Exploiting KAD: Possible Uses and Misuses

Moritz Steiner, Ernst W. Biersack, Taoufik En-Najjary
Institut Eurecom, Sophia–Antipolis, France
{steiner, erbi, ennajjar}@eurecom.fr

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
Authors take full responsibility for this article's technical content.
Comments can be posted through CCR Online.

ABSTRACT

Peer-to-peer systems have seen a tremendous growth in the last few years and peer-to-peer traffic makes a major fraction of the total traffic seen in the Internet. The dominating application for peer-to-peer is file sharing. Some of the most popular peer-to-peer systems for file sharing have been Napster, FastTrack, BitTorrent, and eDonkey, each one counting a million or more users at their peak time.

We got interested in KAD, since it is the only DHT that has been part of very popular peer-to-peer system with several million simultaneous users. As we have been studying KAD over the course of the last 18 months we have been both, fascinated and frightened by the possibilities KAD offers. Mounting a Sybil attack is very easy in KAD and allows to compromise the privacy of KAD users, to compromise the correct operation of the key lookup, and to mount DDOS with very little resources.

In this paper, we will relate some of our findings and point out how KAD can be used and misused.

Categories and Subject Descriptors

H.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Systems and Software – Distributed systems

General Terms

Algorithms, Security

Keywords

Distributed Hash Table, Sybil attack, peer-to-peer system.

1. INTRODUCTION TO KAD

KAD is a Kademlia-based [15] peer-to-peerDHT routing protocol implemented by several peer-to-peerapplications such as Overnet [18], eMule [11], and aMule [1]. The two open–source projects eMule and aMule have the largest number of simultaneously connected users since these clients connect to the eDonkey network, which is a very popular peer-to-peersystem for file sharing. Recent versions of these clients implement the KAD protocol.

As in other DHTs, each KAD node has a global identifier, referred to as KAD ID, which is a 128 bit randomly generated identifier. The KAD ID is generated when the client application is started for the first time and is then permanently stored with that client. The KAD ID stays unchanged on subsequent join and leaves of the peer, until the user deletes the application or its preferences file. However, as we show [26] there are quite a few peers that do not follow this rule and change their KAD ID very frequently.

1.1 Routing Lookup

Routing in KAD is based on prefix matching: Node a forwards a query, destined to a node b, to the node in his routing table that has the smallest XOR-distance. The XOR-distance $d(a, b)$ between nodes a and b is $d(a, b) = a \oplus b$. It is calculated bitwise on the KAD IDs of the two nodes, e.g. the distance between $a = 1011$ and $b = 0111$ is $d(a, b) = 1011 \oplus 0111 = 1100$. For details of the implementation see [27]. The entries in the routing table of a peer P point to peers that are a various distances from P: A peer P stores only a few contacts to peers that are far away in the ID space and increasingly more contacts to peers as we get closer P. For details of the implementation see [27].

Routing to a given KAD ID is done in an *iterative way*. To improve robustness against node churn that can result in stale routing table entries and to improve and look-up speed, the requesting peer P runs *three parallel* routing lookups for a given key at the same time: A peer P first consults his routing table to determine the three peers closest to the KAD ID. P sends `route requests` to these three peers, which may or may not return to P `route responses` containing new peers even closer to the KAD ID, which are queried by P in the next step. The routing lookup terminates when the returned peers are further away from the KAD ID than the peer returning them.

While iterative routing experiences a slightly higher delay than recursive routing, it offers increased robustness against message loss and it greatly simplifies crawling the KAD network. In KAD, a routing lookup will be performed in a first step by both, the publish and the search module.

1.2 Publishing and Searching

A **key** in a peer-to-peer system is an identifier used to retrieve information. In many peer-to-peer systems a key is typically published on a single peer that is numerically closest to that key. In KAD, to deal with node churn, a key is published *on ten different peers whose* KAD ID *agrees at least in the first 8-bits with the key*. This range of KAD IDs around a key that agree in the first 8-bits with the key is called the **tolerance zone**. Note that the key is not published on the ten peers *closest* to the key, but simply on peers whose KAD IDs are in the tolerance zone. To assure persistence of the information stored, the owner periodically republishes the information every 5 or 24 hours, depending on the type of information.

As for the publishing, the search procedure uses the routing lookup to find the peer(s) closest to the key searched for. To increase the robustness of the search in case of stale routing table entries, three searches are launched in parallel. If the first arriving `route response` contains peers that are closer to the destination, immediately new `route requests` are sent. The four most important message types are:

- `hello`: to check if the other peer is still alive and to inform the other peer about one's existence and the KAD ID and IP address.
- `route request/response(kid)`: To find peers that are closer to the KAD ID kid.
- `publish request/response`: to publish information.
- `search request/response(key)`: to search for information whose hash is `key`.

2. EXPLORING KAD

We have developed our own crawler for KAD, with the aim to crawl KAD frequently and over a duration of several months. Our crawler runs on a local machine and uses a simple breadth first search issuing `route requests` to find the peers currently participating in KAD. The speed of our crawler allows us to crawl the entire KAD system (entire KAD ID space) in about 8 minutes, which was never done before. During a full crawl, we found between 3 and 4.3 million different peers. Between 1.5 and 2 million peers are not located behind NATs or firewalls and can be *directly contacted* by our crawler.

However, to limit the network load and the data volume, we decided to crawl only a part of the KAD ID space by carrying out a **zone crawl** on a 8-bit zone, where we try to find all active peers whose KAD IDs have the same 8 high-order bits. A zone crawl explores one 256-th of the entire KAD ID space and takes less than 2.5 seconds. For *slightly less than 6 months* we crawled the same zone every 5 minutes. The detailed results of our crawl are reported in [24, 26].

We made some surprising findings such as (i) several thousand KAD clients that all had the same KAD ID and (ii) several hundred peers with the same sub-net IP addresses and KAD IDs that all agreed in a large number of least significant bits. The last case seems to indicate a **Sybil attack**, which was first defined by J. Douceur [10] as "the forging of multiple identities". KAD, as are all the other peer-to-peer systems, is vulnerable to Sybil attacks. In the following, we will discuss how Sybil attacks can be exploited in KAD for various purposes.

3. SYBIL ATTACKS IN KAD

The main idea of the **Sybil** attack [10] is to introduce malicious peers, the *sybil*s, which are all controlled by one entity. Positioned in a strategic way, the *sybil*s allow to gain control over a fraction of the peer-to-peer network or even over the whole network. The *sybil*s can monitor the traffic (behavior of the other peers) or abuse of the protocol in other ways. Routing requests may be forwarded to the wrong end-hosts or rerouted to other *sybil* entities.

3.1 Spying on Publish and Search Traffic

Assume that we want to find out in the least intrusive way what type of content is published and searched for in a zone $\mathcal{Z}$ of the KAD network. For this, one needs to introduce *sybil*s in the zone $\mathcal{Z}$ and to make them known, so that their presence is reflected in the routing tables of the *regular*, i.e. non-sybil peers.

We have developed a light-weight implementation of such a "spy" that is able to create thousands of *sybil*s on one single physical machine as they do not keep any state about the interactions with the regular peers [25].

When we spy on a 8-bit zone, we introduce 2^{16} *sybil*s: the first 8 bit are defined by the zone we spy on, the following 16 bits are different for each *sybil*. The spy works as follows:

- First, crawl a zone $\mathcal{Z}$ of the KAD ID space using our crawler to to learn about the peers $\mathcal{P}$ currently online whose KAD IDs are in $\mathcal{Z}$.
- Then, send `hello requests` to the peers $\mathcal{P}$ in order to "poison" their routing tables with entries that point to our *sybil*s. The peers that receive a `hello request` will add the *sybil* to their routing table if the corresponding bucket of the routing table is not filled.
- Later, when a `route request(kid)` initiated by regular peer P reaches a *sybil* that request will be answered with a set of *sybil*s whose KAD IDs are closer to the target in case the `kid` falls into the zone $\mathcal{Z}$ and ignored otherwise.

  This way, P has the impression of approaching the target. Once P is "close enough" to the target KAD ID, it will initiate a `publish request` or `search request` also destined to one of our *sybil* peers. Therefore, for any `route request` that reaches one of our *sybil* peers we can be sure that the follow-up `publish request` or `search request` will also end-up on the same *sybil*.
- Store the content of all the requests received in a database for later evaluation.

As described in Section 1, a key is published ten times and for a search three parallel search requests are issued. For our spy scheme to work as intended, the optimum would be to attract exactly one copy of every search or publish request. This way, publish and search request would also "terminate" on regular peers that would correctly execute them, avoiding any disruption of KAD due to our spy. There are two parameters to control the level of intrusiveness: The number of *sybil*s placed in a zone and the rate at which *sybil*s are announced to regular peers.

The spy has already allowed us to make a number of interesting observations concerning the frequency of the keywords used in the publish and the search requests. Spying on 8-bit zone for one day, we see 1.4 million distinct files being published, using 42,000 different keyword hashes, by 1.5 million distinct users. Per minute, about 1000 search requests, 10,000 publish requests and 25,000 route requests hit our *sybil*s, which amounts to a load of approx. 400 KByte/sec for the incoming and approx. 200 KByte/sec for the outgoing traffic.

We also measured the total traffic due to the different types of requests and were very surprised to see that the "publish traffic" by far outweighs the "search traffic". In fact, the "publish traffic" is one order of magnitude larger – in terms of the number of messages – and *two orders of magnitude larger* – in terms of the total number of bytes transmitted than the "search traffic" [25]. This observation lead us to design an improved publish scheme that maintains the same degree of availability for the information published while reducing the amount of traffic by one order of magnitude [4].

3.2 Eclipsing Content

A special form of sybil attack is the **eclipse** attack [22] that aims to separate a part of the peer-to-peer network from the rest. The way we perform an eclipse attack resembles very much that of the sybil attack described above, except that the KAD ID space covered is much smaller.

To eclipse a particular keyword K, we position a certain number of *sybil*s closely around K, i.e. the KAD IDs of the *sybil*s are closer to the hash value of K than the KAD IDs of any real peer. We then need to announce these *sybil*s to the regular peers in order to "poison" the regular peers routing tables and to attract all the `route requests` for keyword K. Our experiments showed that as few as eight *sybil* peers are sufficient to make sure that all `search requests` for K will terminate on one of the *sybil*s.

Note that even if the keyword K can not be found anymore using the search algorithm employed in KAD, it does still exist on the regular peers where it was originally published.

Depending on the popularity of the content to be eclipsed, the resource consumption varies as we can see in table 1. This data was collected using 32 *sybil*s all running on the same physical machine. We see that it is possible to eclipse content using a very limited amount of resources.

message type (messages per min)	keyword *the*	keyword *dreirad*
`route`	41801	818
`hello`	1091	433
`publish`	12360	290
`search`	704	49
Total incoming bandwidth (KByte/sec)	186	32

Table 1: Traffic seen by the *sybil*s that eclipse the keywords *the* and *dreirad*.

3.3 DDOS Attacks

A sybil attack can also be used to launch a **DDOS** attack that enlists a large number of peers that participate in KAD. As the previous two attacks, we need to place *sybil* peers. However, in difference to the eclipse attack where incoming search queries have been dropped by the *sybil* peer, the *sybil* peer now replies to the request and includes in his response the IP address of the "target" to be attacked.

Depending on the number of *sybil*s and their placement in zones that receive more or less search traffic, the amount of attack traffic can be controlled. We have tried such an attack against some of our own machines that were hit by an incoming traffic in the order of several Mbits/sec.

These kinds of attacks are already happening in the Internet. A news release from earlier this year by Prolexic reports [19] that DDOS attacks using peer-to-peersystems that involve more than 300,000 peers have recently been observed.

4. KAD AS AN EXPERIMENTAL PLATFORM

In the previous section, we have seen the vulnerabilities of KAD, which are also common to other DHTs. However, we feel that KAD also has quite some potential as an experimental platform for research in distributed systems. Let us just outline a few possible uses.

KAD as a Public DHT

Experimental platforms such as Planetlab find intensive use in the research community and various services have been implemented on top Planetlab such as CoDeeN, Coral, or OpenDHT [7, 13, 17].

KAD is one of the largest distributed peer-to-peer applications with several million active peers at any point of time. Using the KAD primitives for routing, publishing and searching, one can utilize KAD as a "public DHT". In a such realistic setting with high node churn, large geographical diversity and many low bandwidth connections, one can investigate alternative routing lookup policies (varying the degree of parallelism), or different publishing strategies (varying the replication factor or the content refresh intervals).

DDOS Defense Research

As we have seen, peers in KAD can be easily tricked in participating in a DDOS attack by making them connect to any machine on the Internet that is the target of the attack.

Researchers that work on DDOS defense could use KAD to test the effectiveness of their defense system by subjecting their system to an attack. These experiments, as we have seen in Section 3.3, can be carried out in a very controlled way and as soon we stop our *sybil*s from returning the IP address of the target, the attack will stop.

Another, more questionable use could be DDOS attack retaliation. The victim of DDOS attack could use KAD to counterattack the machines that originate the attack.

5. HOW TO PREVENT SYBIL ATTACKS

Sybil attacks pose a serious threat to the security of peer-to-peer systems. While there have been various attempts to address this issue, we feel that the solutions proposed are not practicable since they, for instance, impose heavy constraints on the structure of the routing table or require auditing procedures that are difficult to implement.

We feel that there is a great need for solutions that are technically feasible and easy to put into place. Basically, we need to prevent a peer (i) from choosing the KAD ID he will use and (ii) from obtaining a large number of KAD IDs. We will sketch out a *centralized* solution that makes it impossible for an attacker to obtain arbitrary KAD IDs. While centralized solutions have their obvious disadvantages such as single point of failure, they have proven in practice often to be quite satisfactory. Just take the example of BitTorrent with the tracker as centralized component. At first sight such a tracker seems to be an easy target for a denial of service attack. More recent implementation have therefore started to replace the tracker by a DHT. However, when we compare the vulnerabilities of DHTs as discussed in this paper for the case of KAD we may well conclude that using a tracker-based approach is subject to fewer vulnerabilities than a DHT.

The central idea of our proposal is to tie the possibility of obtaining a KAD ID to the possession of a cell phone number. The protocol is as follows:

There is a **central agent** (CA) responsible for generating KAD IDs. The CA needs to have a pair of public and private keys `Kpub` and `Kpriv`.

A client R that needs a KAD ID sends a request containing his cell phone number `phone`, the IP address `IP@` of the peer that will run KAD, and a desired expiration time `To` to the central agent.

When the CA receives the request, it will

- concatenate (`IP@`, `To` , `pad`) into a string `ST`, where `pad` is a padding sequence that assures that `ST` has the required length
- encrypt `ST` with the private key `Kpriv` to obtain the requested KAD ID `id`.
- Send an SMS (short message) to the cell phone number `phone` of the requester and either communicate `id` or another shorter string (password) that then allows to obtain the `id` via the Internet.

If the public key `Kpub` is known to all peers, any peer can verify if a given KAD ID is valid by decrypting the KAD ID using `Kpub` and comparing the IP address contained with the one of the originator of the message.

If the CA keeps lists of all (`phone`, `To`) and (`IP@`, `To`) pairs, it can assure that it will not issue another KAD ID to the same cell phone number and for the same IP address before the previous one has expired.

The scheme just presented has two main drawbacks. Whenever a peer changes his IP address, it needs to obtain a new KAD ID. Many access providers change the IP address of their end-users at a regular basis. However, if we do not tie the IP address to the KAD ID there is no way to prevent clients from either "giving away" their KAD ID or to prevent fraudulent clients from "stealing" the KAD IDs of other peers.

Another obstacle to the deployment of the scheme may be the need for the CA to send a large number SMS per day. However, companies like Google already do so, for instance to inform users about the newly installed e-mail account or about an appointment in their agenda that is due. Alternatively, we may replace the use of SMS by a "Reverse Turing Test" using e.g. a CAPTCHA [3]. However, in this case the effort to obtain multiple KAD IDs will be reduced if we assume that is it easier to solve multiple CAPTCHA than to obtain an equivalent number of different cell phone numbers.

In any case, it will never possible to prevent an attacker with a lot of resources from obtaining multiple (*random*) KAD IDs. For this reason, it may be worthwhile to explore techniques that make the routing lookup in peer-to-peer systems more robust against Sybil attacks as has been proposed in [8, 9]. Nevertheless, in a peer-to-peer system of the size of KAD, which has several Million simultaneous peers, an attacker will probably need to introduce thousands of sybils in order to disturb the system.

6. RELATED WORK

There has been a small body of work that addresses the issue of DDOS attacks using peer-to-peer systems. Naoumov et al. [16] discuss attacks for the case of the now defunct Overnet system. Since routing in Overnet resembles closely routing in KAD, their findings are very relevant to KAD. Two types of attacks are identified: **Index poisoning** attacks where bogus records are inserted into the overlay in order to direct peers searching for content to a target host that will become the victim of the DDOS attack. **Routing poisoning** attacks where many peers are tricked into adding the target host into their routing table. As a consequence the target host will receive a lot of signaling (query, publish and maintenance) traffic.

El-Defrawy et al. [12] have investigated index poisoning attacks in BitTorrent and Athanasopoulos et al. [2] discuss how to launch DDOS attacks in Guntella, an unstructured peer-to-peer file sharing system.

There exist quite a few proposals in the literature to improve the security of DHTs.

DHT-based overlay systems are susceptible to various attacks launched by malicious peers that may corrupt data, deny response to lookup queries, or impersonate other peers so that data objects may be stored on rogue peers.

In DHTs-based systems, each node has a global identifier ID, which is generated when the client application is started for the first time. If an attacker controls a fraction, even small, of nodes with smartly chosen IDs, it can "eclipse" correct nodes and prevent correct overly operation. The malicious nodes may be different entities or the same entity with many identities (IDs).

Sit et al. [23] provide a clear description of security considerations that involve peers that do not follow the protocol correctly: routing deficiencies due to corrupted routing lookup nd updates; vulnerability to partitioning when new peer joins and contacts malicious peers; lookup and storage attacks; inconsistent behaviors of peers; denial of service attacks; and unsolicited responses to a lookup query. They argue that the peer's identifier assignment must be done in a verifiable way, and that the identifier must not be chosen by the node itself. However, they mention that a central identification authority is not desirable in all situations.

Douceur [10] was the first to consider the problem of multiple identities in the context of DHT-based peer-to-peer systems (The Sybil attack). He showed that without the use of a centralized authority that certifies all nodes, it is impossible to prevent this attack.

Castro et al. [5] presented a design and analysis of techniques for secure peer joining. They propose to certify the node IDs by a set of trusted certification authorities (CAs). Node ID certificates are signed by the CAs, which use a public key that must be known by all network nodes. To prevent an attacker from obtaining certificates, they propose to bind the ID to peer's IP address, or require paying money for certificate.

Rowaihy et al. [20] propose an admission control system that mitigates Sybil attacks by adaptively constructing a hierarchy of cooperative admission control nodes. This creates a tree structure with static root. A node wishing to join the network is serially challenged using a hash puzzle by the nodes from the leaf to the root. Each challenger node creates a cryptographic puzzle based on a hash function and the solver has to invert the hash. As hash-functions are non-invertible, the solver must use brute force to find the solution, which will require a large number of attempts. This solution relies on the limitation of computational power of the joining node, however, it may still allow a resourceful attacker to launch a substantial attack, especially if the potential for damage is disproportionate to the fraction of the system that is compromised.

Yu et al. [28] propose SybilGuard, a protocol for limiting the corruptive influence of the Sybil attack. SybilGuard is based on social network among user identities, when an edge between two identities indicates a human-establish trust relationship. Malicious users can create many identities but will have only few trust relationships. The deployment of SybilGuard requires the existence of a well-connected social network, which not the case of todays DHT-based peer-to-peer systems.

While a successful Sybil attack can be used to mount an Eclipse attack, Eclipse attacks are possible even in the presence of an effective defense against Sybil attacks. To defend against eclipse attacks, Castro et al. [5] proposed the use of Constrained Routing Tables (CRT), where a node's neighbor set contains nodes with identifiers closest to well-defined points in the identifier space, which leaves no flexibility in neighbor selection and therefore prevents optimizations like proximity neighbor selection, an important and widely used technique to improve overlay efficiency [6, 14]. In addition to CRT, Singh et al. [21, 22] propose to bound the in- and out-degree of overlay nodes, and present a defense strategy based on anonymous auditing of nodes' neighbor sets. If a node has significantly more links than the average, it might be a malicious node, and then it can be removed from the neighbor sets of the correct nodes.

7. CONCLUSION

Distributed systems for content sharing are presumably believed to be more robust against attacks as centralized systems that have a single point of failure. However, in practice this may not be the case as long as the Sybil attack is possible. We have discussed the implications of the Sybil attack in the case of KAD, which is the largest DHT currently deployed:
The privacy of the end-users can easily be compromised, KAD itself can be arbitrary disrupted, and the peers that participate in KAD can be enlisted against their will to participate in a DDOS attack. Any of these attacks can be launched from a single PC connected to the Internet via a broadband connection. For all these reasons, it is urgent to implement practical solutions that prevent sybil attacks.

On the positive side, we have also seen that KAD can be used as an Open DHT providing a realistic test-bed for research in peer-to-peer systems.

8. REFERENCES

[1] A-Mule. `http://www.amule.org/`.

[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. of ACNS 2006*, June 2006.

[3] CAPTCHA. `http://en.wikipedia.org/wiki/CAPTCHA`.

[4] D. Carra and E. Biersack. Building a reliable p2p system out of unreliable p2p clients: The case of kad. Technical report, Institut Eurecom, July 2007. Submitted to Conext 2007.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of OSDI'02*, Boston, USA, Dec. 2002.

[6] M. Castro, P. D. Y. C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical Report MSR-TR-2002-82, Microsoft Research, 2002.

[7] CoDeeN. `http://codeen.cs.princeton.edu/`.

[8] T. Condie, V. Kacholia, S. Sankararaman, J. Hellerstein, and P. Maniatis. Induced churn as shelter from routingtable poisoning. In *Proc. 13th Annual Network and Distributed System Security Symposium (NDSS)*, 2006.

[9] G. Danezis, C. L. Laas, F. M. Kaashoek, and R. Anderson. Sybil-Resistant DHT Routing. In *ESORICS*, pages 305–318, Sept. 2005.

[10] J. R. Douceur. The Sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, LNCS, pages 251–260, March 2002.

[11] E-Mule. `http://www.emule-project.net/`.

[12] K. El-Defrawy, M. Gjoka, and A. Markopoulou. BotTorrent: Misusing BitTorrent to Launch DDoS Attack. In *Proc. USENIX SRUTI*, June 2007.

[13] M. J. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with coral. In *Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, Mar. 2004.

[14] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03*, 2003.

[15] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer informatiion system based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 53–65, Mar. 2002.

[16] N. Naoumov and K. Ross. Exploiting p2p systems for ddos attacks. In *International Workshop on Peer-to-Peer Information Management*, May 2006.

[17] OpenDHT. `http://opendht.org/`.

[18] Overnet. `http://www.overnet.org/`.

[19] Prolexic. Prolexic Distributed Denial of Service Attack Alert, May 2007. `http://www.prolexic.com/news/20070514-alert.php`.

[20] H. Rowaihy, W. Enck, P. McDaniel, and T. La Porta. Limiting sybil attacks in structured p2p networks. In *26th IEEE International Conference on Computer Communications (INFOCOM)*, pages 2596–2600, 2007.

[21] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. In *ACM SIGOPS 2004*, 2004.

[22] A. Singh et al. Eclipse attacks on overlay networks: Threats and defenses. In *Proc. Infocom 06*, Apr. 2006.

[23] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of IPTPS'02*, Cambridge, MA, Mar. 2002.

[24] M. Steiner, E. W. Biersack, and T. En-Najjary. Actively Monitoring Peers in Kad. In *Proceedings of the* 6th *International Workshop on Peer-to-Peer Systems (IPTPS'07)*, 2007.

[25] M. Steiner, W. Effelsberg, T. En-Najjary, and E. W. Biersack. Load reduction in the kad peer-to-peer system. In *Fifth International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2007)*, 2007.

[26] M. Steiner, T. En-Najjary, and E. W. Biersack. A Global View of KAD. In *Proceedings of the Internet Measurement Conference (IMC)*, 2007.

[27] D. Stutzbach and R. Rejaie. Improving lookup performance over a widely-deployed DHT. In *Proc. Infocom 06*, Apr. 2006.

[28] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: Defending against sybil attacks via social networks. In *SIGCOMM*, 2006.