desired reliability = 100 (b) with desired reliability = 50

Figure 8: Number of messages vs population density with $t^* = 100$ rounds

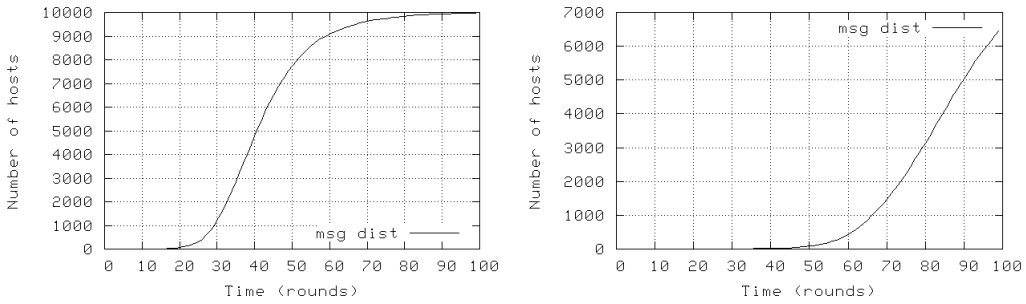
### 6.1.2 Scale Free Networks

Due to the lack of mathematical epidemic models for Scale Free networks we can only give empirical results of our simulations, at the moment. [31] found, that the approximation $k \approx \langle k \rangle$ is not valid for heterogeneous (i.e., Scale Free) networks. They observed that given $k$ fluctuating in the range

$[k_{MIN}, k_{MAX}]$ [6], for a value of the infectivity corresponding to $k = k_{MIN}$, the obtained spreading of the infection $I(t^*, k_{MIN})$ will satisfy the following property:

$$I(t^*, k) > I(t^*, k_{MIN}) \quad \forall k \in ]k_{MIN}, k_{MAX}] \tag{5}$$

In other words, if $k_{MIN}$ is selected in the calculation of the value of the infectivity, the value of `Reliability` can be considered approximately as a guaranteed lower bound of the reliability level. Properties of the network such as the value of $k_{MIN}$, $\langle k \rangle$, and the network size $n$ can be dynamically retrieved and set by crawling the network in periodical intervals.



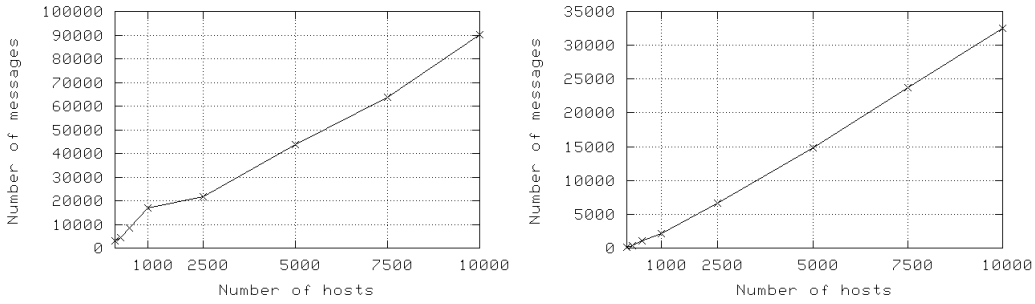

(a) with desired reliability = 99

(b) with desired reliability = 50

Figure 9: Infection spreading curve in BA Networks with $t^* = 100$ rounds, using a $\langle k \rangle = 20$ value

Figure 9 shows the infection spreading curve in Scale Free networks. A network size of 10000 nodes is used, a $k = 30$, and a time period of $t^* = 100$ rounds. Our empirical evaluation has shown that if a $k_{MIN}$ is used in the calculation of the infectivity, the system is overestimating too much and a too high number of nodes is reached. We therefore use $\langle k \rangle$ in Scale Free networks, as well. The curves in Figure 9 show a similar behaviour as in Random Graph networks.

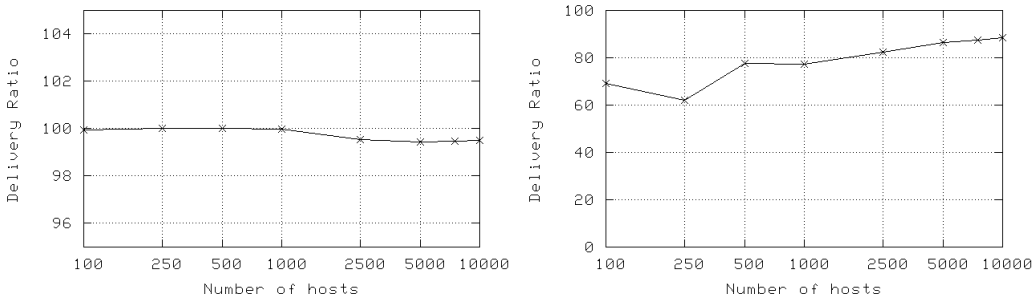
[6] Homogeneous networks can be considered as a particular case of heterogeneous networks with $\langle k \rangle = k_{MIN} = k_{MAX}$




(a) with desired reliability = 100 (b) with desired reliability = 50

Figure 10: Number of messages vs population density in BA networks with time = 100 rounds

Figure 10 shows the number of sent messages as a function of the number of hosts in a Scale Free network. A number of hosts from 100..10000 nodes is used, $k = 30$, $t^* = 100$, and 20 experiments per sample. The curves look very similar to Figure 8. As in Random Graphs, it seems that the number of messages increases almost linear with the number of hosts.




(a) with desired reliability = 100 (b) with desired reliability = 50

Figure 11: Delivery ratio vs population density in BA networks with $t^* = 100$ rounds

Figure 11 shows the delivery ratio for the respective number of messages

shown in Figure 10. As for a desired reliability = 100, the delivery ratio for all numbers of hosts is very close to 100. For a desired reliability = 50 the algorithms seems to overestimate a significant amount. The delivery ratio is around 60 and even peaks at 70 for a number of hosts = 500.

## 6.2 The Epidemic Algorithm as a Search Algorithm

With the introduction of the two-tier topology and the development of the DQP, Gnutella managed to overcome its scalability problems and has increased its user base in 2006 to over 2 million users [9]. However, recent research [30, 29] still tries to improve certain aspects such as the search for rare items in Gnutella-like P2P networks. The proposed DHT-based hybrid search algorithms suffer from high content publishing overhead as described in Section 4.1, and, due to its hybrid network topology, is not as simple and elegant from a software engineering point of view, as algorithms based on simple unstructured networks. To compare our `Epidemic Search` with the Gnutella Search, we consider three parameters which we think are fundamental for an efficient, reliable and slim search algorithm:

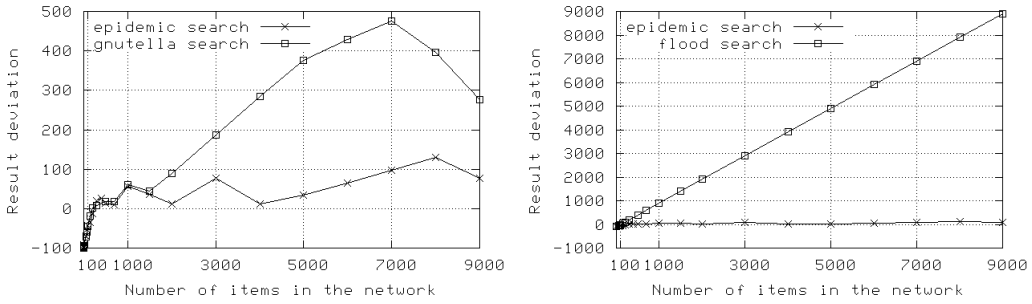
- `Deviation from desired to received results`: This is the error of the number of results the user wants to the actually number of received results. The lower this deviation is, the more economical the algorithm is in terms of overhead.
- `Sent messages`: This is the number of overall sent messages in the network. The less messages a query needs to successfully returns the number of desired results, the more efficient an algorithm is.
- `Sent/Received ratio`: This is the number of sent messages, divided through the number of received messages. A low sent / received ratio means a highly efficient query.

Each of these parameters are evaluated in a network with 10000 nodes, using a $k = 30$, and a time period of $t^* = 100$ rounds. Each of the parameters is printed as a function of `Number of items` in the network that is the

file popularity. Apparently, the overhead produced by a high deviation of desired results and the resulting number of sent messages is far higher for popular files. On the other hand, a low sent/received ratio is only meaningful in combination with a low deviation of desired results. For a user it is more important to receive the desired number of results, so it is acceptable for a query to need more query messages for rare files. The potential in saving messages lies in finding popular files efficiently, without producing high overhead.

We compare the Epidemic Search with the DQP of Gnutella, considering a Random Graph overlay network consisting of ultrapeer nodes. In a first step, we inject a pseudo file containing a search string into a certain percentage of nodes of the network. The nodes are chosen in a random manner. In the search process, every node runs through the message buffer and as soon as there is a query message in the buffer of a node and the node possesses the queried file it responds to the search initiator with a query hit message. The system counts the overall number of sent messages, the number of query hit messages and calculates the deviation from the desired number of results. Each simulation for each sample consists of 50 experiments, each experiment takes 200 rounds (100 rounds for the probe and the standard search phase, respectively). As a reference, we also include simulation results of Flood Search, a TTL-constrained search algorithm similar to the one used in early Gnutella implementations. We use a $TTL = 7$, which is the maximum value the TTL field is allowed to have [22].

Figure 12.a shows the deviation from the number of desired results a user wants (in our case 100 results). With increasing file popularity the Epidemic Search is far closer to the number of desired results than Gnutella, meaning that Gnutella still produces a considerable overhead when searching for popular files. The efficiency of Epidemic Search becomes more apparent when compared to Flood Search in Figure 12.b: While the deviation of desired results appears to stay almost constant in Epidemic Search, Flood Search produces a linear increase of the result deviation. This clearly shows the advantage of the dynamic two-step process of Epidemic Search and DQP. For




(a) Gnutella vs. Epidemic Search

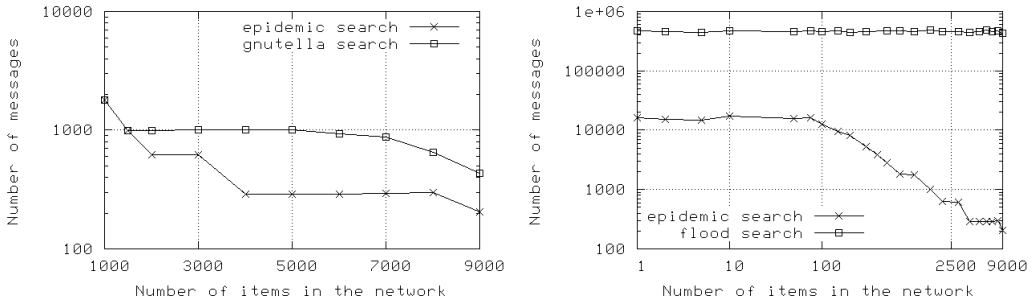
(b) Flood Search vs. Epidemic Search

Figure 12: Comparison of the deviation from desired results (100) in Gnutella, Flood Search and Epidemic Search, with a variable number of files (items to search) in the network.

popular files, the flooding-based algorithm produces far more results a user is ever likely to use.

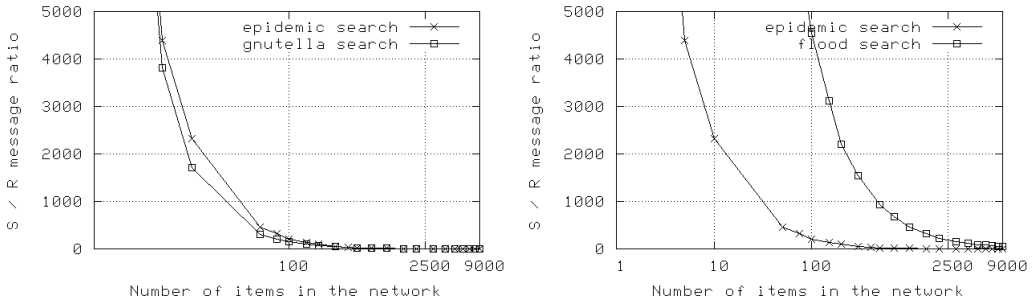
Figure 13.a shows the overall number of sent messages of Epidemic Search and Gnutella using the same simulation settings as in Figure 12. For popular items again, Epidemic Search shows its supremacy. It needs significantly less messages than Gnutella. The magnitude of savings of messages is even more impressive in Figure 13.b, when compared to the sent messages of Flood Search.

Figure 14.a shows the sent/received ratio of Gnutella and Epidemic Search. The performance of both of the algorithms are very similar for popular files. Epidemic Search exhibits a slightly higher sent/received ratio for rare files. This is a wanted behavior, because the user is interested in actually retrieving search results, if there are files in the network that match the user's query. It is therefore acceptable to send more search messages than Gnutella sends, if the results are better and more search results are produced. This behavior is confirmed by Figure 15. It compares the number of received search results produced by the Epidemic Search, Gnutella Search, and the ideal (i.e.,




(a) Gnutella vs. Epidemic Search (b) Flood Search vs. Epidemic Search

Figure 13: Comparison of the sent messages in Gnutella, Flood Search and Epidemic Search, with a variable number of files (items to search) in the network.




(a) Gnutella vs. Epidemic Search (b) Flood Search vs. Epidemic Search

Figure 14: The sent/received message ratio in Gnutella, Flood Search and Epidemic Search, with a variable number of files (items to search) in the network.

the maximum) number of search results. In this simulation we increase the file distribution from 1 to 500 file replicas, in a system consisting of 10000 nodes. We use a probe reliability value of 99% for the Epidemic Search. The obtained number of results are very close to the ideal in the case of the Epidemic Search. The curve of the Gnutella Search confirms the results of [29], who found that Gnutella fails to deliver reliable search results for rare items.

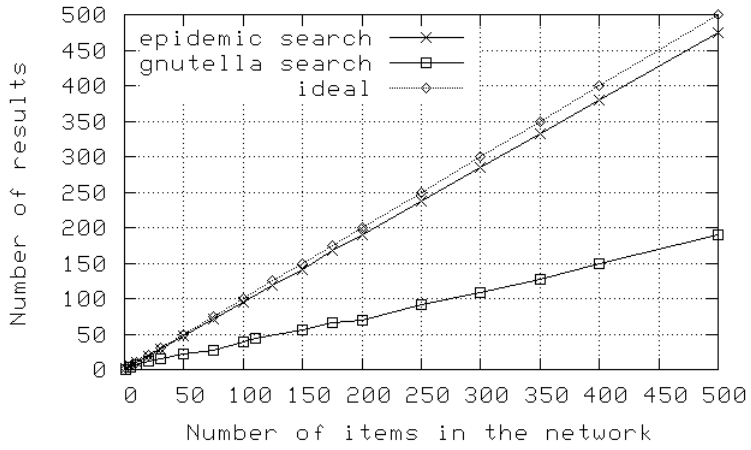



Figure 15: Comparison of the number of search results of Epidemic Search, Gnutella Search and the ideal (max.) number of search results, in a system with a low file distribution [1,500], 20 experiments, nodes = 10000.

Figure 16 and Figure 17 show the number of sent messages and received results as functions of the `probe reliability`. As a basis for these results we lay the following assumptions and observations:

- The chosen value for the probe reliability parameter is vital for the efficiency of the Epidemic Search algorithm.
- There is no optimal value of the probe reliability parameter that is adequate for all kind of file popularities. A rare file needs a different value for this parameter than a popular file.

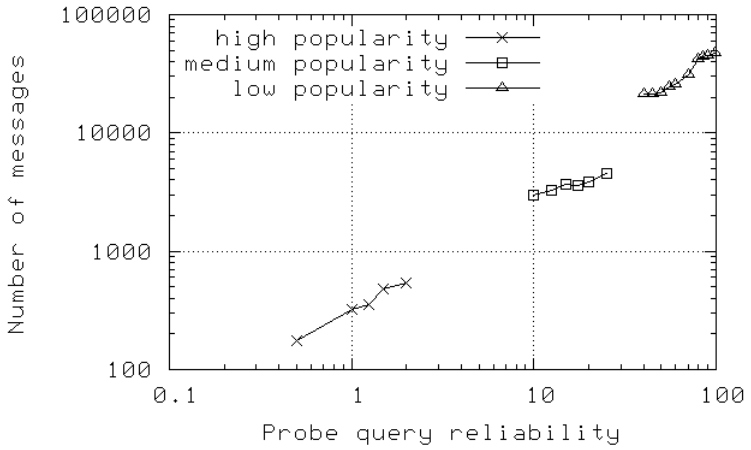



Figure 16: Number of sent messages in a system with low (50 files), medium (200 files) and high (2000 files) file distribution. Desired number of results = 50, nodes = 10000.

- We therefore need a scheme to decide how popular a file is, to appropriately choose the probe reliability parameter. We have described this scheme in Section 4.1.

- We assume that in a real application a similar scheme is used to statistically evaluate the popularity of a certain file, so an appropriate probe reliability value can be chosen.

For the simulation we empirically set the probe reliability parameter. We choose a low value for popular files, to minimize the overhead produced by a query that reaches a too high number of hosts and by that returns a higher number of results than needed. We choose a high probe reliability value for rare files, to cover a high percentage of nodes for the probe query in order to collect a statistically valid sample to reliably locate the few number of replicas of rare files in the network. Figure 16 shows the overall number of sent messages needed to receive the corresponding number of results in Figure 17, in a 10000 host scenario, with a desired number of results equals 50, in a Random Graph topology with $\langle k = 30 \rangle$, a message time validity of $t^* = 100$ rounds and a number of trials per sample of 20.

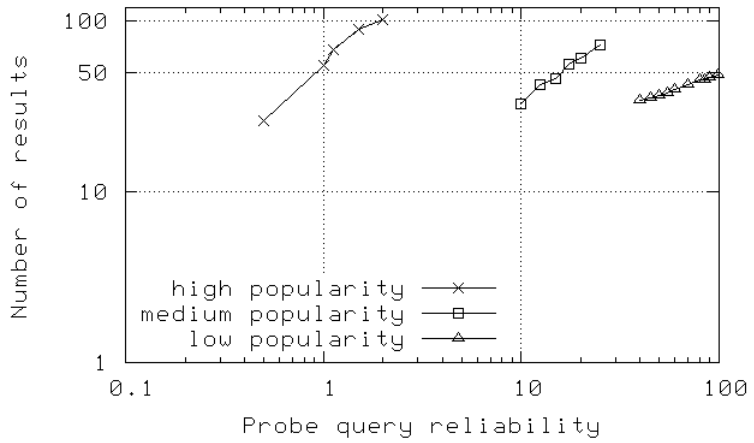



Figure 17: Number of results in a system with low (50 files), medium (200 files) and high (2000 files) file distribution. Desired number of results = 50, nodes = 10000.

Figure 17 shows that for rare files (low popularity) a probe reliability equals 99 percent is needed to locate the desired number of results. It is important to note that the desired number of results in this scenario is equal to the number of files in the network. The 48.4 number of average results over 20 trials, using a probe reliability value of 99 means an exhaustive search result. Figure 17 also shows a needed probe reliability value of 17.5 percent for a medium file popularity and a reliability value of 1 percent for a high popularity. The number of messages needed for these results, as shown in Figure 16, are 47805.45, 3594.45 and 321.15 for a low, medium and high file popularity, respectively.

The results show that the Epidemic Search algorithm performs well and is able to produce better results than even the highly optimized Dynamic Query algorithm of Gnutella. While the Gnutella search algorithm is optimized for the specific topology characteristic of the current Gnutella network, our Epidemic Search algorithm is not restricted to any specific network characteristic. It certainly works best in a highly connected Random Graph

network, but due to its dynamic adaptation mechanisms it is also able to handle less connected graphs and graphs with a very high or very low number of nodes. The results also suggest that the algorithm performs well in heterogeneous networks, although there needs to be done more theoretical and analytical work, until we can say for sure that our algorithm in its current form universally works in homogeneous and heterogeneous networks.

# 7 Conclusion and Future Work

In this work we have introduced an algorithm for epidemic-style information dissemination, that relies on a probabilistic process for spreading information. The mathematical model is founded on the work of [31], who use complex network theory to fine-tune the epidemic information dissemination in MANETs. We have characterized search algorithms in current P2P networks and have outlined their assets and drawbacks. We have characterized homogeneous and heterogeneous networks and have given simulation results of the Epidemic algorithm for both of them. We have implemented this algorithm in the more wide-scale environment of P2P networks, using PeerSim, a Java-based peer-two-peer simulator. Being inspired by the two-step process of Gnutella's Dynamic Query Protocol we have designed a search algorithm, based on the probabilistic Epidemic algorithm. We have compared this algorithm with Gnutella's approach and with a flooding based search algorithm. We also have given a characteristic of the network topology of Gnutella-like P2P networks. As an evaluation, simulation results have been given for all three search algorithms.

We have shown the superiority of the Epidemic Search algorithm over Gnutella's Dynamic Query Protocol. Due to the probabilistic model the algorithm is based on, certain knowledge of the underlying network topology is needed. Furthermore, if applied as a search algorithm in P2P networks, the algorithm needs to know about the possible file popularity. This is global knowledge a node can only retrieve by additional crawling techniques to gather the required information. It is important to point out that the Epidemic Search algorithm only works well if knowledge about network topology characteristics and file popularity is available. It may fail in reliably locating rare files, if an insufficient probe reliability value is used or may produce unwanted high overhead if a high probe reliability value is used for locating popular files. [29] developed a bundle of schemes for discovering rare items which could be published in their DHT-based search. A very simple scheme is the `Query Results Size`. A parameter Results Size Threshold is used to

decide whether a file is popular or not. If the number of results is below this threshold, it gets published in the DHT. Because this scheme is used to discover rare items, it suffers from the fact that many rare items may not have been previously queried and found. We can exploit this simple scheme for discovering popular items. If a query creates a number of results which are above the results size threshold, a file is considered popular and a low probe reliability value can be used for the probe query. This results in a low percentage of hosts to reach that is sufficient to return the desired number of search results and at the same time uses a low number of overall sent query messages.

Further work needs to be done to better understand the characteristics of heterogeneous networks and how they affect the behavior of the Epidemic algorithm. Our empirical simulation results suggest that relaxing the assumption of homogeneous networks by using a $k_{MIN}$ instead of a $\langle k \rangle$ might result in a massive overestimation of the system and a flooding behavior. We have found that using $\langle k \rangle$ in the calculation of the infectivity value gives the better results. We do not fully understand by now, if this is due to the topology characteristic of Scale-free networks in general, or because of the specific implementation of PeerSim's Barabasi-Albert (BA) model.

The simulation results for the Epidemic Search algorithm encourage us to apply it in a running Gnutella-like P2P application, in the future. Therefore, an unstructured P2P protocol similar to Gnutella, but with a timestamp instead of a TTL field as an message expiry value, needs to be designed. Crawling techniques for gathering network topology information and information about file popularity need to be implemented. A reliable, highly scalable and fast working search algorithm is not enough to make up a popular, user accepted P2P file-sharing application. Modern P2P applications additionally consists of an easy-to-use user interface, multi-source download, possible compression technologies to save bandwidth, integrated chat, encryption, file metadata look-up and so forth.

It would also be interesting to further investigate possible other areas of application of the Epidemic Search algorithm. It should be possible to successfully apply it wherever a simple to use and simple to apply search algorithm is needed in network applications, provided there is exact or approximated knowledge of the network topology available.

# References


[1] *Mathematical Modeling in Epidemiology*, chapter 1, pages 1–11. Springer-Verlag Berlin Heidelberg New York, 1980.

[2] eMule Project Network Guide, 2004. http://www.emule-project.net.

[3] Gnutella Development Forum, March 2006. http://www.the-gdf.org/wiki/.

[4] Gnutella Dynamic Query Protocol v.0.6, March 2006. http://www.the-gdf.org.

[5] HTTP - Hypertext Transfer Protocol, March 2006. http://www.w3.org/Protocols/.

[6] P2P Telephony Explained ? For Geeks Only, March 2006. http://www.skype.com/products/explained.html.

[7] Peer-to-peer, March 2006. http://en.wikipedia.org/wiki/Peer_to_peer.

[8] PHEX Gnutella Client, March 2006. http://phex.kouk.de.

[9] Slyck - File Sharing News and Info, March 2006. http://www.slyck.com.

[10] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47, 2002.

[11] Roy M. Anderson and Robert M. May. *Infectious Diseases of Humans: Dynamics and Control*. Oxford University Press, 1992.

[12] Sebastien Baehni, Chirdeep Chabra, and Rachid Guerraoui. Frugal Event Dissemination in a Mobile Environment. In *Proceedings of ACM Middleware'05*, 2005.

[13] Albert-Laszlo Barabasi and Eric Bonabeau. Scale-Free Networks. *Scientific American*, May 2003.

[14] Marc Barthélemy, Alain Barrat, Romualdo Pastor-Satorras, and Alessandro Vespignani. Dynamic Patterns of Epidemic Outbreaks in Complex Heterogeneous Networks. *Journal of Theoretical Biology*, 2005.

[15] Bela Bollobas. *Random Graphs*. Cambridge University Press, Second edition, 2001.

[16] Cern. Colt - High Performance Scientific and Technical Computing in Java, 2004. http://dsd.lbl.gov/ hoschek/colt/.

[17] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable, 2003.

[18] Thomas M. Chen and Jean-Marc Robert. Worm Epidemics in High-Speed Networks. *IEEE Computer*, pages 48–53, June 2004.

[19] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. pages 81–86, 2001.

[20] Alam Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic Algorithms for Replicated Database Maintenance. *ACM SIGOPS Operating Systems Review*, 22(1), January 1988.

[21] Patrick T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massouli. Epidemic Information Dissemination in Distributed Systems. *IEEE Computer*, May 2004.

[22] A. Fisk. Gnutella Dynamic Query Protocol v.0.1, 2003.

[23] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random Walks in Peer-to-Peer Networks. *IEEE Infocom*, 2004.

[24] Oliver Heckmann and Axel Bock. The eDonkey 2000 Protocol. Technical Report KOM-TR-08-2002, Multimedia Communications Lab, Darmstadt University of Technology, December 2002.

[25] Gian Paolo Jesi. PeerSim HowTo: Build a new Protocol for the PeerSim 1.0 Simulator, December 2005.

[26] Gian Paolo Jesi. PeerSim HowTo: Build a Topology Generator for the PeerSim 1.0 Simulator, December 2005.

[27] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[28] Yoram Kulbak and Danny Bickson. The eMule Protocol Specification, 2005.

[29] Boon Thau Loo, Joseph M. Hellerstein, Ryan Huebsch, Scott Shenker, and Ion Stoica. Enhancing P2P File-Sharing with an Internet-Scale Query Processor, 2004.

[30] Boon Thau Loo, Ryan Huebsch, Ion Stoica, and Joseph M. Hellerstein. The Case for a Hybrid P2P Search Infrastructure, 2004.

[31] Mirco Musolesi and Cecilia Mascolo. Controlled Epidemic-style Data Dissemination in Mobile Ad Hoc Networks. Technical report, Department of Computer Science, University College London, March 2005.

[32] J. Postel. User Datagram Protocol, 1980. http://www.faqs.org/rfcs/rfc1320.html.

[33] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network. Technical report, Berkeley, CA, 2000.

[34] R. Rivest. The MD4 Message-Digest Algorithm, 1992. http://www.faqs.org/rfcs/rfc1320.html.

[35] Daniel Stutzbach and Reza Rejaie. Characterizing the Two-Tier Gnutella Topology. *Sigmetrics'05*, June 2005.

[36] Daniel Stutzbach, Reza Rejaie, and Amir H. Rasti. On the long-term evolution of the gnutella network. Technical report, University of Oregon, 2005.

[37] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. Characterizing unstructured overlay topologies in modern P2P file-sharing systems. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, October 2005.

[38] Dimitrios Tsoumakos and Nick Roussopoulos. A Comparison of Peer-to-Peer Search Methods. *WebDB*, 2003.

[39] Amin Vahdat and David Becker. Epidemic routing for Partially Connected Ad Hoc Networks. Technical Report CS-2000-06, Dept. of Computer Science, Duke University, 2000.

[40] Werner Vogels, Robert van Renesse, and Kenneth P. Birman. The Power of Epidemics: Robust Communication for Large-Scale Distributed Systems. *ACM Computer Communication Review*, 33(1):131–135, January 2003.

[41] Duncan J. Watts. *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton Studies on Complexity. Princeton University Press, 1999.

[42] Lidia Yamamoto. Epidemic dissemination in ad hoc networks. In Matthias Bossardt, Georg Carle, D. Hutchison, Hermann de Meer, and Bernhard Plattner, editors, *Service Management and Self-Organization in IP-based Networks*, number 04411 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2005.

# A Appendices

## A.1 User Manual

This user manual is provided to enable users to run the simulation and to reproduce the simulation results given in this work. The system is implemented in PeerSim, and comes as an Eclipse project. Eclipse can be downloaded from http://www.eclipse.org.

**System Requirements:** This simulation can be run on every system on which PeerSim can be run. Please read the documentation on http://peersim.sourceforge.net/ if you encounter any problems. We used Java 1.5 to run the simulation, because PeerSim 1.0 demands it. Our code is still compatible to Java 1.4.x.

**Running the Simulator:** The PeerSim simulator is run through the command `java peersim.Simulator config-file.txt`, where 'config-file.text' needs to point to an existing configuration file. We provide three configuration files located in the folder `conf`. Source code 9 shows the configuration of the plain Epidemic Algorithm. Every line that starts with an  is a comment and not considered by the simulator. Important parameters to change are *simulation.cycles* and *simulation.experiments* that specify the number of rounds one experiment is running and the number of experiments a simulation consists of. *network.size* specifies how many nodes the network consists of. *init*.0.$k$ constrains the number of links a node has in the network. *init*.1.*reliability* specifies the reliability parameter (i.e., the percentage of hosts the algorithm reaches). The value has to be in the range of [0,1]. *init*.1.$msgTimeValid$ constrains, how many rounds the epidemic message is valid. To follow the conventions PeerSim is using, *System.out* is redirected to a PrintStream. The class that derives from *PrintStream*, can be specified via *simulation.stdout*. This is a convenient way to analyze the System.out stream and write it to a file, for example. Therefore, *System.err* is used to print out the output of the simulation.

Source code 10 shows the configuration file of the Epidemic Search Algorithm. *protocol*.1.$msgTimeValid$ specifies the time the query message is valid, in this case. *protocol*.1.$defaultProbeReliability$ specifies the reliability parameter that is used to determine, how high the percentage of the hosts needs to be, to be reached

by the probe query of the algorithm. $protocol.1.defaultDesiredResults$ specifies, how many results the algorithm aims to deliver. $init.1.perc\_msg\_dist$ defines, how many nodes of the network contain the search string, the algorithm searches for. The value of the parameter specifies a percentage of hosts, and therefore has to be between [0,100]. All the other parameters have the same semantic as in Source code 9.

Source code 11 shows the configuration file of the Gnutella search algorithm. Here, only the parameters $init.1.perc_msg_dist$ and $init.2.ttl$ are relevant parameters. $init.1.perc\_msg\_dist$ has the same semantic as in source code 10. $init.2.ttl$ specifies the TTL of the query message. It should be in the range [1,7].

## A.2 System Manual

This system manual is provided to enable developers to understand and extend our work. We give an overview over the project- and package structure and describe in short the functionality of the most important classes.

**Project Structure:** The project structure follows roughly the Sun project conventions, which can be found on http://java.sun.com/blueprints/code/projectconventions.html. The source code can be found in the package src. All classes used to implement the different algorithms are in subpackages of `hok`. The plain Epidemic Algorithm can be found in `hok.p2p`. It consists of `EpDisseminationProtocol`, a protocol component representing a node of the network. It holds the message buffer and the algorithm of disseminating a message to its neighbours. `EpInitializer` and `SFEpInitializer` contain the algorithms of calculating the parameters used for the dissemination process, which are then stored in the message. The former initializer is used together with Random Graphs, the latter can be used to adapt the algorithm to Scale Free networks.

`hok.epsearch` contains classes for the Epidemic Search Algorithm. The actual search algorithm is implemented in `DynamicEpQueryEngine`, which is used by the protocol class `EpSearchProtocol`. Initializer classes for initializing the query message and the searched file distribution, and a observer class for collecting information complete the package.

`hok.gnutella` contains a portation of Gnutella's Dynamic Query Protocol implementation of PHEX [8]. The structure is very similar to `hok.epsearch` and is used in a similar way. An additional initializer to use a flooding-based search protocol is provided in `GFloodSearchInitializer`.

`hok.msg` contains classes that build the messages that are disseminated by the algorithms. They are used by the Epidemic Search Algorithm and the plain Epidemic Algorithm. The Gnutella implementation uses the simplified message format of PHEX, which can be found in the package `phex.msg`.

`hok.log` contains `PrintStream` implementations to be used together with the Observer classes in the various packages. Please find further information in the

javadoc of the PrintStream classes.

The subfolder `conf` contains the configuration files, which specify which protocol, initializer and other controls components to use. Source code 9, 10, 11 show the configuration we used for our simulations.

The subfolder `doc` contains the javadoc documentation, `lib` the needed additional libraries such as PeerSim and `statData` the simulation data and graphs. The folder statData also contains scripts for GnuPlot to generate the graphs.

## A.3 Project Plan

**Project Title:** Epidemic Dissemination Approaches applied to P2P Networks

**Date:**November 2005

**Name:** Holger Kampffmeyer

**Supervisor:** Cecilia Mascolo

**Aim: Evaluation of epidemic dissemination algorithms based on models of epidemics spreading in complex networks.**

This project will consist in the implementation of an epidemic dissemination algorithm using the PeerSim simulator. The algorithm is based on models of epidemics spreading in networks.

**Language:** Java

**Related links:** PeerSim Simulator webpages http://peersim.sourceforge.net/

**Objectives:**

- To understand epidemic information dissemination techniques.
- To gain knowledge about the epidemic model presented in [31].
- To understand PeerSim and to be able to implement own protocols.
- To develop a system that is able to evaluate the Epidemic Algorithm.
- To evaluate the implemented system.

**Extended Objectives:**

- To research current P2P search algorithms
- To develop the Epidemic Algorithm into a search algorithm
- To find a suitable P2P system that could gain from the Epidemic Search Algorithm.
- To evaluate and compare the search algorithm with an state-of-the-art P2P search algorithm.

**Expected Outcomes/Deliverables:**

- An overview of current P2P applications
- A fully working, documented implementation of the Epidemic Algorithm in PeerSim.
- A comprehensive outline of the theoretical background used in the project.
- A full evaluation using simulation results of all relevant parameters of the implemented algorithms.
-

## A.4 Interim Report

**Project Title:** Epidemic Dissemination Approaches applied to P2P Networks
**Current Project Title:** Improving Gnutella Search Algorithms through Epidemic Dissemination

**Date:**February 2006
**Name:** Holger Kampffmeyer
**Supervisor:** Cecilia Mascolo

**Abstract**

Search algorithms in unstructured P2P networks such as Gnutella use flooding-based techniques for communication and as a consequence, they produce high message overhead. More dynamic algorithms such as Gnutella's Dynamic Query Protocol take into account the user's desired number of results and network topology properties to increase scalability. However, these algorithms only work well for popular files and often fail in locating rare content. Proposed structured approaches such as DHTs are good in finding rare files, but due to their significant overhead and problems with high network fluctuations, they are not very applicable for finding popular content.

In this report, we propose a search algorithm based on epidemic-style information dissemination techniques, that shows good performances in finding both rare and popular files. It exploits the structure of the underlying network and by that maximizes its search horizon and minimizes the number of needed search messages. The presented simulation results show that the search algorithm not only works well in Gnutella-like networks, but would be also applicable in a much broader context such as scale-free networks.

### Progress to date:

- Reading on network theory
- Setup and get to know PeerSim, a java-written P2P network simulator

- Understand epidemic information dissemination / the epidemic data dissemination in MANETs work of Cecilia Mascolo
- Implementation of the Epidemic Algorithm in PeerSim
- Experiment with PeerSim, first diagrams
- Test different network topologies (random graph, scale free networks)
- Reading on P2P and Gnutella
- Understand the code and algorithms of PHEX, a java Gnutella client
- Porting of the Dynamic Search Algorithm of PHEX into PeerSim to compare with EpidemicSearch
- Implement the EpidemicSearch in PeerSim
- Test/Evaluate/Compare the two Search Algorithms
- Produce further Diagrams
- Start writing thesis
- About 20 pages of thesis written so far

**Further work to be done:**

- Reading on network theory
- Finishing simulations
- Production of final diagrams
- Finishing writing thesis

```
# PEERSIM
#random.seed 1234567890

simulation.cycles 100
simulation.experiments 1
simulation.stdout hok.log.EpObserverPrintStream
#control.shf Shuffle
network.size 10000

protocol.0 peersim.core.IdleProtocol
protocol.1 hok.p2p.EpDisseminationProtocol
protocol.1.linkable 0
protocol.1.buffer 1

#init.0 peersim.dynamics.WireKOut
#init.0 peersim.dynamics.WireScaleFreeDM
init.0 peersim.dynamics.WireScaleFreeBA
#init.0 peersim.dynamics.WireWS
#init.0.beta 0.5
init.0.undir true
init.0.protocol 0
init.0.k 15

init.1 hok.p2p.EpInitializer
#init.1 hok.p2p.SFEpInitializer
init.1.protocol 1
init.1.reliability 0.5
init.1.msgTimeValid 100

control.0 hok.p2p.EpObserver
control.0.protocol 1

control.1 hok.stats.EpObserverTopolDist
control.1.protocol 1
control.1.outf epFileDist
```

**Source code 9:** conf/config-file.txt.

```
# PEERSIM
#random.seed 1234567890

simulation.cycles 200
simulation.experiments 1
simulation.stdout hok.log.EpSearchObserverPrintStream
control.shf Shuffle
network.size 10000

protocol.0 peersim.core.IdleProtocol
protocol.1 hok.epsearch.EPSearchProtocol
protocol.1.linkable 0
protocol.1.msgTimeValid 100
protocol.1.defaultProbeReliability 0.2
protocol.1.defaultDesiredResults 150

init.0 peersim.dynamics.WireKOut
#init.0.undir true
init.0.protocol 0
init.0.k 30

init.1 hok.epsearch.EPFileDistInitializer
init.1.protocol 1
init.1.perc_msg_dist 0.1

init.2 hok.epsearch.EPDynamicSearchInitializer
init.2.protocol 1

control.0 hok.epsearch.EPObserver
control.0.protocol 1
```

**Source code 10:** conf/ep-config.txt.

```
# PEERSIM
#random.seed 1234567890

simulation.cycles 200
simulation.experiments 1
simulation.stdout hok.log.GnuObserverPrintStream
#control.shf Shuffle
network.size 10000

protocol.0 peersim.core.IdleProtocol
protocol.1 hok.gnutella.GnutellaProtocol
protocol.1.linkable 0
protocol.1.defaultDesiredResults 150

init.0 peersim.dynamics.WireKOut
#init.0 peersim.dynamics.WireScaleFreeBA
#init.0.undir true
init.0.protocol 0
init.0.k 30

init.1 hok.gnutella.GFileDistInitializer
init.1.protocol 1
init.1.perc_msg_dist 90

init.2 hok.gnutella.GDynamicSearchInitializer
#init.2 hok.gnutella.GFloodSearchInitializer
init.2.protocol 1
init.2.ttl 7

control.0 hok.gnutella.GObserver
control.0.protocol 1
```

**Source code 11:** conf/gnu-config.txt.