Fundamenta Informaticae 109 (2011) 383–403
DOI 10.3233/FI-2011-518
IOS Press

eDonkey & eMule's Kad: Measurements & Attacks

Thomas Locher
IBM Research, Zurich, Switzerland
thl@zurich.ibm.com

Stefan Schmid*
T-Labs & TU Berlin, Berlin, Germany
stefan@net.t-labs.tu-berlin.de

Roger Wattenhofer
ETH Zurich, Zurich, Switzerland
wattenhofer@ethz.ch

Abstract. This article reports on the results of our measurement study of the Kad network. Although several fully decentralized peer-to-peer systems have been proposed in the literature, most existing systems still employ a centralized architecture. The Kad network is a notable exception. Since the demise of the Overnet network, the Kad network has become the most popular peer-to-peer system based on a distributed hash table. It is likely that its user base will continue to grow in numbers over the next few years due to the system's scalability and reliability.

The contribution of the article is twofold. First, we compare the two networks accessed by eMule: the centralized paradigm of the eDonkey network and the structured, distributed approach pursued by the Kad network. We re-engineer the eDonkey server software and integrate two modified servers into the eDonkey network in order to monitor traffic. Additionally, we implement a Kad client exploiting a design weakness to spy on the traffic at arbitrary locations in the ID space. The collected data provides insights into the spacial and temporal distributions of the peers' activity. Moreover, it allows us to study the searched content. The article also discusses problems related to the collection of such data sets and investigates techniques to verify the representativeness of the measured data.

Second, this article shows that today's Kad network can be attacked in several ways. Our simple attacks could be used either to hamper the correct functioning of the network itself, to censor content, or to harm other entities in the Internet not participating in the Kad network, such as ordinary web

*Address for correspondence: Ernst Reuter Platz 7 (TEL 16), D - 10587 Berlin, Germany

servers. While there are heuristics to improve the robustness of Kad, we believe that the attacks cannot be thwarted easily in a fully decentralized peer-to-peer system, i.e., without some kind of a centralized certification and verification authority. This result may be relevant in the context of the current debate on the design of a clean-slate network architecture for the Internet which is based on concepts known from the peer-to-peer paradigm.

Keywords: Peer-to-Peer, Robustness, Measurements, Distributed Systems, eMule, Kademlia, Future Internet Architecture

1. Introduction

Today's peer-to-peer (p2p) networks come in different flavors. On the one hand, there are completely decentralized systems such as the *Kad* network which is based on a *distributed hash table* (DHT) where both the task of indexing the content and the content itself is distributed among the peers.[1] Other systems still rely on centralized entities, e.g., a cluster of servers takes care of the data indexing in the *eDonkey* network, or so-called trackers organize the peers in *BitTorrent* swarms. A server-based solution has the advantage that it is easier to implement and that it works reliably as long as the servers function correctly. Clearly, the downside of this approach is that the servers can only sustain a certain number of peers, implying that the scalability is limited and that an overload of concurrent requests can cause a system failure. Purely decentralized systems do not depend on the availability of any particular entity; however, such systems often demand larger contributions from all participants.

This article examines popular representatives of the two network types: the server-based *eDonkey* and the decentralized *Kad network*. eDonkey is one of the largest p2p networks in use today; millions of users around the planet use it to share various types of multimedia contents. While there are other clients to gain access to the eDonkey network, by far the most popular client is *eMule*.[2] Additionally, eMule allows its users to connect to the *Kad network*. This network, which is based on *Kademlia* [16], is currently the most popular distributed hash table (apart from the *Mainline DHT* and the *Azureus DHT* which are used by BitTorrent as trackers for peer discovery).

In order to investigate various properties of eDonkey and Kad, we collected large amounts of data from both networks (mostly in 2007). For this purpose, we reverse-engineered the eDonkey server software and published two own servers which successfully attracted a considerable amount of traffic despite the fact that our servers never returned any real content. For our Kad tests, we implemented a client that is capable of spying on the traffic at any desired position in the ID space. Section 3 describes the setup of our measurement infrastructure.

Section 4 reports on our measurement results. We are particularly interested in the user behavior in both networks. In this article, in contrast to other literature, we monitor the actual user requests and ignore automated requests which occur without any user intervention. Our measurements show that the temporal request distributions of the two networks are very similar, exhibiting a high activity in the early evening with high loads at the eDonkey servers or at the peers hosting popular files in Kad. We also found that both networks are predominantly used in European countries, but there are also many active users from Israel, China, Brazil, and the United States. Section 4 also investigates the content shared in

[1] Unstructured decentralized systems such as *Gnutella* are not considered in this study.
[2] See http://www.emule-project.net/.

the two systems. For example, we find that popular content in the eDonkey world is often also popular in Kad, and that eDonkey follows the popularity trends of the real world. In general, our results indicate that peer activity results in eDonkey directly carry over to the Kad network and vice versa. This observation is not self-evident, given that we analyze only user-generated events. In Section 5, we raise the question of the representativeness of the collected data. In the Kad network, accurate data on the activity of a specific file can be obtained, but due to the distributed nature of the DHT, it is inherently difficult to compute global aggregates such as the most active file in the network. On the other hand, in the eDonkey network, a server receives queries for virtually all keywords, but it has to compete against other servers for the requests. If only a minor fraction of the traffic arrived at our servers or if the servers to be queried were selected with respect to specific properties such as latency, the data could become biased. We will provide evidence that there is no critical bias in our measurements.

Subsequently, in Section 6 we question whether the p2p approach is mature enough to step outside of its "comfort zone" of file sharing and related applications. In particular, not much is known about the ability of DHTs to meet critical security requirements (as those required nowadays, e.g., for domain name servers) and its ability to withstand attacks. To this end, as a case study, we evaluate the feasibility of various attacks in the Kad network. Our study reveals that while the Kad network functions reliably under normal operation, today's Kad network has several critical vulnerabilities, despite ongoing efforts on the developers' part to prevent fraudulant and destructive use. This article describes several protocol exploits which prevent peers from accessing particular files in the system. In order to obstruct access to specific files, file requests can be hijacked, and subsequently, arbitrary information can be returned instead of the actual data. Alternatively, we show that publishing peers can be overwhelmed with bogus information such that pointers to the original files can no longer be accessed. Moreover, it is even possible to *eclipse* certain peers, i.e., to fill up their routing tables with information about malicious peers, which can subsequently intercept all messages. Additionally, we briefly discuss how our network poisoning attacks can also be used to harm machines outside the Kad network, e.g. web servers, by tricking the peers into performing a Kad-steered distributed denial of service (DDoS) attack. It is virtually impossible to determine the true culprit in this scenario, as the initiating peer does not take part in the attack.

All our attacks have been tested on the real Kad network using a modified C++ eMule client. Already with three attackers, virtually no peer in the system was able to find content associated with any given keyword for several hours, which demonstrates that with moderate computational resources, access to any targeted content can be undermined easily.

2. Related Work

Measurement studies are an important means to gain deeper insights into the working of distributed systems. While theoretic models allow researchers to reason formally about a system's behavior and to prove its properties, such models are often simplifications and may not reflect reality well. For more complex systems, *in silico* experiments are conducted, desirably for as many points in the parameter space as possible. However, although such simulations—and also experiments on PlanetLab [11]—can provide additional confidence in a system's performance, it is not until the real deployment when the system properties become clear.

There exist many measurement results for various p2p systems today. Saroiu et al. [19] have analyzed several characteristics such as the bottleneck bandwidths of the peers participating in Gnutella and Nap-

ster. Adar et al. [1] have investigated the contributions of the Gnutella users. An important algorithmic challenge in p2p computing is understanding churn, and hence traces of membership changes in the systems deployed today [23] have been collected. There is also a community aiming at reverse-engineering closed-source projects such as Skype by studying the traffic patterns [10].

We have decided to study the eDonkey and the Kad networks as they are two of the largest p2p networks in use today, and as there is not much literature on these networks. Interesting results on the *Kad network* have been obtained by Biersack, Steiner, and others in [6, 24, 25, 26]. For instance, in [25], possible misuses of the protocol are discussed. Stutzbach et al. [28] describe implementation details of Kad in eMule, and [27] presents crawling results on the behavior of Kad peers. The most related work to our study of the Kad network is due to Steiner, Biersack and En-Najjary [23]. The authors have crawled the Kad network during several weeks and found, e.g., that different classes of participating peers exist inside the network. In contrast to their work which has studied the churn induced by the peers' joins and leaves, our focus is on the peer *activity* while the peers are online, which we measure by monitoring the lookups. As stated in [23], peer IDs can change frequently, even as often as once per download session while other IDs remain in the network for several weeks. Due to these conditions and the fact that several peers might share the same IP address, it is hard to draw any conclusions about peer behavior when monitoring the peer IDs and the IP addresses in the network. Since keyword lookups are hardly automated, observing lookups is the best and presumably the only way to get insights into the activities of users in such networks. To the best of our knowledge, this is the first peer activity study by means of monitoring lookup requests in distributed networks. It is also the first study to take both server-based and decentralized systems into account.

The immense computational resources of p2p networks are also attractive to attackers, and there is already a large body of literature on the subject [7, 30].[3] Reasons to attack a p2p system can be manifold: For example, a peer may seek to perform a more or less passive "rational attack" [18] to be able to benefit from the system without contributing any resources itself [1, 13]. While such selfishness can threaten a peer-to-peer system, which essentially relies on the participant's contributions, there are more malicious attacks seeking to harm the system directly. An attacker may, for example, strive to partition the system or to eclipse individual nodes. The *eclipse attack* [21], as also described in this work, can be used by a set of malicious peers to position themselves around a given peer in the network such that the peer's contact list consists only of the colluding peers. In a *Sybil attack* [9], a single entity creates multiple entities of itself in order to gain control over a certain fraction of the system. Such an attack can undermine redundancy mechanisms and is hard to counter in a completely decentralized environment. Attackers may also exploit a peer-to-peer system to efficiently spread a *worm* [31]. Furthermore, the resources of a p2p system may also be used to attack *any* machine connected to the Internet regardless of whether it is part of the peer-to-peer network or not. A *denial of service attack* can be launched in various p2p systems, e.g., Gnutella [2], Overnet [17], and BitTorrent [8]. During this attack, information about the victim, i.e., the targeted machine in the attack, is spread in the system. The victim is falsely declared as an owner of popular content, causing other peers searching for this content to contact the victim repeatedly. In BitTorrent, tracker information can be faked which leads peers to believe that the victim is a tracker for the desired content [8]. In the Kad network, DoS attacks can be launched by means of a redirection attack where a queried peer, the attacker, will return a response containing the address of the victim [29]. As mentioned before, the attacks presented in this work can also be used to launch a DoS attack.

[3] See also http://www.prolexic.com/news/20070514-alert.php/.

The work closest in spirit to our work on attacks in p2p networks is the study of *index poisoning attacks* in FastTrack and Overnet [12]. The index poisoning attack in [12] is akin to our publish attack where bogus information is pushed aggressively to the nodes responsible for the desired keywords. However, while this attack is also quite successful, it is not as effective in the Kad network as it is in FastTrack and Overnet. We show that a different, even simpler poisoning attack is feasible and even more effective. Moreover, our study of attacks in the Kad network is not limited to content poisoning and index poisoning, but also considers the eclipse attack to prevent peers from accessing a specific file. It is also worth pointing out that, in comparison to Kad, it is generally easier to perform attacks on Overnet, as it, e.g., does not check whether the sender of a publish message provided its own IP address as the owner of the file, and no cryptography is used for authentication.

While we believe that there are methods to contain the potential damage caused by such attacks to a certain extent, it is known that some sort of logically centralized entity is required to thwart attacks such as the Sybil attack [9]. There also exists literature on the robustness of Kad. For example, Steiner et al. [25] initiated the study of Sybil attacks in Kad, and propose to tie the possibility of obtaining a Kad ID to the possession of a cell phone number. Their solution therefore requires a centralized entity as well. There is also some interesting theoretical work on how to identify and exclude large sets of colluding peers [4]. However, the described techniques cannot be used to counter our attacks as we only need a very small number of attackers close to a given ID, which is not sufficient to raise suspicion. For a more thorough discussion of possible countermeasures against attacks in p2p networks, the reader is referred to the corresponding literature (e.g., [7]).

3. Background and Measurement Framework

The *eMule client* provides access to the classic, server-based eDonkey network and the decentralized Kad network, an implementation of the distributed hash table Kademlia [16]. The different nature of the two networks requires different measurement techniques. In the following, we will first present our approach to collect data in the eDonkey network. Subsequently, we will report on the functionality of our Kad client which allows us to monitor traffic at arbitrary spots in the ID space.

3.1. eDonkey Network

When a user issues a query using the eMule client, the keywords of the query are sent to a subset of servers, which subsequently respond to the client with information about where to obtain the requested file. We found that the peers typically iterate over the list of servers contained in their server file, querying one server after the other as long as less than 300 results have been returned. The order of servers in this list reflects the history of when peers learned about these servers, i.e., old servers are at the top of the list while new servers are appended at the end of the list.

Today, there is a large number of eDonkey servers all over the world, most of which are based on the *lugdunum* software.[4] This software is not open-source as the developers try to prevent the creation of fake servers or any other undesirable modification that could endanger the correct functioning of the lugdunum servers. In order to collect data in the eDonkey network, we reverse-engineered the server software and set up two servers ourselves which operate as follows. Initially, our server imports all

[4]See http://lugdunum2k.free.fr/kiten.html.

known eDonkey servers from a file and announces itself to every server on that list, one after the other. For each server on the list, a *server list request* is sent, followed by a *server status request* and a *server description request*. In return, our server receives a list of servers that are alive, and the current status and description of the corresponding server. As a side effect of these queries, our server is added to the other server's list. This is vital as peers keep their server lists up to date by periodically asking the servers they are connected to for their lists of currently known servers; i.e., once our server appears in these server lists, all peers will quickly learn about the existence of our servers. In order to remain a member of these lists, our servers correctly answer the status requests of other servers. However, due to legal concerns, we neither store nor return any real data. Moreover, we pretend having a high number of users and shared files, but we deny any login requests and reply with a message indicating that our server is full.

Due to the iterative lookup procedure described before, our servers are contacted perpetually, regardless of which servers the peers are connected to. As a result, we can collect large amounts of data about many different kinds of requests, making it possible to compute global aggregates such as the most popular keyword in the network, or the most active peer's IP address. Naturally, this data is only representative if we receive a substantial fraction of all requests in the network. This issue is discussed in more detail in Section 4.

3.2. Kad Network

In the Kad network, both the files and the index is stored in a distributed manner by the peers themselves; there are no indexing servers. Each peer in the Kad network has a 128-bit identifier (ID) which is normally created by a random generator. This ID is stored at the peer even after it has left the network and is re-used once the peer returns. Routing in the network is performed using these identifiers and the XOR metric, which defines the distance between two identifiers as the bitwise exclusive or (XOR) of these identifiers interpreted as an integer. For all $i \in [0, 127]$, every peer stores the addresses of a few other peers whose distance to its own ID is between 2^i and 2^{i+1}, resulting in a connected network whose diameter is logarithmically bounded in the number of peers. For each of these *contacts* in the routing table, a Kad ID, an IP address, and a port is stored.

The publish and retrieval mechanisms work roughly as follows. Each keyword, i.e., a word in a file name, and the file itself, are hashed, and information about the keywords, its associated file, and the address of the owner is published in the network, i.e., this information is stored at the peers in the DHT whose identifers are closest to the respective hash values. More specifically, in Kad, information is replicated ten times in a zone where peers agree in the first 8 bits with the published key. Usually, this so-called *tolerance zone* contains several thousand peers. While most of the peers are very close to the key, this is not always the case, e.g., due to churn and also for keys that are very popular and published by many different peers.

In order to find a file for a given keyword k, a peer computes a hash function $h(k)$ of k and routes, in a multi-hop manner, the request to the peer having the overlay ID closest to $h(k)$; this peer stores the hash codes of all the files associated with this keyword. The matching filenames and the corresponding hash codes of these files are then returned. Given a hash code $h(f)$ of a file f, it is then possible to get a list of all the peers possessing a copy of f by again routing to the peer whose ID is closest to $h(f)$ as this peer is responsible for the sources of f.

We created our own Kad client in order to collect data on the peer activity in the Kad network. Our client exploits the fact that Kad uses randomly chosen overlay IDs, which enables us to place our peers

at any desired place in the ID space. On the one hand, performing measurements in the Kad network is simpler than in the eDonkey network. This is due to the fact that a small set of peers close to the hash of a file f will be contacted by all peers interested in obtaining this file f. Thus, as there is a unique location where peers obtain information about f, data of good quality can be collected by occupying the corresponding area around this ID and spying on the traffic. On the other hand, the distributed nature of the Kad network renders it more difficult to measure global quantities such as the most popular file in the network. Answering such a query would require to occupy a large portion of the entire ID space. Hence, we confine ourself to acquiring small samples of the entire traffic and try to juxtapose these samples and the data acquired in the eDonkey network in a reasonable manner.

4. Measurements

This section summarizes our main measurement results. We investigated the distribution of the user base across countries of both eDonkey and Kad as well as the temporal distribution of the users' requests. In addition, the content that users search in the system is examined. Our measurements were conducted mostly in 2007.

4.1. Request Distributions

Within a few days after announcing our servers, they attracted much traffic. Figure 1 shows the activity of our servers during 4 days. We see that the request pattern remains fairly stable across all days. On average, during a measurement period of 2 weeks, our servers received roughly 1,550 login requests, 448 keyword requests and 150,228 source requests per minute. The average bandwidth required to run each server is approximately 300 KB/s. Note that a correct server requires substantially more bandwidth as it has to reply to all keyword and source requests. Due to the additional traffic caused by re-announcing

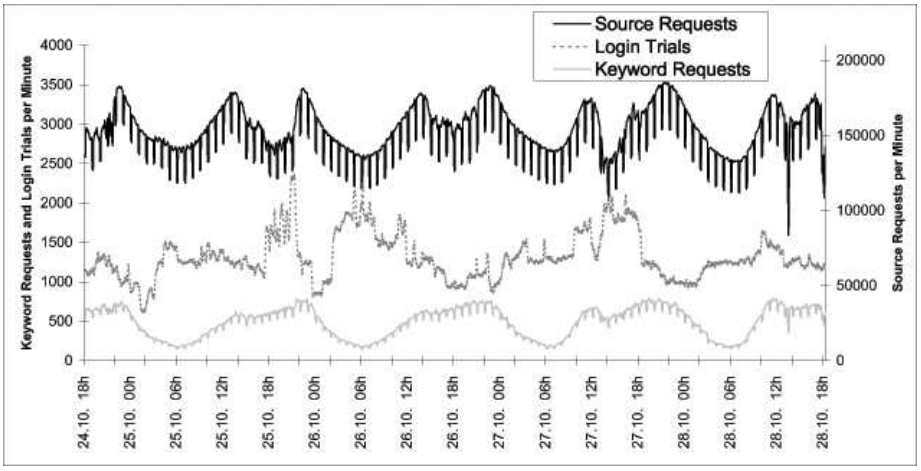

Figure 1. Different server requests over time. The y-axis for the source requests is shown on the right, for the login trials and the keyword requests it is shown on the left.

our servers at other servers once per hour, our servers are overloaded for a short time resulting in regular drops of handled requests, which is most apparent in the curve of the recorded source requests.

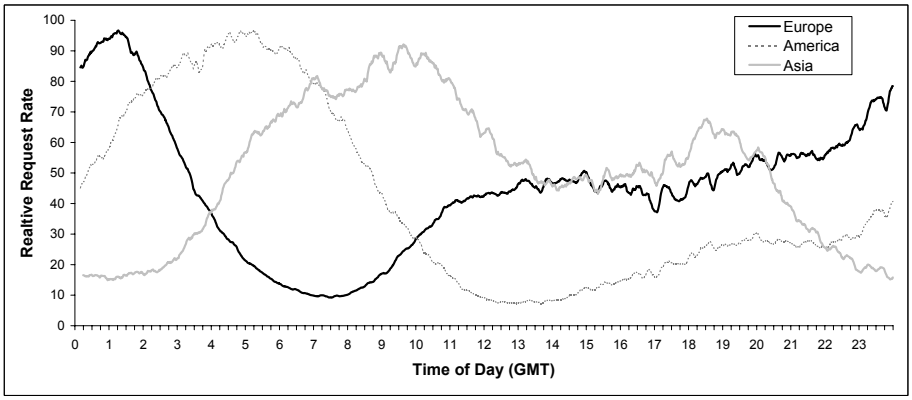

Figure 2. Temporal distribution of keyword search requests on an average day on eDonkey, grouped by continents. The time on the x-axis is based on the Greenwich Mean Time.

The keyword searches are particularly interesting to study, as they are entered by users directly and are hardly automated. Consequently, the amount of search requests varies over the day. Figure 2 shows this distribution for different continents. The figure reveals that in Europe and America the minimum number of requests is reached in the early morning and this number continuously increases until midday, where it stays on a more or less constant level during the whole afternoon. Then it increases again after the working hours until the maximum is reached at around midnight. The curve for Asia looks slightly different; the maximum is also reached at midnight, but there is not such a sharp decline during the night, and the number of requests even increases again reaching a second local maximum in the early morning. Note that the maximum number of requests is set to 100% for each continent in order to show this diurnal pattern. The total number of requests per day in Europe, America, Asia, and Africa plus Middle East are 397,060, 156,322, 42,287, and 48,850, respectively, which necessitates this normalization and also demonstrates the predominance of Europe in the eDonkey network.

As one might expect, the distribution of the search requests in the Kad network is similar. Figure 3 depicts the temporal distribution of requests again for the three continents in the Kad network. Again, the curve for Asia is quite different from the others. As opposed to the other continents, the maximum number of requests in Asia is reached in the morning and not late in the evening. We occupied 14 randomly chosen IDs and logged all requests on these peers and used the average number of requests in this figure.

We can look at the origins of the requests in more detail and observe that European countries play an important role in eDonkey, the only country among the five most active countries outside of Europe is Brazil. Figure 4 depicts the percentage of all requests originating from each of the 20 most active countries per month, both for the eDonkey and the Kad network in descending order of activity in the eDonkey network. A first observation that can be made is that the spacial distribution is more concentrated in Kad than in eDonkey. Moreover, it can be seen that the same countries are the most active ones in both networks. Note that, although eMule grants access to both networks, users have to enter *manually*

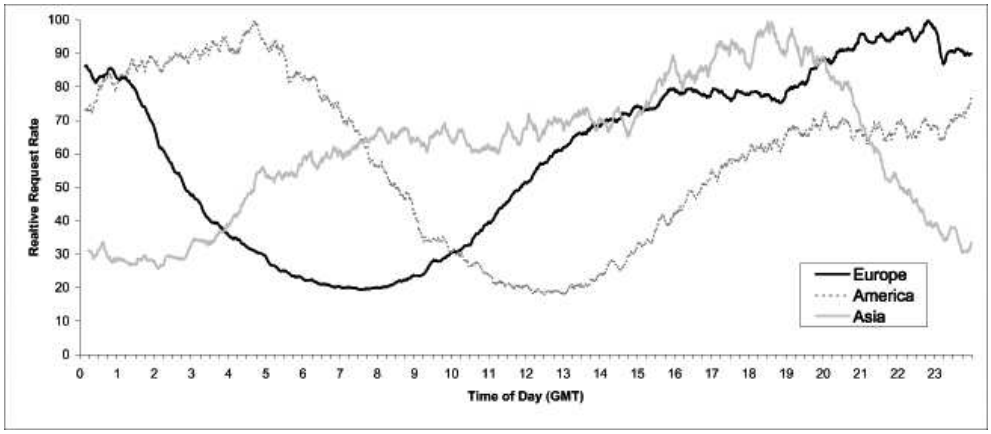

Figure 3. Temporal distribution of keyword search requests on an average day on Kad, grouped by continents. 14 monitoring peers in Kad are used to compute these numbers. The time on the x-axis is again based on the Greenwich Mean Time.

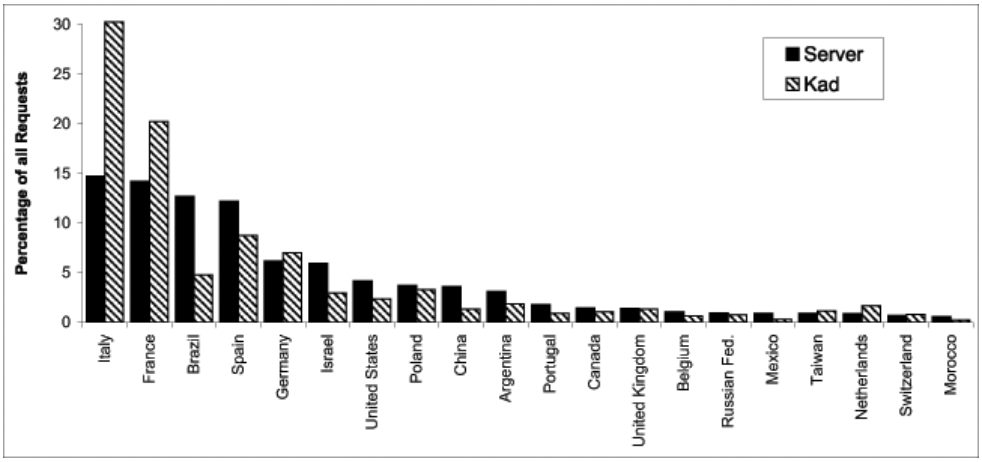

Figure 4. Origins of keyword search requests on our servers and in the Kad network.

where they want to search and thus this result is not self-evident. Furthermore, the Kad network seems to be significantly more used in Europe, especially in Italy and France, than elsewhere. The question whether this is due to a more lenient legislation remains open.

It is difficult to assess the popularity of these networks by comparing the absolute number of requests, as there are countries with a much larger population or a higher Internet penetration rate. For this reason, we have normalized the request rates received from each country by the number of Internet users in that country.[5] As can be seen in Figure 5, the picture looks different in the normalized case. There are three quite active countries, Morocco, Algeria, and Israel, while all other countries have a comparably small

[5]Data obtained from http://www.internetworldstats.com.

number of requests per Internet user per month. The reason for this exceedingly high number of request originating from Morocco and Algeria might be simply due to the small number of Internet users in these countries. Another possible reason is that relay servers are positioned in these countries in order to obfuscate network traffic. The observation that a large number of requests originate from a small number of IP addresses supports this claim. As there are many different IP addresses active in Israel and given that it is generally one of the most active countries, it seems that these networks are simply highly popular in Israel, even more so than in Europe. As far as the other countries are concerned, the graph shows that there is not a significant difference between the popularity of eDonkey and Kad among them. What is more, the distribution for both networks has a long tail; as many as 21 countries exhibit a normalized search activity of at least 20% of the search activity of Spain, implying that both networks are popular in many countries. We further found that both networks are indeed much more popular in Europe than in the United States, the activity of the United States normalized by the number of Internet users is about 30 times smaller than the activity of Spain, making it the country with almost the smallest activity overall. Clearly, this is partly due to the large number of Internet users in the United States. Overall, only six countries contribute more keyword searches than the United States, which indicates that also in the United States both networks have a large user base. Finally, however, note that the data in Figure 5 could also be slightly biased, as the Internet penetration data might not be perfectly accurate.

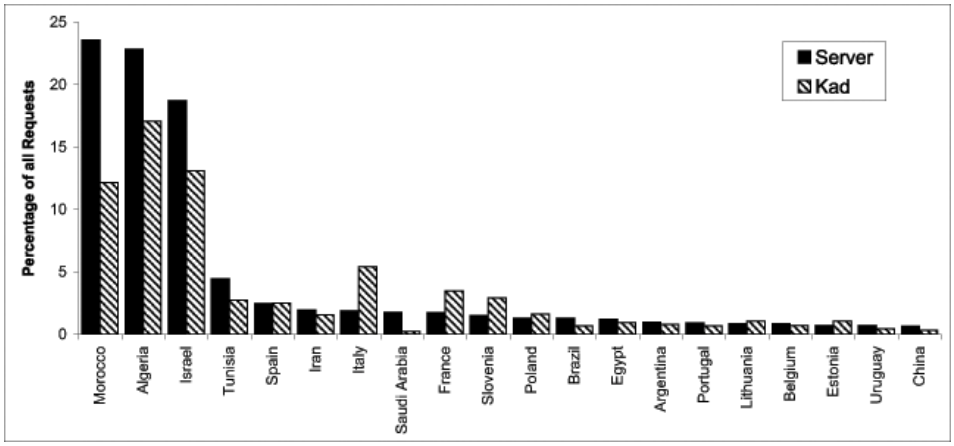

Figure 5. Keyword search requests normalized by the number of Internet users of the 20 most active countries on our servers and in the Kad network.

4.2. Search Contents

The main objective of both the eDonkey and the Kad system is to provide users with a mechanism to find and download files. Information about the searched content can be an interesting source for research, for example, such data might give insights into the potentially different preferences of users in different countries.

For this purpose, a record indicating the popularity of each data item in each country would be required. Unfortunately, the compilation of such a record is quite difficult—not only in Kad, but also in the eDonkey network. One reason is that there is no automatic one-to-one correspondence between

keywords and files. There might be different spellings of the same keywords, files containing the same content are typically available in different languages, and the corresponding filenames often contain typing errors. Moreover, the popularity of the files we monitor in Kad can change quickly, particularly when versions of the same content, e.g., a video file, of increased quality appear. Figure 6 plots different versions found when quering for a specific exemplary keyword during a period of 50 days. Version v_1 is the worst quality, v_2 is the same content in better quality, and v_3 has the best quality. As expected, the number of occurrences of v_1 decreases over time, first at the expense of v_2, and after v_3 becomes more and more popular, the number of occurrences of v_2 start decreasing as well.

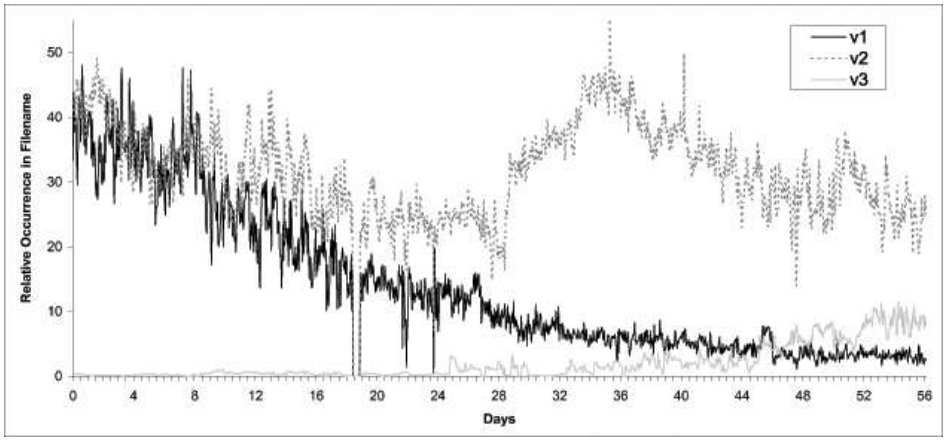

Figure 6. Different quality versions, distinguished by specific keywords in the filename, in percentages of all files.

In another experiment, we tried to evaluate to what extent the popularity of certain content in eDonkey and Kad corresponds to the popularity of the same content in the real world. To this end, we observed the popularity of newly released movies in eDonkey and Kad. We find that there is indeed a strong correlation, i.e., movies that are currently playing in movie theaters are popular both in eDonkey and Kad. Figure 7 shows this correlation for a specific movie (namely "Superbad"). In this figure, the total

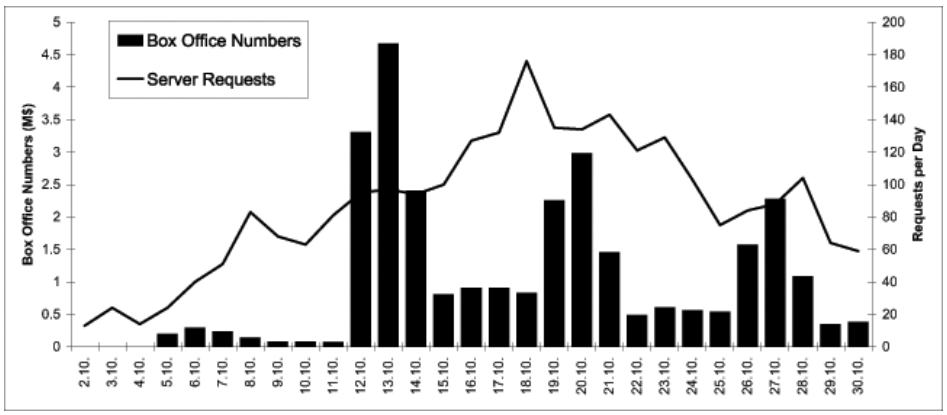

Figure 7. Comparison of the box office gross and the requests on our servers for a specific movie ("Superbad").

gross[6] in the U.S. is depicted for each day and also the number of requests for this movie on our servers. The movie opened on October 5 2007, but it did not attract many movie-goers until the next weekend. Afterwards, the daily gross declined again with smaller peaks at the weekends as usual. In this graph, we see that the popularity in eDonkey roughly follows these trends. Observe that the request pattern in the network is delayed for about a week, reaching its maximum about a week after the movie reached its peak. Experiments using other content yielded more or less the same graph, also with a certain delay. In order to take the Kad network into account, we further compared how often keywords are looked up in eDonkey and in Kad and found that basically the same keywords are looked up more often than others in both networks.

A thorough discussion of content popularity is beyond the scope of this article. Our preliminary findings indicate that there is not only a strong correlation between eDonkey and Kad, but also between the two networks and the popularity of content in the real world.

5. Representativeness

Conducting measurement studies of distributed systems is a difficult endeavor. Even if large amounts of data are collected, the statistical significance of the empirical results might be limited if the data is biased. In order to obtain solid claims, it is important that the underlying data be either complete, or a uniform and random subset thereof. In this section, we provide evidence that our data can be considered representative.

We consider the data collected by the servers first. As mentioned before, the servers receive requests for all possible keywords. However, since a peer does not send requests to all the servers in its server list, i.e., some servers might receive completely different requests, which could potentially bias the collected data. As the eMule clients typically send *source requests* to both networks, in order to estimate what fraction of all requests we receive, we compared the number of source requests at our eDonkey server with the number of source requests obtained in Kad. Our experiments showed that for a given file, we receive roughly 10 times more requests in Kad than at the server. Since virtually all requests for a given file are received in Kad, this indicates that our server roughly receives 10% of all keyword requests in the network—a surprisingly large number. At the same time, the distribution of the origins of the requests does not differ between the two networks. This suggests that they are already contacted with a reasonably large probability, although our servers are relatively new, and also that they get a more or less random subset of the entire traffic.

In the Kad network, it is easy to obtain unbiased request data for a given file, since all requests for a particular file are routed to the same ID. However, making statements about the global distributions of the requests requires to collect data at all locations in the ID space, which is impossible. In this article, we have taken a best-effort approach and aimed at getting data from a moderately large set of peers whose IDs are distributed uniformly at random. By averaging these measurements, we get similar distributions as those measured in eDonkey, which indicates that the obtained data is fairly representative. Although we believe that the quality of our results is quite good, it has to be taken into account that, similarly to our client, other peers can also choose their overlay IDs at will, which could bias such a random sampling approach. It is known that there are communities that select their Kad IDs from a small subset of the entire ID space [23].

[6]Data obtained from http://www.boxofficemojo.com.

6. Kad Attacks

We now turn our attention to the robustness of Kad, and report on three different attacks that limit the peers' access to a given file f. In a *node insertion attack*, an attacking peer seeks to attract search requests for f, which are answered with bogus information. Alternatively, access to f can be denied by filling up the index tables of other peers publishing information about f (*publish attack*). Finally, we describe how an attacker can *eclipse* an arbitrary peer: By controlling all the peer's incoming and outgoing traffic, the attacker can prevent a peer from either publishing information about f or from accessing it.

6.1. Node Insertion Attack

By performing a *node insertion attack*, it is possible to corrupt the network by spreading polluted information, e.g., about the list of sources, keywords, or comments. We have implemented the attacks for *keywords*, that is, a search for the attacked keyword will not give the correct results, but instead arbitrary data chosen by the attacker is returned.

For this attack to work, we have to ensure that the search requests for the specific keyword are routed to the attacking peer rather than to the peers storing the original information. This can be achieved as follows. In our modified eMule client, it is possible to select the peer's Kad ID manually. Thus, an attacker can choose its ID such that it matches the hash value of the targeted keyword. Consequently, the peer will become the node closest to this ID and will receive all the corresponding search requests. The nodes storing the correct files typically have a larger distance to the keyword's ID than the attacker, as the probability for a peer to have a random ID that perfectly matches the 128-bit keyword ID is negligible.

In order to guarantee that peers looking for a certain keyword only receive faked results, the attacker must provide enough result tuples, as the eMule client terminates the search after having gathered 300 tuples. The attacker further has to include the keywords received from a peer in the filenames, otherwise the replies are not accepted. In our attacks, we use filenames that contain a unique number, the message "File removed from Kad!", and the keywords. Unique file hashes are needed such that the 300 tuples are not displayed as one tuple in eMule's search window.

We frequently observed that eMule sends search requests not only to the closest peer, even though this peer provided enough answers. This can be explained by the delay caused when transmitting the 300 search results from the closest peer. eMule will send another request to the second closest peer before all of the results are received from the closest one. This of course may harm the effectiveness of the attack, and hence it is beneficial to gain control over the second, third, etc. closest IDs as well by means of additional attackers. These attackers behave exactly the same way: All requests are answered by supplying 300 faked tuples.

Figure 8 depicts the traces obtained during two week-long node insertion attacks performed using our modified eMule client on the keyword "Simpsons." Note that this attack influences all queries in the entire Kad network not only for the search term "Simpsons", but also all other queries starting with the term "Simpsons" such as "Simpsons Movie" or "Simpsons Soundtrack" etc. are affected automatically.

In the first trace, only one attacker whose ID exactly matches the hash of the keyword infiltrated the network. We used another client to search for the term "Simpsons" once a minute and examined the returned results. Since a single attacker is not sufficient, as mentioned before, the attack is moderately successful in that only approximately 40% of the returned results originated from the attacker. What is more, every single query returned at least some results that are not faked. Further experiments showed

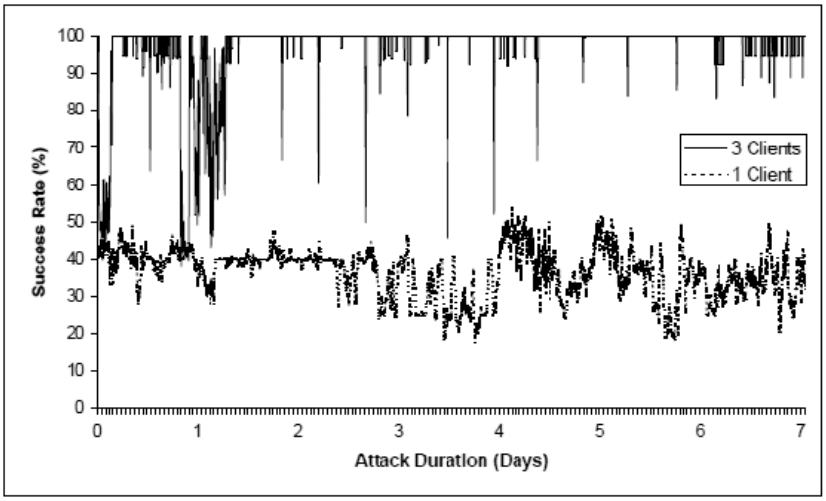

Figure 8. Percentage of successfully hijacked keyword requests in a node insertion attack for 1 and 3 attackers during a time period of one week.

that using two attackers instead of one does not increase the success rate substantially, but three attackers is already enough to hijack virtually all requests. The second trace shows the success rate of the node insertion attack using three attackers. On average, more than 95% of all returned tuples were faked, and every batch of tuples contained at least some bogus data created by the attackers. The plot shows that there are sudden drops of the success rate once in a while. An explanation for this behavior is that peers join and leave the network at a high rate, resulting in inaccurate routing tables. Consequently, a lookup request can be routed to a peer that still stores results for this request and does not know about our attacking peers yet.

The attack was repeated at other times using different keywords. All our other experiment resulted in a similar picture and confirmed our findings made with the "Simpsons" keyword. Our attacking peers received roughly 8 requests per minute from other peers in the network during the experiments. As expected, the peer having the closest ID received the most requests at a rate of roughly 4 requests per minute.

6.2. Publish Attack

In contrast to the node insertion attack, which forces the search requests to be routed to the attacker, the publish attack directly attacks the peers closest to the ID of the attacked keyword, comment, or source entry. The index tables stored by the peers in the Kad network have a limited length; for instance, the keyword table can store up to 50,000 entries for a specific ID. Moreover, a peer will never return more than 300 result tuples per request, giving priority to the latest additions to the index table. This makes it possible to replace the original information by filling up the tables of the corresponding peers with poisoned entries. Thus, an attacker seeks to publish a large amount of information on these peers. Once the index tables of the attacked peers are full, they will not accept any publish requests by other peers anymore. Therefore, the attacked peers will only return our poisoned entries instead of the original

information. Since every entry has an expiration time (24 hours for keyword and comment entries, and 5 hours for source entries), the clients have to be re-attacked periodically in order to achieve a constant fraction of poisoned entries. In addition, an attacker has to take into consideration the newly joining peers in the network; if they have an ID close to the one attacked, their tables also have to be filled.

We have implemented the publish attack for keyword entries as well, again by modifying the original eMule application. An existing timer method is used to run the attack every 10 minutes. In the first phase, the 12 peers closest to the targeted ID are located using eMule's search mechanism. In each run, only peers are selected that have not been attacked before or which need to be re-attacked due to the expiration of the poisoned entries. In the second phase, all the peers found in the first phase are attacked, beginning with the closest peer found. To guarantee a full cache list, 50,000 poisoned entries are sent divided into 250 packets containing 200 entries each. In order to prevent overloading the attacked client, the sending rate was limited to 5 packets per second. Every entry consists of a unique hash value and filename as in the node insertion attack. Since these entries ought to match all search requests containing the attacked keyword, it is necessary to include all additional relevant keywords (e.g. song titles for an interpreter, year and language for a film title) in the filename; otherwise, all the lookups with additional keywords would not receive the poisoned entries, because not all the keywords are included. In the node insertion attack, this problem does not occur as the additional keywords are obtained from every search request and can directly be appended to the filename to match the request. The success of each run is measured with the load value sent in every response to a publish packet. This value should increase with every poisoned packet sent, from a starting level of about 10 - 20% to 100% when the attack is finished.

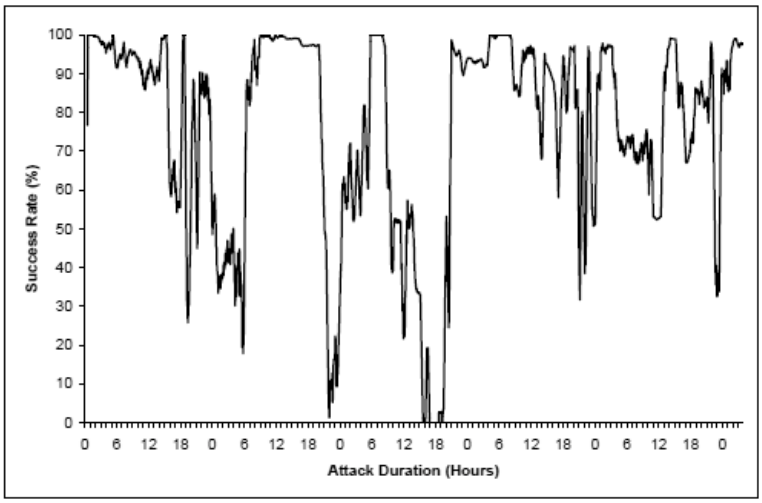

Figure 9. Percentage of faked replies received in a publish attack for the keyword "Simpsons" during a time period of 5 days. Sometimes, the success rate drops but then recovers again quickly.

In comparison to the node insertion attack, it is clearly harder to maintain a high success rate using the publish attack, due to the permanent arrivals of new peers and the need to re-attack the peers periodically. While the node insertion attack yields constantly high rates, this is not true for the publish attack. Figure 9 plots the success rate of an attack on the keyword "Simpsons" over a period of 5 days. While the attack

works fairly well on average, at a success rate of roughly 80%, the success rate periodically drops and remains low for a certain time period before it recovers again.

Overall, the success rate is lower than in the case of a node insertion attack, although performing a publish attack is more expensive. Again, repeating the attack at other times using different keywords results in a similar pattern. The reason for this peculiar behavior is that the peers responsible for the targeted IDs that are online during the phase where the success rate is low refuse to accept our publish messages. In fact, these peers do not even reply to publish messages, even though they can be contacted, otherwise we could not receive any lookup results from them. As this behavior is not in accord with the protocol implemented in the real eMule client, we suspect that modified versions of the original client cause this effect. What clients are used is hard to determine as they do not directly provide this information. Thus, the use of modified clients appears to be another reason why the node insertion attack is superior to the publish attack. In order to improve the success rate, a more sophisticated implementation could be used where the attack is split up into two concurrent processes. The first one would permanently search for new peers with an ID close to the one attacked and pass them to the second process which then would attack these peers simultaneously. This would minimize the time during which peers can respond with original data. As this improvement would not solve the problem of uncooperative peers, it was not implemented.

6.3. Eclipse Attack

Instead of poisoning the network to keep peers from obtaining certain information, we can also attack the requesting peers directly and keep them from sending requests into the Kad network. In the eclipse attack, the attacker takes over the targeted peer's routing table such that it is unable to communicate with any other peer in the Kad network except the attacker. As the attacker simulates the whole Kad network for that peer, it can manipulate the attacked peer in arbitrary ways, e.g., it can specify what results are returned for any lookup, or modify comments for any file. The peer's requests can also be directed back into the Kad network, but modified arbitrarily.

Typically, the contacts in the Kad routing table are not uniformly distributed over the whole ID space. Rather, most of the contacts are located around the peer's ID to maintain short lookup paths when searching for other peers in the Kad network (see also [16]). The attacker takes advantage of the fact that there are relatively few contacts in most parts of the ID space. Concretely, we inject faked peer entries into these parts of the routing table to achieve a dominating position. Subsequently, the faked peers are selected for almost all requests. If we set the IP address of all those faked entries to the address of our attacking peer, we receive most requests of the attacked peer and can process them as desired. We make use of the fact that the standard eMule client accepts multiple neighbors of the same IP address.

Our measurements showed that a peer running eMule for an extended period of time has up to 900 contacts in its routing table. As the maximum number of contacts is 6,310, there is plenty of space in the routing table for faked entries. In order to inject faked entries the *Hello Request* message is used, which is normally utilized during connection set up to check whether known peers are still alive. As a side effect of this message, the sender of the message is added to the receiver's routing table. After enough entries are injected, the attacking peer has to process the requests from all those entries in order to keep them in the routing table of the attacked node.

We implemented the eclipse attack in a stand-alone application and ported all necessary parts from the source code of eMule. The application maintains a list that holds all faked entries sent to the attacked

peer. This is necessary, because every new entry in the routing table is validated by sending a hello request. This request has to be answered with the same ID as we have chosen when injecting the entry. In order to differentiate between the entries, we assign a new port to every faked entry and maintain a data structure to store this information. The other part of our application processes the requests of the attacked peer. If it asks for new peers close to a specific ID, we reply with new faked peers that match this ID, or are very close to it, to guarantee the success of the attack. If the peer asks for stored information we deliver poisoned results, as in the two attacks discussed before.

Table 1. Percentage of faked replies received during 10 runs of the eclipse attack. Each run r was measured 15 minutes with an interval of one minute.

Minute	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	$\bar{r}$
1.	0	0	0	0	0	0	0	0	0	0	0
2.	0	0	0	0	0	0	0	0	0	0	0
3.	0	0	0	0	0	0	0	0	0	0	0
4.	0	0	0	81	0	0	78	0	0	0	15.9
5.	72	100	100	65	23	100	81	81	100	65	78.7
6.	78	100	90	72	85	100	78	72	100	81	85.6
7.	81	82	100	81	78	81	100	78	100	100	88.1
8.	65	100	100	100	81	100	100	68	81	100	89.5
9.	58	100	100	95	100	100	100	89	100	100	94.2
10.	78	100	100	100	100	100	98	100	100	100	97.6
11.	100	100	100	100	100	100	100	100	100	100	100
12.	100	100	100	100	100	100	100	100	100	100	100
13.	100	100	100	100	100	100	100	100	100	100	100
14.	100	100	100	100	100	100	100	100	100	100	100
15.	100	100	100	100	100	100	100	100	100	100	100

Table 1 shows the results of 10 repeated eclipse attacks under the same conditions. To measure the success rate of the attacks, we periodically ran searches on the attacked peer and counted the number of poisoned results. As the success rate virtually always reaches 100% within minutes, we can conclude that the attack works well, especially if the attack is focused on a single keyword, but it is naturally limited to merely a single attacked peer. The other two attacks are clearly preferable if an attacker aims at hiding content from *all* peers.

7. Discussion

The preceding section has presented three different attacks that can be used to keep peers from acquiring the requested information. Naturally, these attacks can also be combined in order to increase the chances

of a successful attack. However, these poisoning attacks cannot only be used for this purpose. Rather, they can serve an attacker as basic building blocks to pursue completely different aims.

We will now briefly illustrate how they can be used for another attack. The resources of the Kad network's peers and our attacks can be used to drive a *distributed denial of service attack* (DDoS) against any machine internal or external to the Kad network as follows: A node insertion attack is performed in order to occupy some popular keywords. Let μ be the machine (e.g., a server) to be attacked. We inform all requesters that μ contains the desired files. Consequently, all requests are directed to the attacked machine. Of course, the resulting load on μ is not larger than on the machine performing the node insertion. However, the advantage of this attack is that the attacking machine *remains hidden*; moreover, it is generally harder to counter a distributed DoS attack than a normal DoS attack as the requests originate from different (and valid) IP addresses. Also the Publish Attack can be used for the DDoS attack if we advertise wrong IP bindings of keywords. This has the additional advantage that the attack induces more load on the attacked machine than on the attacker, as the different Kad peers are directed to the attacked machine directly. Note that DDoS attacks using a p2p system such as Kad are particularly nasty as the peers store information about sources for a long period of time, implying that such an attack could last several days with steadily changing peers involuntarily performing the attack.

As all the described attacks can be performed easily and have a large impact, it is mandatory to derive and implement counteractive measures. In order to overcome the node insertion attack it must be guaranteed that choosing specific IDs is infeasible. A straightforward approach, which is often described in literature (and which is used, e.g., by the Azureus BitTorrent client), is to bind the ID directly to the peers' IP addresses, e.g., by hashing the IP address. However, there are several reasons why real-world p2p systems do not adhere to this simple rule. First, multiple peers may share the same IP address, for example, peers in a local area network behind a NAT router are typically addressed using the same public IP address. These peers would all have the same peer identifier. Second, IP addresses are often given out dynamically and the assignment of addresses may change. In case of an ID-IP binding, this implies that peers have to rebuild their routing tables when reconnecting to the network with a new IP. Additionally, all the credits gathered by uploading data would be lost irretrievably because the peer ID changed and hence the peer cannot be recognized by other peers anymore. It seems that some of these problems can be solved easily and the IP address can still be incorporated into the ID, e.g., by hashing the IP address and a randomly chosen bit string to solve the NAT problem, or by using a different, randomly chosen ID for the credit system, together with a public and private key pair to protect it against misuse.[7] Hashing the IP address and a user-generated bit string is preferable to including the port as this would require a static assignment of ports, and switching ports would also lead to a new ID. However, a crucial point is that creating such a binding is not sufficient to avert the attack in general, as long as the ID includes a user-generated part. Assuming that a hash function such as SHA-1 is used, an attacker can try out millions of bit strings in a short period of time in order to generate an ID that is closest to the targeted keyword even in a network containing more than a million peers. These observations indicate that some form of peer authentication is required, which is hard to achieve without the use of a centralized verification service. As part of the strength of the network is its completely decentralized structure, relying on servers does not seem to be an acceptable solution.

A simple heuristic to render the Kad network more resilient to publish and eclipse attacks is to limit the amount of information a peer accepts from the same IP address, i.e., a peer does not allow that its

[7] In fact, Kad already uses public and private keys to authenticate peers whenever a new session starts.

entire contact list is filled by peers using the same IP address. This is also a critical solution as several peers behind a NAT may indeed have the same public IP address. What is more, an attacker with several IP addresses at its disposal can circumvent this security measure.

An important observation is that many of the discussed vulnerabilities do not only pertain to the Kad network, such attacks can be launched against any fully decentralized system that does not incorporate strong verification mechanisms. We believe that in recent literature, some interesting approaches have been proposed that may be useful not only in the context of the Kad network, especially the work on join-leave attacks [20] by Scheideler who studies how to spread peers over a virtual ID space $[0, 1)$ in a robust way. In [5], Awerbuch and Scheideler proposed a robust distributed (round-robin) random number generator. Intriguingly, while constructing a single random number is difficult, it turns out that a set of random numbers can be generated by a group of peers in a scalable manner that is resilient to a constant fraction of adversarial peers. Unlike the verifiable secret sharing algorithm described in [3], their solution cannot fail if the initiating peer does not behave correctly, and a peer cannot rerun the protocol sufficiently many times until an ID is generated that falls into a desired range. This is certainly a desirable property to overcome the node insertion attacks described in this article. However, important questions remain open, for instance, how to handle concurrent rejoin operations, or how to cope with ongoing DoS attacks.

8. Conclusion

Understanding the behavior of peers in large p2p networks enables the development of new and more efficient distributed algorithms or may even pave the way for novel applications in distributed systems. In this article, we have reported on our measurement insights and compared the peer activity in the server-based eDonkey network to the distributed hash table Kad, two of the largest peer-to-peer networks in use today. We find that not only do most requests arrive roughly during the same time interval every day in both networks, the searched content is also quite similar. Moreover, by counting the number of source requests we discovered that our server receives roughly 10% of all eDonkey requests. Using this estimate, and given that we receive virtually all requests for certain keywords in Kad, we conclude that the eDonkey network is still more popular. In total, we estimate the total number of requests in eDonkey to be somewhere between 1.3 and 2 times larger than in Kad. It will be interesting to see how the situation develops in the future.

Due to their properties, the use of DHTs or similar structured networks has been proposed as the foundation of the "future Internet" in order to overcome the deficiencies of today's Internet. Therefore, in the second part of this article, the robustness of Kad is examined in more detail. In particular, we provide evidence that the Kad network can be attacked with a small amount of computing resources such that access to popular files is denied. It is clear that such attacks could significantly lower the throughput of the entire system as the sought-after files are no longer found, and that this imposed censorship would frustrate the users. Moreover, the possibility of leveraging the immense computational resources of the entire system to attack arbitrary machines constitutes a serious threat. Finally, one may also speculate that the increasing number of spam replies observed in eMule today may be due to mechanisms that are similar and inspired by the attacks described in this article.

We argue that the presented attacks can basically be launched in any peer-to-peer system that does not incorporate sound peer authentication mechanisms, and we have initiated a discussion of different

approaches to overcome these vulnerabilities. From this first discussion we conclude that while there are both practical and theoretical schemes that seem to improve the robustness, more research is needed on how to apply them optimally "in the wild".

Acknowledgments

We would like to thank David Mysicka for his work on the measurements, and Bernhard Ager and Christian Scheideler for interesting discussions. Preliminary versions of this article appeared at DYNAS 2009 [14] and ICDCN 2010 [15].

References

[1] E. Adar and B. A. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), 2000.

[2] E. Athanasopoulos, K. G. Anagnostakis, and E. P. Markatos. Misusing Unstructured P2P Systems to Perform DoS Attacks: The Network That Never Forgets. In *Proc. 4th International Conference on Applied Cryptography and Network Security (ACNS)*, 2006.

[3] B. Awerbuch and C. Scheideler. Towards a Scalable and Robust DHT. In *Proc. 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 318–327, 2006.

[4] B. Awerbuch and C. Scheideler. Towards Scalable and Robust Overlay Networks. In *Proc. 6th IPTPS*, 2007.

[5] B. Awerbuch and C. Scheideler. Robust Random Number Generation for Peer-to-Peer Systems. *Theor. Comput. Sci.*, 410(6-7):453–466, 2009.

[6] D. Carra and E. W. Biersack. Building a Reliable P2P System out of Unreliable P2P Clients: The Case of KAD. In *Proc. ACM CoNEXT*, 2007.

[7] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 299–314, 2002.

[8] K. E. Defrawy, M. Gjoka, and A. Markopoulou. BotTorrent: Misusing BitTorrent to Launch DDoS Attacks. In *Proc. 3rd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2007.

[9] J. R. Douceur. The Sybil Attack. In *Proc. 1st IPTPS*, 2002.

[10] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proc. 5th International Workshop on Peer-to-Peer Systems*, 2006.

[11] A. Haeberlen, A. Mislove, A. Post, and P. Druschel. Fallacies in Evaluating Decentralized Systems. In *Proc. 5th International Workshop on Peer-to-Peer Systems*, 2006.

[12] J. Liang, N. Naoumov, and K. W. Ross. The Index Poisoning Attack in P2P File Sharing Systems. In *Proc. 25th Annual IEEE Conference on Computer Communications (INFOCOM)*, 2006.

[13] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding in BitTorrent is Cheap. In *Proc. 5th Workshop on Hot Topics in Networks (HotNets)*, 2006.

[14] T. Locher, D. Mysicka, S. Schmid, and R. Wattenhofer. Invited Paper: A Peer Activity Study in eDonkey & Kad. In *Proc. International Workshop on Dynamic Networks: Algorithms and Security (DYNAS)*, 2009.

[15] T. Locher, D. Mysicka, S. Schmid, and R. Wattenhofer. Poisoning the Kad Network. In *Proc. 11th International Conference on Distributed Computing and Networking (ICDCN)*, 2010.

[16] P. Maymounkov and D. Mazières. A Peer-to-Peer Information System Based on the XOR Metric. In *Proc. 1st IPTPS*, 2002.

[17] N. Naoumov and K. Ross. Exploiting P2P Systems for DDoS Attacks. In *Proc. 1st International Conference on Scalable Information Systems (INFOSCALE)*, 2006.

[18] S. J. Nielson, S. A. Crosby, and D. S. Wallach. A Taxonomy of Rational Attacks. In *Proc. 4th IPTPS*, 2005.

[19] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. of Multimedia Computing and Networking (MMCN)*, 2002.

[20] C. Scheideler. How to Spread Adversarial Nodes?: Rotate! In *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 704–713, 2005.

[21] A. Singh, T.-W. J. Ngan, P. Druschel, and D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proc. 25th Annual IEEE Conference on Computer Communications (INFOCOM)*, 2006.

[22] M. Steiner. Private Communication.

[23] M. Steiner, E. W. Biersack, and T. En-Najjary. Actively Monitoring Peers in the KAD. In *Proc. 6th IPTPS*, 2007.

[24] M. Steiner, D. Carra, and E. W. Biersack. Faster Content Access in KAD. In *Proc. 8th IEEE Conference on Peer-to-Peer Computing (P2P)*, 2008.

[25] M. Steiner, T. En-Najjary, and E. W. Biersack. Exploiting KAD: Possible Uses and Misuses. In *Computer Communication Review 37(5)*, 2007.

[26] M. Steiner, T. En-Najjary, and E. W. Biersack. A Global View of KAD. In *Proc. 7th ACM IMC*, 2007.

[27] D. Stutzbach and R. Rejaie. Understanding Churn in Peer-to-Peer Networks. In *Proc. 6th IMC*, 2006.

[28] D. Stutzbach and R. Rejaie. Improving Lookup Performance over a Widely-Deployed DHT. In *Proc. 25th IEEE INFOCOM*, 2006.

[29] X. Sun, R. Torres, and S. Rao. Preventing DDoS Attacks with P2P Systems through Robust Membership Management. *Technical Report TR-ECE-07-13, Purdue University*, 2007.

[30] D. S. Wallach. A Survey of Peer-to-Peer Security Issues. In *Proc. International Symposium on Software Security*, 2002.

[31] L. Zhou, L. Zhang, F. McSherry, N. Immorlica, M. Costa, and S. Chien. A First Look at Peer-to-Peer Worms: Threats and Defenses. In *Proc. 4th IPTPS*, 2005.