Information Diffusion in eDonkey Filesharing Networks

Tobias Hoßfeld*, Kenji Leibnitz†, Rastin Pries*, Kurt Tutschku*, Phuoc Tran-Gia*, and Krzysztof Pawlikowski‡
*Deptartment of Distributed Systems, University of Würzburg, Germany
Email: {hossfeld|pries|trangia|tutschku}@informatik.uni-wuerzburg.de
†Graduate School of Information Science and Technology, Osaka University, Japan
Email: leibnitz@ist.osaka-u.ac.jp
‡Department of Computer Science and Software Engineering, University of Canterbury, New Zealand
Email: K.Pawlikowski@ieee.org

***Abstract*— Peer-to-Peer (P2P) applications have become highly popular in today's Internet due to the spread of file sharing platforms such as KaZaa, eDonkey, and BitTorrent. In this paper we investigate the diffusion behavior of a file shared by multiple sources in an eDonkey-like peer-to-peer (P2P) environment. We provide two simulation models, one which captures the chunk transfer in a detailed way and a simplified model, where the upload bandwidth is equally split among all chunks of the file. The simulation of a single chunk transmission is sufficient to obtain accurate results, which leads to a much shorter simulation time. We further investigate the non-stationary process of file diffusion in a statistically reliable way. Depending on the popularity of the file, we consider flash crowd arrivals and constant arrivals. We derive simple analytical expressions showing the relationship between the propagation of the file, i.e. the number of peers sharing the chunk, and the main influencing factors: e.g. sharing probability, access probability, or arrival rate.**

***Index Terms*— P2P, file-sharing, user modeling, simulation**

I. Introduction

Recently, *peer-to-peer* (P2P) applications have caused an enormous increase of traffic volume provided over the Internet. The P2P technology has been introduced as a new approach in providing an application-specific overlay network structure between the participating *peers* above the Internet topology. In contrast to conventional Internet applications, e.g. WWW or FTP, which have a strict distinction between clients and servers, the peers in P2P act simultaneously as both.

Currently, the most widespread application of P2P is *file-sharing*, e.g. Gnutella, KaZaa, BitTorrent, or eDonkey [1]. Unlike WWW documents that usually consist only of several kilobytes of data, the files shared in a P2P network are usually audio or video files that can range in size from several megabytes up to gigabytes. Numerous studies can be found in the literature that deal with P2P networks. Whereas, many researchers deal with improving the P2P architectural issues, e.g. [2], [3], [4], [5], there are also several publications dealing with measurement and characterization of data from existing file-sharing networks, see [6], [7], [8], [9], [10].

In [6], a measurement based traffic profile of the eDonkey is provided and reveals that there is a strong distinction between download flows and non-download streams. Similar studies for the Gnutella network [7] and BitTorrent [10] exist as well. The authors in [8] and [9] compare measurements made for P2P traffic and show that the popularity of objects deviates substantially from that of Web traffic and does not follow *Zipf's* law. Zipf's law states that the popularity of the ith-most popular object is proportional to $i^{-\alpha}$, where α is the Zipf coefficient. Furthermore, in [9] it is shown that the file popularity in a KaZaa network depends on the time that the object is introduced to the network and in general tends to be short-lived.

In this paper we investigate the diffusion behavior of a file shared by multiple source download in an eDonkey-like environment. Our aim is to study the impact of the main parameters influencing the spreading of a file: the request arrival rate, the sharing probability, and the access speed. The popularity of a specific file depends greatly on the age of the information the file contains. New files are of special interest and this is reflected in the arrivals of requests for this file, whereas older files have a much less popularity and thus, the request rates for these files show a reduced burstiness. Another key parameter which is of interest in this study is the probability of sharing a file after it has been downloaded. In a system with many "selfish" users, who immediately remove their downloaded file from sharing, the file does not circulate so well as when everyone keeps sharing it after downloading.

The paper is organized as follows. In Section II we describe the basic mechanisms of the existing eDonkey P2P architecture. This is followed in Section III by the model and the approximations we used in our simulation study. As this in some cases not fully corresponds to the actual eDonkey mechanisms, we refer to it as an eDonkey-like network. In Section IV we discuss further aspects on the employed simulation strategies. Numerical results with analytical approximations are given in Section V and Section VI gives a conclusion and outlook on future work.

II. The eDonkey P2P File-sharing Application

The eDonkey file-sharing application [1] belongs to the class of hybrid P2P architectures and comprises two applications: the *eDonkey client* (or *peer*) and the *eDonkey server*. The eDonkey client shares and downloads files. The eDonkey

server operates as an index server for file locations and distributes addresses of other servers to clients.

A. Searching and Sharing of Files

Before an eDonkey client can download a file, it first gathers a list of all potential file providers. To accomplish this, the client connects to one of the eDonkey servers. Each server keeps a list of all files shared by the clients connected to it. When a client searches for a file, it sends the query to its main server which may return a list of matching files and their locations. If none or an insufficient number of matches is returned, the client may resubmit the query to another server.

B. Downloading of Files

A main feature of P2P file sharing applications like Kazaa and eDonkey is the *multiple source download* mode, i.e. peers can issue two or more download requests for the same file to multiple providing clients in parallel and the providing clients can serve the requesting peer simultaneously.

In eDonkey, the multiple source download is enabled by dividing files into fixed size pieces, denoted as *chunks*. A chunk has a size of approximately 10 MBytes. The consuming client can reassemble the file using the chunks or parts of chunks. A client can share a file as soon as it has received a complete chunk.

When an eDonkey client decides to download a file, it asks the providing peers for an upload slot. Upon reception of a download request, the providing client places the request in its *upload queue*. A download request is served as soon as it obtains an upload slot. The clients in eDonkey may restrict their total upload bandwidth to a given limit.

The upload management of a peer maintains an upload queue which consists of two lists, the *waiting list* and the *uploading list*. The uploading list holds the requests which are currently served. A download request is served as soon as it obtains an upload slot, i.e. it moves from the waiting list to the uploading list. Typically, each served request gets an *equal share* of the upload capacity. However, different modifications (*mods*) exist, which may change this behavior [11]. The complex *scoring mechanism* of eDonkey decides which request is served next. One important factor of the scoring system is the "high ID/low ID" mechanism to ensure fairness among all peers. A high ID increases the score, whereas a low ID reduces it. A peer gets a low ID if the server cannot establish a new connection to the peer, e.g. the peer is located behind a firewall or a NAT. The blocking of incoming connections or an invalid IP address would hinder to contact this peer and is unfair since these peers would not answer to file requests.

Further details on the eDonkey architecture, performance, and the download mechanisms can be found in [6], [12], [13].

III. Simulation Model

In this section we describe the underlying model used for the simulation studies in this paper. We elaborate on the downloading mechanism in the eDonkey network and discuss the approximative assumptions in our model.

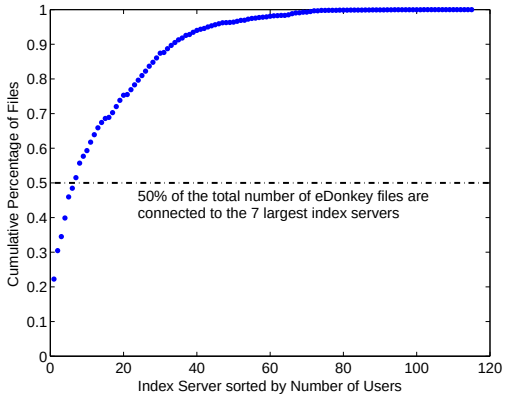

Fig. 1. Number of users connected to different index servers

A. Modeling the Index Server Population

As described in Section II, all peers are connected to an index server and send a list of files they share to this index server. When a search request is issued, the index server notifies the file requesting peer about all clients which share chunks of this file. For our investigation, we measured typical values of the population size of public eDonkey servers found in the Internet [14]. The list in [14] revealed that the largest index servers can host up to 500,000 peers, whereas about half of the servers have less than 1,000 connected peers. Fig. 1 shows the cumulative percentage of peers connected to different index servers. On the abscissa, we listed the index servers sorted decreasingly by the number of connected peers. We can see that 50% of the total number of eDonkey users are connected to the seven largest index servers. Thus, it is reasonable to assume population sizes between 50,000 and 500,000. Since the connected number of peers at these index servers is so large, it is justified to assume a Poisson process for the generation of file requests. However, as the popularity of the requested file varies over time, we assume a time-dependent arrival rate $\lambda(t)$, which is non-stationary during one day and periodic on a daily basis.

B. Shared Files in eDonkey

The general structure of an arbitrary file f that is shared in the eDonkey network is depicted in Fig. 2. The file has a size of f_{size} kB and comprises a number of $c_{max} = \lceil \frac{f_{size}}{c_{size}} \rceil$ chunks, each with a constant size of $c_{size} = 9500$ kB with

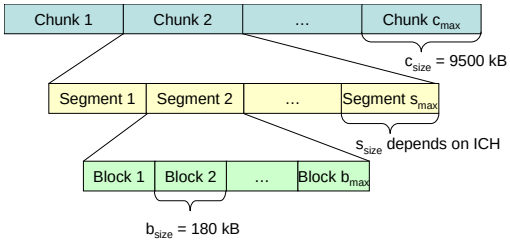

Fig. 2. Structure of a file on eDonkey application layer

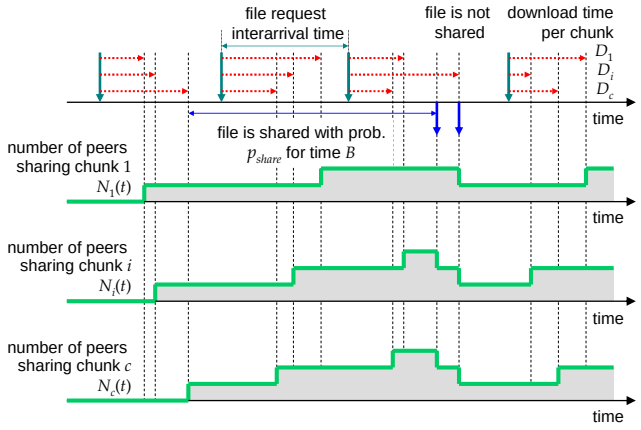

Fig. 3. Arrival process of peers sharing chunks of a file

exception of the final chunk c_{max} which may be smaller in size. The operator $\lceil x \rceil$ returns the next largest integer value of x. Whenever a chunk is shared, it is transmitted in units of blocks with size $b_{size} = 180$ kB. In our studies we consider two different file sizes. One corresponds to an mp3-audio file with a data size of 5 MB, thus occupying only a single chunk. The other considered file type is a complete mp3 album archive of 76 MB and consists of several chunks.

The number of peers sharing a certain chunk i at time instant t is denoted as $N_i(t)$. After a peer has successfully downloaded all blocks of chunk i, he immediately acts as a *sharing peer* for this chunk and the number $N_i(t)$ is incremented by one. Thus, being a P2P application, all users in an eDonkey network act simultaneously as sharing peers and downloading peers. Although, the user cannot influence that each chunk is shared during downloading, he can show a different behavior after the file has been entirely downloaded. We take this into account in our model by introducing p_{share} as the probability that a user shares file f for an exponentially distributed period B. All users in the system use the identical values of p_{share} and B. Hence, $p_{share} = 0$ indicates a system consisting entirely of *leechers*, i.e. users who only share the file during the download and immediately stop sharing it once the download is completed. The sharing process in our eDonkey model is illustrated in Fig. 3.

C. Error Detection and Recovery

In the original version of eDonkey, error detection is done after all blocks of a chunk have been received and the complete chunk is discarded in case of an error. However, in more recent versions of eDonkey clients, e.g. eMule, the *Intelligent Corruption Handling* (ICH) mechanism is implemented that performs the error detection on smaller data units than chunks and which we define in the following as *segments*, see Fig. 2. Instead of discarding the complete chunk when at least one corrupted block is received, only all blocks of the damaged segment need to be re-requested. The size of a segment depends on the ICH mechanism and we assume in our study that a chunk consists of two segments, i.e. $s_{max} = 2$.

D. Upload Queue Management

Let us consider an arbitrary peer x wishing to download file f. The peer issues a request for the file to the index server and receives a list of all known peers that share the complete file or chunks of it. Note that in the existing version of eDonkey additional methods are used to obtain further sharing peers. The eDonkey protocol implements a *source exchange* mechanism between peers which also permits to access files from peers that are connected to other index servers. This feature becomes mainly effective during long connection durations. For the sake of simplicity, we only consider a single index server in our model and approximate source exchange by using a sufficiently large population of connected peers.

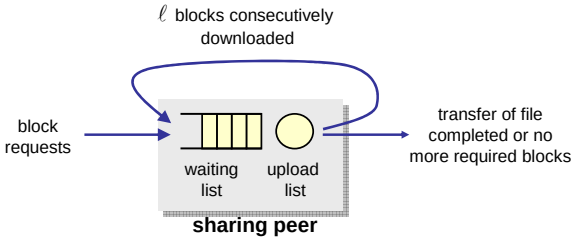

Fig. 4. Upload queue model

The peer requests individual blocks from other peers sharing the chunk containing the desired block. All requests are appended to the waiting list of the sharing peer and a weighting mechanism handles the scheduling of the upload queue requests for transmission. The detailed procedure of the queue management takes several features into account that depend on the individual settings of the sharing peer like upload bandwidth and number of simultaneous uploads. In our model, an approximative assumption simplifies the upload queue management behavior. If a peer downloads a block from another peer, additional blocks might be of interest, if the latter is not already sharing the complete file. The weighting mechanism takes this into account by giving higher priority to peers from which blocks have been previously downloaded. We include this interaction by considering that not individual blocks but rather a series of ℓ blocks is downloaded at a time after moving from the waiting list to the uploading list. The waiting list is modeled as a FIFO (*first-in-first-out*) queue and the value ℓ is estimated from measurements [6] that yield the average data volume downloaded per sharing peer. After the ℓ blocks have been transmitted, the user may issue another request for further blocks, cf. Fig. 4.

If we include all these eDonkey mechanisms in detail, the simulation is very close to reality. However, it would take a whole day to simulate 12 days with the detailed model. In order to study the general behavior of the diffusion of a file f in the eDonkey network, we need to make some additional assumptions at this point to reduce the computation time.

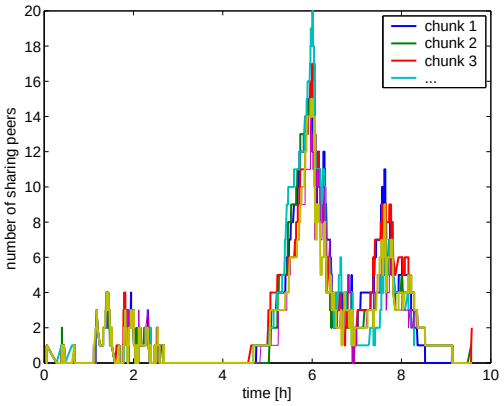

Fig. 5. Detailed simulation of the eDonkey specific mechanisms

The available bandwidth of a downloading peer consists of the sum of bandwidths of all connections to the sharing peers. Therefore, we assume that the available bandwidth $R(t)$ is split equally among the number c_{max} of chunks which corresponds to the simultaneous download of chunks from multiple sources. The calculation of the bandwidth is described in the following section.

As all blocks are downloaded in parallel, they will be finished at the same time and there is no need to simulate individual chunks. Thus, the number of sharing peers is equal for each chunk. This assumption is reasonable because the probability to choose chunk i for downloading the next ℓ blocks is equal for all chunks which offer not yet downloaded blocks. Fig. 5 shows the number of sharing peers for each chunk for the detailed simulation. We see that the number of sharing peers changes similarly for all chunks. If we perform several simulation runs and compute the mean values and the corresponding confidence intervals, the curves for the chunks are identical, which justifies our assumption. This is quite obvious, as the probabilities for choosing each chunk for downloading the next blocks are equal.

E. Max-Min Fair Share Bandwidth

All users can access the Internet with different technologies, like modems, ISDN, or DSL. This and the individual settings of each user have a great influence on the downloading speed of a block. An important parameter that each peer can set in eDonkey is the number of concurrent uploads, which imposes an upper limit on the upload list size.

In our model, we use an unlimited length of the upload list and split the upload bandwidth among all requesting peers. The total available upload bandwidth U is distributed according to the *max-min fair share* principle, cf. [15]. U is the sum of the upload bandwidth C_a^{up} of each providing peer and each access class. Let us assume that C_a^{down} is the downlink access speed for traffic class $a \in \mathfrak{A}$ in ascending order and K_a is the number of users downloading at this speed. We initialize the remaining traffic $\tilde{U}$ with the whole upload bandwidth and the remaining number of users $\tilde{K}$ as the sum over all K_a. As long as there is bandwidth left, we iterate over the following steps. If the remaining bandwidth $\tilde{U}$ can accommodate all $\tilde{K}$ users, we assign the bandwidth C_a^{down} to each user of the class a. Then we reduce the remaining bandwidth and the remaining number of users by those who are downloading at this speed. In the case that $\tilde{U}$ cannot support all users at the maximum speed C_a^{down} (of the slowest remaining access class), we simply share $\tilde{U}$ among all remaining users. The algorithm for max-min fair share is summarized in Table I.

TABLE I
THE MAX-MIN FAIR SHARE ALGORITHM

Input:
U total upload bandwidth of all sharing peers
C_a^{down} access speed for traffic class a in downlink direction
K_a downloading peers using traffic class a
$\tilde{U}$ remaining upload bandwidth
$\tilde{K}$ remaining number of users
Output:
downloading bandwidth R_a for each access class a
Algorithm:
initialize $\tilde{U} = U$ and $\tilde{K} = \sum_{a \in \mathfrak{A}} K_a$
for all access classes a in ascending order of C_a^{down}
if all users can be accommodated at the bandwidth C_a^{down}
i.e. $C_a^{down} \times K \leq \tilde{U}$
then
reduce $\tilde{U}$ by $C_a^{down} \times K_a$ and reduce $\tilde{K}$ by K_a
assign all users of class a the speed $R_a = C_a^{down}$
else
assign all remaining users the speed $R_a = \tilde{U}/\tilde{K}$
endif
endfor

IV. SIMULATION STRATEGY

In this section, we give a short overview of how the event-driven simulation is implemented. We focus on the file request arrival process and the statistical evaluation of the simulation results. The corresponding numerical results which deal with both topics are found in Section V-A and Section V-B, respectively.

One important parameter of a simulation scenario is the file request arrival rate $\lambda(t)$ which reflects the popularity of a file. The more popular a file is, the higher is its rate. The corresponding arrival process is modeled by a nonstationary Poisson process, since $\lambda(t)$ depends on daytime, but is kept constant over each day. The reason is that we measured the number of requests for different files at an eDonkey peer connected to the currently largest index server. We detected two noticeable patterns which occurred for several files:

- *flash crowd arrivals* with a large leap of λ for one specific day followed by a smooth decrease, and
- *constant arrivals* with no significant differences from one day to another.

The first type can be interpreted as the reaction of the peers to a temporarily popular file. We observed this behavior for audio files requested in the eDonkey network one day after they were presented in a popular German TV show. This means that the popularity of this file differs from day to day. The constant arrivals may be typical for “classics”.

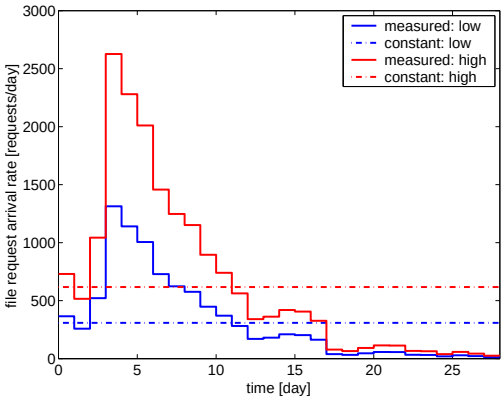

Fig. 6. File request arrival rates of the nonstationary Poisson process

A problem of the measurements is related with the fact that we did not measure the file request rates in eDonkey, but the number of requests which is always increased by one if a user (re-)enters the upload list, cf. Section II. Thus, the measured values are higher than the real file request rates and we adapted the absolute values according to [10]. There, a single file in the BitTorrent P2P network is considered. The resulting $\lambda_{high}(t)$ is referred to as *high measured rate* and illustrated in Fig. 6. In order to compare the two different file request patterns, we compute the corresponding constant, time-independent rate λ_{high} to generate the same number of file requests like $\lambda_{high}(t)$ within the observation period of 23 days: $\lambda_{high} = \int_0^{23} \frac{\lambda_{high}(t)}{23} \mathrm{d}t$. The low measured and constant rates are chosen to be half of the high ones.

Since there is a large peak for the measured rates, the common thinning-method is not practical, because too many random numbers are generated which are rejected. Therefore, we use a modified version of the proposed method in [16] which is based on the inverse-transform method. The random stream *rndStream* is initialized by generating a Poisson arrival time at rate 1.

Another important aspect of evaluating simulation runs is the statistical reliability of the results. We consider a simulation scenario which is replicated r times. In each simulation run, a set $\mathfrak{X}$ of investigated measures, like the number of sharing peers, the number of downloading peers, or the assigned bandwidth per access class, is returned for each time instant when one of these measures changes due to an event-driven simulation. We discretize the time into intervals of length Δt and get the mean value X_i for each measure $X \in \mathfrak{X}$ during the interval $i\Delta t$. Since the simulation is replicated r times, the confidence intervals $C_\gamma[X_i]$ at a confidence level γ and the mean value $E[X_i]$ for the measure X during time interval $i\Delta t$ can be computed.

The relative error ε_γ is the maximum width of confidence intervals normalized by the corresponding mean value for all measures over all time intervals. The smaller the relative error is, the more reliable are the statistical results. Results for the required number of replications are given in Section V-A.

TABLE II

ACCESS SPEEDS OF THE DOWNLOADING AND PROVIDING PEERS

access A	upload	download	$P(A=a)$
modem	28 kbps	28 kbps	5%
ISDN	64 kbps	64 kbps	5%
ADSL 1Mbit	1024 kbps	128 kbps	60%
ADSL 2Mbit	2048 kbps	192 kbps	10%
ADSL 3Mbit	3072 kbps	384 kbps	20%

V. PROBLEM ANALYSIS AND NUMERICAL RESULTS

In this section, we show the numerical results for simulations of the abstract model defined in Section III. The file requests are generated according to Section IV with time-dependent arrival rates plotted in Fig. 6. In the considered scenarios, we assume the following set $\mathfrak{A}$ of different access types with which the peers are connected to the eDonkey network: $\mathfrak{A} = \{$modem, ISDN, ADSL1, ADSL2, ADSL3$\}$. We suppose that these access types cover the most relevant and common ones for eDonkey users. The access speeds in uplink and downlink direction are given in Table II. We assume that the users do not limit the upload and download bandwidth in eDonkey. Thus, the upload/download eDonkey bandwidth of a peer is equal to its uplink/downlink access speed and we do not differ between both. The impact of the probability $P(A=a)$ that a user has access type $a \in \mathfrak{A}$ is investigated in Section V-C. There, we derive the relationship between the number N of sharing peers and $P(A=a)$ analytically. This means that the choice of $P(A=a)$ is arbitrary. In our simulations we use the values given in Table II.

Other parameters are the probability p_{share} to share a file after downloading it and the initial number of sharing peers N_0. The influence of these parameters are shown in Section V-D and Section V-F. We consider two different file types, an mp3-audio file and a complete audio album as a single, compressed file. Both content types are predominantly found in the current eDonkey network. Due to the fact that the files are already in a compressed format, the compression of data packets of eDonkey on application layer has no further effect. This means that the only difference between both content types is the file size, the resulting download time, and the time to share the file after downloading it. Table III shows the parameters for the simulation. These values and the low measured file request arrival rate are our default parameters if not stated otherwise.

The typical behavior for sharing files is based on the following assumptions. We assume that a user requesting an mp3-audio file shares it with a low probability of 10%, as the download takes only several minutes. Thus, the user is more likely to observe when the download is successful and may then move the file into another directory, but if the user wants to share the file, he shares it for a longer period of time, e.g. 1 day. On the other hand, the download of a complete audio album requires much more time than for a single file because of the increased data volume. When a user initiates the download of an album, he does not know the exact moment

when it will be completed and will share it with a higher probability of 40%.

TABLE III
DIFFERENT KINDS OF INVESTIGATED FILES IN THE SIMULATION

content	size	chunks	p_{share}	mean sharing time
mp3-audio	5 MB	0.5	10%	24 hours
album	76 MB	8	40%	5 hours

We further measured the file size of mp3s and retrieved a mean value of 5 MB. A complete audio album consists of 15 single files and has a file size of about 76 MB. In order to assure that the time for downloading a chunk is equally distributed for each chunk of a file, cf. Section III-D, the file size is chosen to be a multiple of the chunk size. Since we did not determine any qualitative differences in the results between both file types, we only present the results for the mp3-file. However, the influence of the file size is significantly perceivable with respect to the utilized bandwidth Δ, cf. Section V-E, and the number of initially sharing peers N_0, cf. Section V-F.

A. Replications for Falling Below an Error Level

In Section IV, we introduced the relative error ε_γ to describe the statistical reliability of our results for r replications of a simulation scenario. We consider the most varying scenario with the high measured arrival rate. Fig. 7 shows the relative error ε_γ for different confidence levels γ depending on the number r of replications. In order not to exceed a relative error of 5%, at least $r \geq 700, 1000, 2000$ replications are needed for given confidence levels of $\gamma = 0.90, 0.95, 0.99$. We consider 1000 replications to be sufficient, i.e. the relative error is below 5% and 7.5% for $\gamma = 0.95$ and $\gamma = 0.99$, respectively.

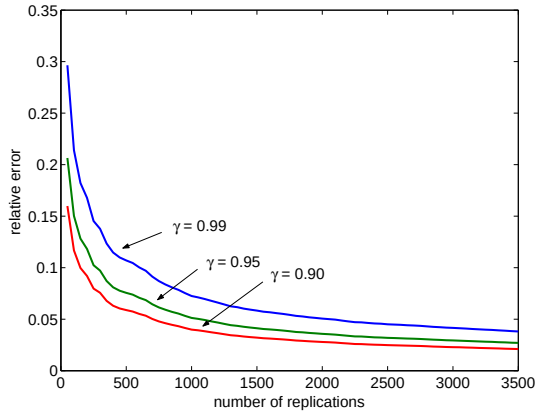

Fig. 7. Relative error in dependence of the number of replications

Fig. 8 shows the total number of sharing peers. We see that the confidence intervals are small enough for error to fall below 5% and will therefore no longer plot confidence intervals in the figures.

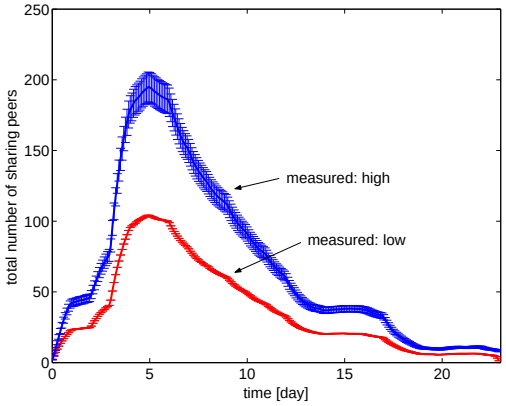

Fig. 8. Number of sharing peers for measured arrival rates

B. Flash Crowd Arrivals vs. Constant Requests Rates

The file request arrival rates reflect the popularity of a file. A constant file rate may be typical for an older song and the flash crowd arrivals for files which are very popular for a short time and are then less requested. We are able to determine the steady-state number N^* of sharing peers for the scenario with a constant rate λ when the system has passed the initial transient phase and reaches the steady state.

$$N^* = \lambda \cdot p_{share} \cdot E[B]. \tag{1}$$

With the mean sharing time $E[B] = 1$ day, we obtain $N^* = 30.8468$, see Fig. 9, but we cannot predict analytically the time instant, when the system reaches this state. Therefore, the simulation is needed and looking at Fig. 9 we see that the scenario reaches the steady state at day five. The reason is that the download time depends on the current number of users in the system due to the max-min fair share of the available total upload bandwidth U. For the flash crowd arrivals with a time-dependent file request arrival rate, we even cannot analytically determine the maximum peak which occurs after 5 days.

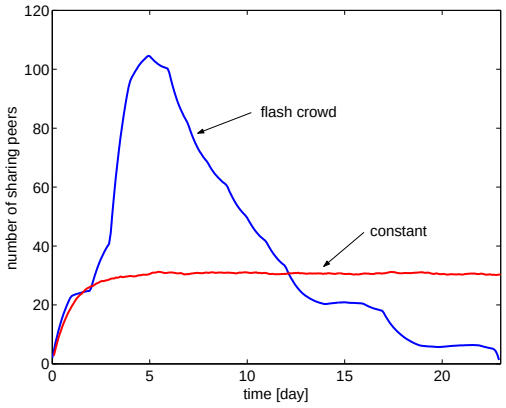

Fig. 9. Number of sharing peers for flash crowd and constant behavior

C. Influence of the Access Type

The probability $P(A = a)$ determines the access types for a new requesting peer according to Table II. Fig. 10 shows the number N_a of sharing peers separated by their access types $a \in \mathfrak{A}$. Obviously, the larger $P(A = a)$ is, the larger is also N_a. Indeed, we find the following relationship between the total number N of sharing peers and the number N_a of sharing peers with access type a for each arbitrary time instant.

$$N_a = P(A = a) \cdot N = P(A = a) \cdot \sum_{a \in \mathfrak{A}} N_a \qquad (2)$$

This means that the number N_a is directly proportional to N with factor $P(A = a)$. If we choose another set of access types and corresponding probabilities, we obtain the same results.

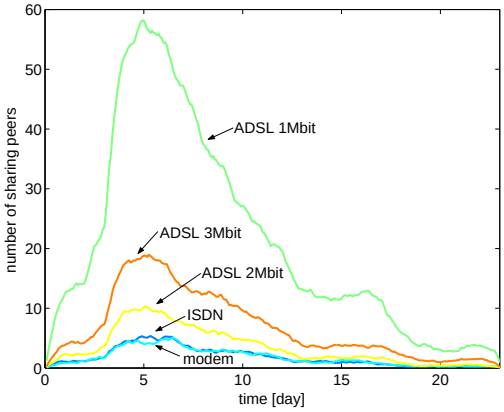

Fig. 10. Number of sharing peers in dependence of the access bandwidth

D. Impact of the Sharing Probability

Fig. 11 illustrates the influence of the sharing probability p_{share} on the number $N(p_{share})$ of sharing peers. Like in Section V-C, we see a linear relationship. If we plot the number $N(p_{share})$ normalized by the sharing probability, we retrieve exactly the curve for $N(100\%)$. Analytically, this behavior is described for each time instant by

$$N(p_{share}) = N(1) \cdot p_{share}. \qquad (3)$$

Although Equation (1) shows the linear influence of p_{share}, the Equation (3) is not obvious because the system is not in steady state.

E. Utilized Bandwidth

The utilized bandwidth Δ is the total available upload bandwidth U minus the maximum bandwidth D with which each user could potentially download. For each time instant, it holds that

$$U = \sum_{a \in \mathfrak{A}} N_a \cdot C_a^{up} \qquad (4)$$

$$D = \sum_{a \in \mathfrak{A}} K_a \cdot C_a^{down} \qquad (5)$$

$$\Delta = U - D \qquad (6)$$

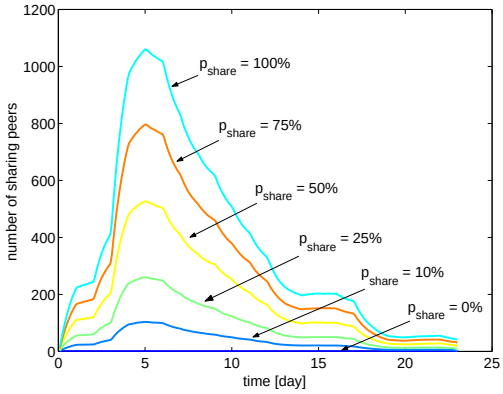

Fig. 11. Number of sharing peers depending on the file sharing probability

This means that if $\Delta > 0$, there is wasted upload capacity which cannot be utilized by the downloading peers because of their limited download bandwidth C_a^{down}. On the other hand, if $\Delta < 0$, the peers cannot download with their possible physical bandwidth C_a^{down}. In this case, the max-min fair share algorithm is applied.

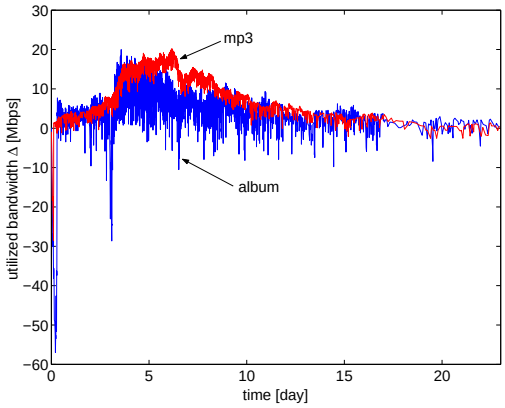

Fig. 12. Utilized bandwidth for an mp3-audio file and an entire album

Fig. 12 shows the utilized bandwidth over time for two single simulation runs: download of an mp3-audio file and of an entire album. For the latter, the peers require much more time to download it. Thus, there are more peers which share the total available upload capacity U than for downloading the mp3-audio file, i.e. $\Delta < 0$ occurs more frequently.

We now consider the download of the entire album for the low constant file request rates scenario. In Fig. 13, the used download bandwidth δ_a of the different access types is plotted over time. It is defined as the ratio between the assigned bandwidth R_a due to the max-min fair share algorithm and the maximum download bandwidth C_a^{down}, i.e. $\delta_a = \frac{R_a}{C_a^{down}}$. The higher the download bandwidth, the worse it can be utilized.

F. Initial Number of Sharing Peers

In our simulation model, we assume that the total upload bandwidth is divided fairly among the requesting peers ac-

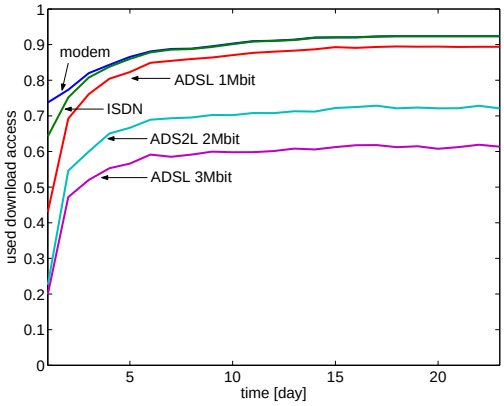

Fig. 13. Used download bandwidth over time

cording to the max-min fair share algorithm. Furthermore, we assume that all chunks are downloaded after the same time. This is correct when we consider files consisting of a single chunk. However, in the case of albums, these assumptions can lead to wrong results, since in real eDonkey networks the number of accepted downloading connections per peer is limited. Therefore, a minimal download time exists for a chunk and after downloading the chunk, the user shares it immediately. Thus, the file diffuses in the network.

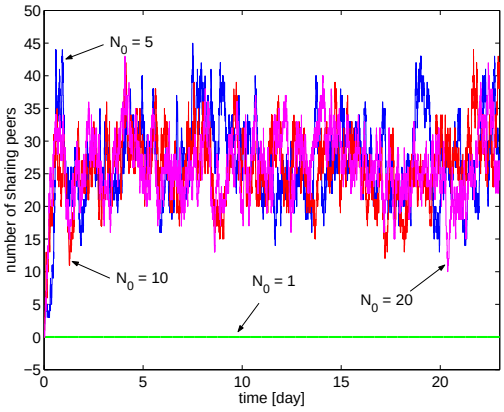

Fig. 14. Influence of the number of initially sharing peers

Fig. 14 shows the number of sharing peers (for single simulation runs with low constant request rates) in dependence of the initial number of sharing peers. For $N_0 = 1$, no file request succeeds during the first 23 days in our simulation model. The reason is that the total upload capacity is divided by all requesting peers according to the max-min fair share algorithm. But this problem can be coped easily by skipping the initial period of the file diffusion. We start investigating the system, when there are already $N_0 = 5$ peers available in the eDonkey network. Then, we see in Fig. 14 that our simulation model is valid again.

VI. Conclusions and Outlook

In this paper we discussed models for the diffusion of a file via multiple source download in an eDonkey-like file-sharing environment. Our focus was on how a file is propagated in the network under different conditions, e.g. initial number of sharing peers, sharing probability, access speed, or file size. We provided two simulation models, one which captured the chunk transfer in a detailed way and a simplified model, where the upload bandwidth was equally split among all chunks of the file. The simulation of a single chunk transmission is sufficient to obtain accurate results, which leads to a much shorter simulation time.

We investigated the non-stationary process of file diffusion in a statistically reliable way. Depending on the popularity of the file, we considered flash crowd arrivals and constant arrivals. We found simple analytical expressions showing the relationship between the propagation of the file, i.e. the number of peers sharing the chunk, and the main influencing factors: e.g. sharing probability, access probability, or arrival rate.

The simplified model showed good results when considering mp3 audio files or audio archives. However, the simplified model fails when we consider very large file sizes, like the sharing of complete CD data with 700 MB or entire DVDs with 4 GB. The extension of the abstract model to very large file sizes and the enhancement of the analytical equations are subject of further work.

References

[1] "eDonkey2000 Home Page," `http://www.eDonkey2000.com/`.
[2] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. of ACM SIGCOMM01*, San Diego, CA, Aug. 2001.
[3] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to- peer systems," in *Proc. of the 18th IFIP/ACM Int. Conf. on Distr. Systems Platforms (Middleware 2001)*, Heidelberg, Germany, Nov. 2001.
[4] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, Jan. 2004.
[5] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara, "Scalable and continuous media streaming on peer-to-peer networks," in *3rd Intern. Conf. on Peer-to-Peer Computing (P2P2003)*, Linkoping, Sweden, 2003.
[6] K. Tutschku, "A measurement-based traffic profile of the eDonkey filesharing service," in *5th Passive and Active Measurement Workshop (PAM2004)*, Antibes Juan-les-Pins, France, Apr. 2004.
[7] K. Tutschku and H. deMeer, "A measurement study on signaling on gnutella overlay networks," in *Kommunikation in Verteilten Systemen (KiVS) 2003*, Leipzig, Germany, Feb. 2003.
[8] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," *IEEE/ACM Trans. on Networking*, vol. 12, no. 2, Apr. 2004.
[9] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proc. of ACM SOSP03*, Bolton Landing, New York, USA., Oct. 2003.
[10] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garc es Erice, "Dissecting BitTorrent: Five months in a torrent's lifetime," in *5th Passive and Active Measurement Workshop (PAM2004)*, Antibes Juan-les-Pins, France, Apr. 2004.
[11] "eMule Forum," `http://www.emuleforum.net/`.
[12] "eMule Project Team," `http://www.emule-project.net`.
[13] F. Lohoff, "Lowlevel documentation of the eDonkey protocol," `http://silicon-verl.de/home/flo/software/donkey`.
[14] "eDonkey Network Server List," `http://ocbmaurice.dyndns.org/pl/slist.pl`.
[15] D.P. Bertsekas and R. Gallagher, *Data Networks*, Prentice-Hall, 1992.
[16] A.M. Law and W.D. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, 2000.