Advanced Application-Level Crawling Technique for Popular Filesharing Systems

Ivan Dedinski and Hermann de Meer
University of Passau , Faculty of Computer Science and Mathematics
94030 Passau, Germany
{dedinski, demeer}@fmi.uni-passau.de

Acknowledgement: This project was partly funded by the German Research Foundation (Deutsche Forschungsgemeinschaft - DFG), contract number ME 1703/4-1 and by EPSRC, contract number GR/S69009/01.

Abstract

P2P filesharing systems are causing the largest traffic ammount in todays Internet, which explains the interest of the research community. On the other hand, most of the filesharing trafic is caused by the exchange of illegal content. That makes research participation in such systems hard, since the systems try to protect themselves from observation. This paper presents an application level crawling technique for current filesharing systems that exploits the minimal openness of the filesharing system to perform a broadband content scan with minimum ressource usage. Such a technique can be used to continuously scan a filesharing system. The gathered information can be used by researchers for studying the dynamics of P2P systems or by companies trying to protect their copyrights. It also could be usefull to influence the behavior of such P2P systems, e.g., by an ISP traffic engineers. The technique was extensively evaluated through a series of measurements in the eDonkey filesharing network. The information gathered gives interesting insights about the behaviour of the users doing filesharing. Some behaviour patterns were found that influence the performance of the suggested technique in a very positive way, proving its feasability. These patterns indicate that a filesharing system should not only be regarded as a technical system, but has to be also viewed as a social network.

1 Introduction

P2P filesharing systems are an important phenomena, since in only few years they became the dominating Internet application, if the generated traffic is considered. These systems transfer thousends of terrabytes daily, often causing problems for Internet Service Providers (ISPs) by overloading their network infrastructure. The problems are not only due to the high traffic load, bit primary due to the hardly predictable nature of the P2P traffic.

In client-server networks for example, where traffic is concentrated in the direction of the servers, these servers usually don't change their positions or go on and off frequently. This ensures a certain predictability which can be used by ISPs e.g. to overprovision their networks at the right place. P2P traffic on the other hand is not directed to any server but is transferred among end users, which makes it highly dynamic. Traffic directed to one part of the network may suddenly change its direction and overload another part. Traffic bursts occur in a hardly predictable manner, e.g. caused by a new interesting content published by an user somewhere in the Internet. Not only the place of occurence but also the spreading dynamics of a certain content can not be easily predicted since it also depends on a high number of (not only technical but also social) factors. Here the monitoring of the lifecycle of traffic-intensive content is essential for a fast reaction to traffic changes (and their prediction to some extent).

Not only ISPs are interested in getting control and insight in P2P filesharing networks. Legal issues in P2P filesharing (piracy) are also hevily discussed nowadays. Music and film industries as well pretend that they are loosing milions of dollars due to illegaly distributed content in P2P networks. However, it is not easy to calculate this precisely, since P2P filesharing networks do not provide any statistical information about their content and users. A company having such information could be able to protect its interests by starting actions against portions of the P2P network, where its content is shared most heavily.

And finally, researchers interested in P2P systems and optimization of P2P traffic also need statistically relevant data about the shared content in popular filesharing networks. By finding patterns in the user behavior

and observing the content dynamics, new optimization strategies and protocols may be developed.

It is obvious that the information necessary to satisfy the above requirements could only be collected through continuous observation. Continuous observation means continuously performing P2P user and content identification [2]. It should answer the question of what content is shared, where is it shared (who shares it) and, eventually, give an estimation of how much traffic will this content produce in future. And it should answer it fast, since P2P networks are dynamic systems, users and content are appearing and disappearig continuously.

This paper presents a measurement architecture which can continuously perform the bigger portion of the P2P identification task in an efficient way - it collects the information on what content is shared in the eDonkey network and how much traffic would it probably produce. The question of where is this content located is somewhat problematic - it requires the identification and collection of IP addresses, which is currently illegal and will probably remain so. Nevertheless, we argue that in current filesharing networks it is straightforward to identify the content locations and present a strategy how to use this adequately, e.g. for traffic optimization purposes.

In Section 2 background on hybride P2P systems is provided. In Section 3 the architecture used with our crawling technique is described. In Section 6 the measurement setup is described and the gathered data is presented and analysed.

2 Background

Filesharing made P2P systems really popular during the recent years. The first widely spread P2P filesharing system was Napster [4]. Generally, it was an indexing server where the Napster users could publish information about the music files they were providing and search for new files. The main advantage of Napster compared to FTP or HTTP servers was the load distribution - the users were downloading the content directly from each other, the server was only used for locating the content.

Although Napster had great success, it was closed due to legal issues. As a reaction, the P2P community presented more resilient P2P designs like Gnutella [7] at the beginning, then eDonkey [5], BitTorrent [5], KaZaa [6], etc. Gnutella, a completely decentralized system, had serious scalability problems. For this reason currently the so called hybride architectures (eDonkey, BitTorrent, KaZaa, etc.) are dominating the Internet.

A hybrid architecture is a mixture of a fully decentralized approach like Gnutella and a centralistic approach like Napster. In the example case of eDonkey there are multiple (few hundreds) of superpeers or eDonkey indexing servers to which the eDonkey users can connect. From the user perspective, the eDonkey indexing server has similar functionality as a Napster server - it allows the users to publish and search for files on that server only. The eDonkey indexing servers however all participate in a fully decentralized (Gnutella-like) server network, which allows clients to extend their search to other servers if necessary. This indexing server network has two properties which ease the task of the observation technique presented in this paper. First, it is of a much smaller size as the whole P2P network, which makes it easy to track. Second, it stores most of the information the observation system is interested of, see Section 4.

Since the hashID of a file is unique with very high probability, but the file name might not be unique, eDonkey uses a two step query mechanism. A client first starts a query for file names containing certain keywords, e.g. “Madonna, mp3”. The eDonkey indexing server replies with a list of names containing the keywords specified, the corresponding hashIDs and some additional data (like file size, number of sources for the file, number of complete sources) which is helpful for the client to choose which file to download. The client then chooses a file and sends its hashID to the indexing server. The server responds with a list of clients providing that file. The quering client then contacts some or all of these clients and starts the download. This two-step query mechanism allows to query content names and properties without collecting any IP-addresses of providing peers, which could lead to legal problems.

One of the reasons why eDonkey is so successfull is the possibility to download a particular content from many providing clients simultaneously. This feature is called Multi Source Download Protocol (MSDP). To be able to parallelize the download like this, each file is splitted in chunks of equal size and each chunk can be transferred independently from the others. Parallel downloads requires strict checksumming to ensure that chunks downloaded from different clients belong to the same file. Thus every chunk and also the whole file is uniquely identified by a checksum called *hashID*. BitTorrent [5] is another popular P2P network that heavily relies on MSDP.

Obviously, the goal of MSDP is to distribute the traffic load fairly among the participating peers and to utilize any available bandwith, causing that content is transferred as fast as possible. To be able to do this, P2P clients need a mechanism to learn other clients for additional parallel connections. In the eDonkey network this is done by gossiping - if a client A is already downloading a content X from client B, A could ask B to tell him all its sources (their IPs) for that content X. By doing this procedure recursively, an observation system could identify the IPs of all peers currently downloading the content X This is important, since such information can

not easily be obtained from elsewhere, e.g. from the eDonkey indexing servers, see next paragraph.

Since a large part of the eDonkey traffic is illegal, the eDonkey servers implement a number of protective mechanisms against observation and attacks like DOS. First of all a query has to contain at least one keyword and returns up to a certain limit of results, mostly 300. Subsequent queries for the same keyword mostly return the same result - the result could only differ if the result set on the server has changed. So it is not easy to query all files containing a keyword, if the number of files is higher than the limit. Second, a client can not start queries too often, because the eDonkey server would put it at a blacklist for a while. The blacklisting times are set per server basis and can vary up to several hours. These two measures not only protect the eDonkey network from overloading, but also make the reverse resolution of the file providers hard - it is not easy to discover all the content shared by a client with a given IP. Another reason for this is that the big majority of eDonkey clients in the Internet do not answer queries about the files they share, although such a query is provided by the eDonkey/eMule protocol.

3 Architecture Basics

This section presents an application level crawling architecture performing much of the observation tasks specified in Section 1 - it discovers as much as possible popular content in the eDonkey network for a given time. It collects information about the discovered content like content name, content hash ID, currently available sources, complete sources, etc. The popularity of the content is defined by the currently available sources for this content in the network. Another important criteria is the number of incomplete sources, since it indicates how many clients are still trying to download the content and are thus producing traffic.

A novelty of the architecture presented here is its ability to not only make snapshot of what is currently shared in the P2P network (what has been done many times in the past, e.g. [3]). In addition, it can continuously monitor the P2P network - efficiently in terms of consumed bandwidth and processing ressources. This was the reason for adopting the application level crawling technique. Compared to passive P2P traffic identification techniques [2], it is much more efficient and reliable.

However, in order to be less dependent on a specific P2P system, the crawling technique relies on a very small subset of the eDonkey protocol, which is required for adequate content location by P2P users anyway. This subset (at least in similar form) is currently present in most of the popular filesharing systems nowadays, so the suggested architecture would be applicable to them too, with small modifications.

In this paper we are only interested in the content names and properties, but are not trying to identify IPs providing this content. Storing these data would be a legal violation. However, having the names and hashIDs of the popular content, it would be possible to find also the providers of that content because of the MSDP properties, see Section 2.

For this architecture to work, four important problems need to be solved. First of all a mechanism has to be implemented, which can issue queries that deliver popular content names as a result. Unpopular content is not of interest, it only produces very small portion of the P2P traffic, but according to studies like [6] it represents the big majority of the total content shared. Consequently, focusing on the popular content would allow a crawling system to find the biggest traffic producers fast. Second, the blacklisting problem (see Section 2) has to be eliminated, since it limits the throughput of the crawling system. Third, it has to be ensured, that each query returns a number of results which is close to the upper result limit, to reduce the overhead caused by a query and again increase the throughput. Last but not least, the amount of duplicate results has to be kept small.

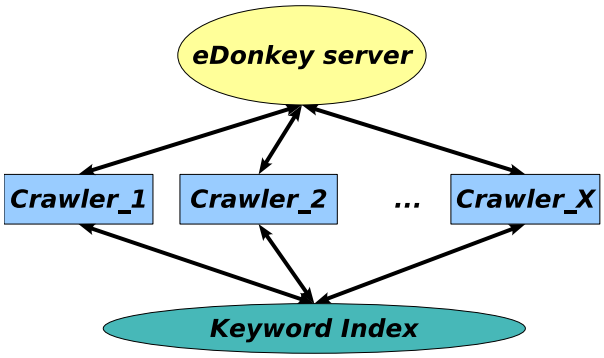

Figure 1: Crawling architecture.

The architecture suggested here is shown in Figure 1. A set of Crawlers connected to a KeywordIndex is used to scan one eDonkey server. The Crawlers are modified eDonkey clients, which query an eDonkey server in parallel. A crawler sends queries with low frequency to avoid blacklisting. But by parallelizing the crawling process, the throughput of the whole system can be increased. Note that a crawler does not necessary need to run on a separate machine, it just requires a separate IP address. The frequency is adaptable - at the beginning all crawlers start at maximum frequency. If a crawler is blacklisted, it immediately informs all other crawlers, which reduce their frequences by a given amount. In the scans run for this paper the frequency was halved each time a crawler has been blacklisted, which led to satisfactory estimation of the optimal frequency - about one

query per 30 seconds. However more complex and precise adaptation algorithms may be used.

All the crawlers are coordinated by the KeywordIndex, which provides the keywords to query for. It also gathers the results collected by the crawlers and feeds them back in the query process, as described in Section 4.

A set of crawlers connected to a KeywordIndex can be used to scan one particular eDonkey server. For scaning more than one eDonkey server, multiple independent Crawler groups with a separate KeywordIndex can be used. Since Crawlers do not store state information except for a single query, a Crawler could change its KeywordIndex if necessary. This is especially usefull in case of blacklisting, when the Crawler needs to connect to a different eDonkey server and query further.

4 The Query Process

A scan of an eDonkey server consists of many keyword query processes which are executed in parallel by the crawlers. When a crawler starts a keyword query process, it first contacts the KeywordIndex and requests a keyword to query. After receiving the keyword, the crawler sends an eDonkey query to the eDonkey server. It then passes results returned by the server back to the KeywordIndex and starts a new keyword query process.

The KeywordIndex parses each content name in the query results received by a crawler and extracts the keywords contained in the name. A simple extraction scheme is used where as delimiters all non-alphanumeric characters are used. The extracted keywords are stored in an index sorted by their frequency. When a crawler starts a new keyword query process and requests a new keyword from the KeywordIndex the KeywordIndex answers with the keyword, which has been most frequently found in content names but was not yet queried. The storage index at the KeywordIndex assures, that a keyword is queried only once at one eDonkey server.

This keyword feedback mechanism utilized by the KeywordIndex tries to follow the naming behaviour of the users connected to the eDonkey server and does not generate artificial keywords like in [3]. The decision which keyword to query next is motivated by the intuitive assumption, that quering popular keywords would also lead to popular content. The measurements presented in Section 6 strongly indicate that the assumption is correct. The feedback mechanism also relies on the assumption that there are no valuable groups of content names with disjunct keyword sets, so that the feedback mechanism could stuck into one of these groups. The gathered results argue for this assumption.

Another issue which is important for the query performance as stated in Section 1 is the result limit imposed by an eDonkey server. Since the result limit may be different for every server, it has to be adaptively determined. A Crawler assumes for the result limit the highest result size it has received from the eDonkey server. If the result limit for a query was reached, the crawler has to ensure, that there are no results which the server did not return. E.g., if a crawler queries the keyword "mp3" it will receive only up to 300 results (if 300 is the result limit). So there will be probably thousands of results not received by the crawler. The way to get to these results is to do a more precise query. One strategy is to use keyword combinations, but it does not give guarantees, that the whole result subspace will be covered. The eDonkey network however provides a convinient set of query options. E.g., one can query all files containing a set of keywords with filesizes between a given upper and lower bound. The filesize bounds allow a very fine-grained division of the result subspace. The architecture uses a binary search technique defined by Algorithm 1 to retreive all results matched by a query with high probability.

Algorithm 1: query(keyword A, lower bound L, upper bound U)

Input : keyword A, lower bound L, upper bound U
Output: result set R
Data : temporary result set R_t
estimated server result limit Li

begin
  R_t = query A at eDonkey server **if** $size(R_t) >= Li$ *and* $U - L > 2$ **then**
    Y = query(A, U, $L/2$) $\cup$ query(A, $L/2$, L)
  else
    $Y = R_t$
end

5 Crawling Strategy

A possible observation strategy would involve two processes, continuously done in parallel. The first process would collect the popular content names and their attributes, as described in Section 3. The second one would focus on the most popular content (about 1 percent) and would query the IPs of the providing peers recursively by using the MSDP features.

The first process provides more general information about the content currently shared in the network. It can perform global scale observations and can up to certain extent measure the content dynamics in the network - the so called *content churn*.

The second process can be seen as a lens - after the content of interest has been identified by the first pro-

cess, the second process can zoom into and provide detailed information of who shares it, where and when, and also with what intensity.

6 Measurement Results and Evaluation

The crawling architecture presented in Section 3 was implemented and started in the PlanetLab network environment [1]. PlanetLab was used because of the IP variety necessary to increase the throughput of the system. During the one week of the experiment one of the bigger eDonkey servers (with about $20 \cdot 10^6$ files in average, changing daily) was scanned. The experiment started with 30 crawlers, but during the one week some of the crawlers disconnected, since their PlanetLab nodes were rebooted or went offline. Only about 20 nodes finished the experiment.

The first important question is what is the netto speed of the crawling process. Netto means how much files after duplicate elimination are found per unit of time. Figure 2 shows the netto learning rate of one particular crawler, which was continuously online throughout the experiment. This is more representative, than the learning rate of the whole system, since the latter is influenced by node feilures in PlanetLab. It can be observed that the netto learning rate decreases with time, since the crawler searches for less popular keywords. Also the chance to find a duplicate increases proportionally to the number of already gathered files.

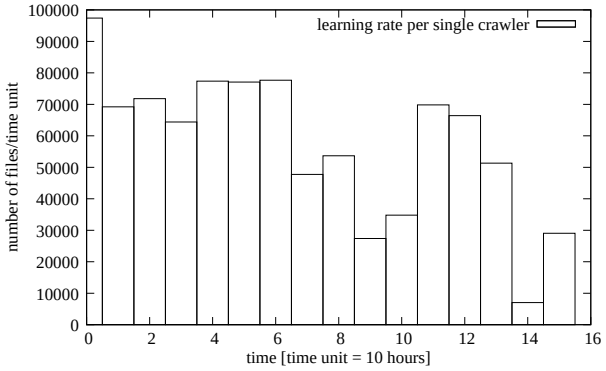

Figure 2: Learning rate for one crawler.

The second important property of the crawling system is its ability to find important (popular) content first. This is illustrated by Figure 3, where the cumulated availability values per 10^6 files is shown in chronological order. Figure 3 also shows the availability graph after excluding the completely unpopular files (having availability of 1). The graph without unpopular files is clearly exponential, proving the efectiveness of the crawling strategy. It is also interesting that the learning rate for completely unpopular files seems to be constant.

In average a file name consists of about 5.4 keywords.

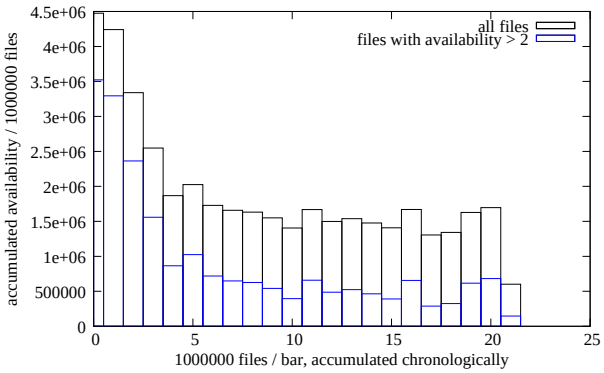

Figure 3: Availability / Time Diagram.

This is also roughly the ratio between brutto and netto learning rate and defines the lookup overhead of the system. Queries returning duplicates however have also an advantage: a file has more than one chance to be found. This is important, since the system is dynamic and users constantly go offline and online.

References

[1] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[2] I. Dedinski, H. DeMeer, L. Han, L. Mathy, D. Pezaros, J. Sventek, and Z. Xiaoying. Cross-layer peer-to-peer traffic identification and optimization based on active networking. In *Proceedings of the 7th International Working Conference on Active and Programmable Networks*. N/A, 2005.

[3] F. Fessant, S. Handurukande, A. Kermarrec, and L. Massoulie. Clustering in peer-to-peer file sharing workloads. 2004.

[4] P. Gummadi, S. Saroiu, and S. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. 2003.

[5] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Hamra, and L. Garces-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. 2004.

[6] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the kazaa network, 2003.

[7] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, pages 99–100, 2001.