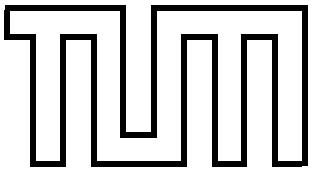


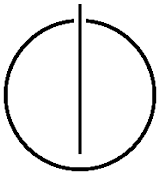
FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Computer Science

Eclipse Attacks on Nodes in the Kad Peer-to-Peer Network

Jan Seeger



Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Eclipse Attacks on Nodes in the Kad Peer-to-Peer Network

—

Eclipse-Angriffe auf Knoten im Kad Peer-to-Peer Netzwerk

Bachelorarbeit in Informatik

durchgeführt am
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

von

Jan Seeger

Aufgabensteller:	Prof. Dr.-Ing. Georg Carle
Betreuer:	Dipl.-Inform. Ralph Holz
Tag der Abgabe:	15. März 2010

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

"In signing this declaration, I confirm that this work is my own. I have acknowledged and referenced all material and sources in the preparation of this thesis"

Garching, den 15. April 2011

Kurzfassung:

Diese Arbeit beschäftigt sich mit Eclipse-Attacken auf Knoten im Kad-Netzwerk. Das Kad-Netzwerk ist ein distributed hash table, welches im eDonkey-Netz für Dateisuchen und -austausch verwendet wird und auf Kademlia basiert. Da Knoten-IDs in Kad zufällig gewählt werden, sind bestimmte Attacken im Netzwer relativ einfach möglich. Diese Arbeit beschäftigt sich im besonderen mit der Eclipse-Attacke.

Eclipse-Attacken wurden bereits im Kad-Netzwerk durchgeführt, das Ziel waren jedoch bisher immer im Netzwerk gespeicherte Daten. Diese Arbeit beschäftigt sich hingegen mit Eclipse-Attacken auf Knoten.

Die Unterschiede zwischen Kad und Kademlia werden erläutert, und Besonderheiten der Kad-Implementierung genannt. Dann wird ein Framework beschrieben, welches die einfache Durchführung und Analyse von Eclipse-Attacken im Kad-Netzwerk erlaubt. Dann werden Angriffsexperimente durchgeführt.

Es ergab sich, dass Eclipse-Attacken auf Knoten im Kad-Netzwerk unter bestimmten Bedingungen möglich sind. Wenn mindestens drei bösartige Knoten in die Routing-Tabellen des suchenden Knoten aufgenommen werden, ist eine Eclipse-Attacke mit maximal 44 Knoten möglich.

Abstract:

This work evaluates the possibility of eclipse attacks on nodes in the Kad network. The Kad network is a distributed hash table which is used in the eDonkey network for file and source exchange. It is based on Kademlia. Several kinds of attacks are made possible by the fact that node IDs are chosen randomly in the network. This work will concern itself with the Eclipse attack.

Eclipse attacks have been executed in the Kad network before. However, the aim of these attacks has always been the content stored in the network. This work, however, will evaluate the possibility of executing eclipse attacks on nodes.

The differences between Kad and Kademlia are briefly presented. A framework is created which allows easy execution and analysis of eclipse attacks in the Kad network. Then, several attack experiments are excuted.

The results show that eclipse attacks on nodes in the Kad network are possible under certain circumstances. When at least three malicious nodes are added to the routing tables of the searching node, an eclipse attack is possible with a maximum of 44 malicious nodes.

Contents

List of Figures

1. Introduction

Peer-to-Peer networks have become very popular in the last years. Benefits of using peer-to-peer technology are scalability, flexibility and resilience against attacks. With these benefits, peer-to-peer networks are useful for transferring large files or sharing large workloads, and have become responsible for the majority of Internet traffic in recent years.

However, the flexibility of peer-to-peer networks also enables certain attacks. Among these are the *Sybil attack* and the *eclipse attack*. The Sybil attack describes the addition of multiple malicious nodes under control of a single attacker into the network. The eclipse attack essentially allows the disconnection of clients from the network by a malicious attacker.

This work will focus on executing eclipse attacks on nodes in the Kad network. While eclipse attacks have been executed in the Kad network before, these attacks were always focused on eclipsing content. While eclipsing content leads to that content no longer being available in the network, eclipsing a node leads to all content stored by that node to become unavailable. Eclipsing nodes is substantially different from attacks on content, and thus, additional research is necessary.

Several popular file sharing networks exist today, such as Gnutella, eDonkey, and BitTorrent. One of the biggest peer-to-peer (from now on abbreviated as P2P) networks is the eDonkey network. The eDonkey network is a server-based P2P system that also contains a serverless component. The serverless component of eDonkey is called Kad, and will be the focus of this work. Kad is a distributed hash table (a DHT) that implements the functionality of the server-based network without centralized servers. Since the eDonkey network has more than 2 million users at most times, and almost all eDonkey clients support Kad, this makes Kad one of the largest public DHTs available, and a good subject for our work.

To evaluate the impact of eclipse attacks in the Kad network, first, the basic DHT mechanisms will be explored. The ancestor of Kad, Kademlia, will be described in Chapter 2.1. Also, eclipse attacks in a (hypothetical) Kademlia network will be described. This allows understanding of the basic attack mechanisms and hurdles, as well as potential differences between attacking nodes ad content.Then, in Chapter 2.2, the differences between Kad and Kademlia will be explained. Details of the

Kad implementation in an eDonkey client will be described as they pertain to the execution of eclipse attacks.

In Chapter 3, related work will be described. Eclipse attacks have been researched in the Kad network before. However, most of this research has focused on attacking content in the network. The differences between prior research and this work will be explained.

In Chapter 4, the experiment framework used will be described. The different roles of nodes and the execution of an experiment will be described in Chapters 4.1 and 4.2. Finally, in Chapter 4.3, the modifications used with aMule to allow execution of the attack will be described.

In Chapter 5, preliminary experiments and attack experiments will be described. The preliminary experiments will be used to assess the boundary conditions of the network, and design attack experiments. Then, attack experiments to evaluate impact and feasibility of an eclipse attack will be run.

In Chapter 6, the experiment results will be summed up. Also, several improved attack strategies and mitigation techniques will be evaluated.

Finally, in Chapter 7, conclusions will be drawn from this work and an outlook for further research will be given.

2. Background

In the next chapter, the protocols Kad and Kademlia and the execution of eclipse attacks in those networks will be explained.

Since Kad is a modification of Kademlia, the Kademlia protocol and the execution of eclipse attacks will be described first in Chapter 2.1. Then, the differences of Kad and Kademlia and the execution of eclipse attacks in the Kad network will be outlined in Chapter 2.2.

2.1 The Kademlia protocol

Kademlia is a distributed hash table (DHT) protocol proposed by Mazières and Maymounkov in [MaMa02]. A DHT is a distributed storage mechanism that takes key-value pairs and stores them in a distributed fashion over a network of participating nodes. An algorithm is then used to find and retrieve the stored data.

A DHT consists of several nodes connected by an *underlay* network (UDP/IP in our case) forming a secondary network, a so-called *overlay* network. Overlay and underlay networks are independent from each other (although performance optimizations in the overlay may take underlay structure into account). Interaction in the overlay network takes place transparently over the underlay.

All nodes are equal, and transfers of data occur directly between two nodes, without the mediation of a server (this is where the name "peer-to-peer" comes from). A DHT connects these nodes to form a large overlay network that supports the storage and retrieval of key-value information.

All DHTs have two operations: `PUT` and `GET`. When a $PUT(k, v)$ is issued into the network, the value v is stored under the key k (like in a regular hash table). When a $GET(k)$ is executed, the value v should be returned from the network. Note that the address space is flat, i.e. not hierarchical. Keys are opaque values that serve only to retrieve the data connected to it.

Storage of the data is persistent over the lifetime of the network, and in order to keep information with nodes joining and leaving the network, redundancy is employed. The degree of redundancy can be adjusted to satisfy different storage requirements.

Kademlia is not modeled for a specific application model. It is a basic DHT that uses several novel ideas in order to implement a simple, yet well performing and robust DHT. Kademlia differs from other DHT implementations in two important points: Firstly, the usage of the XOR metric for distance measurement and secondly, the usage of node uptime as a means for ensuring higher stability of the network.

These design choices lead to a higher overall network stability, decrease required network traffic and allow a flexible network structure. The exact ramifications of these choices will be explained in the next two sections. Then, the basis of executing eclipse attacks will be shown in chapter 2.1.3. After that, differences between the Kad protocol and its ancestor Kademlia will be explained. Finally, in chapter 2.2.2, the execution of eclipse attacks on the Kad network will be described.

2.1.1 The Kademlia Routing Table

All Kademlia entities are identified by an ID in a uniform ID space. Both Kademlia nodes and keys are identified by 160 bit positive integers, node IDs are chosen randomly, and keys are opaque values identifying some content.

The distance between two IDs in the Kademlia network is defined by their XOR distance: $d(ID_1, ID_2) = ID_1 \oplus ID_2$. Using this distance metric has several advantages.

Firstly, the XOR metric is *symmetric*. The distance between node a and node b is equal to the distance between node b and node a: $d(a, b) = d(b, a)$. This is of advantage, since it allows nodes to draw routing information from incoming and outgoing requests. Packets take the same path in both directions. This would not be the case if routing were asymmetric, as, for example, in Chord [SMKK+01]. With an asymmetric distance metric, two routes can have greatly differing lengths in both directions.

Secondly, it is *unidirectional*. This means that there is exactly one node with a certain distance d. This leads to unambiguous routing paths, since there is always exactly one closest node to route to. When the distance metric is not unidirectional, several closest nodes exist, and the routing path is no longer unambiguous. Since all queries take a similar path, this then allows caching of content along this path.

The XOR metric encodes similarity of node IDs: The more bits two IDs have in common, the smaller their distance is. Leading bits have higher impact on the metric, which means the XOR distance is also a weighted hamming distance. This means the shared prefix length of two node IDs can be calculated from their distance by simply finding the index of the largest set bit, or numerically by

$$shpl(ID_1, ID_2) = Length(ID) - floor(log_2(ID_1 \oplus ID_2))$$

.

When keys are chosen at random, they are distributed evenly over the network. Under this assumption, several observations can be made. In a Kademlia network with n participants, the average distance between nodes is $2^{160}/n$. Also, if each bit of the key is equally likely to be one or zero, the probability of a randomly selected node sharing a prefix of length n with a chosen node is 0.5^n. Calculating the mean by $\int_0^{160} n * 0.5^n \mathrm{d}n$ gives us an average shared prefix length of 1.76 bits in a large network. This will be useful later for laying out experiments in the Kad network.

Nodes are organized in a routing table. The routing table of Kademlia is a binary tree whose leaves are *k-buckets*. A k-bucket is simply a container with a maximum size of k. As such, each leaf of the tree can contain at most k contacts. k is a system-wide replication parameter. k is used to adjust the redundancy of data, and thus influences the resistance of the network against data loss and partitioning.

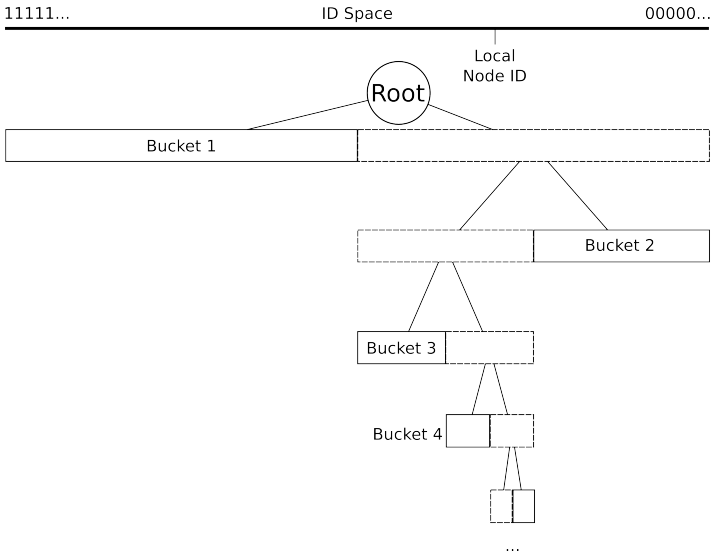


Figure 2.1: The Kademlia tree of buckets

The k-buckets are organized in an unbalanced binary tree, where the n'th bucket stores contacts whose shared prefix with the local node ID is at least n. Each bucket thus covers half of the available ID space: The first bucket covers $\frac{1}{2}$ of the total ID space ($2^{160}/2$), the next node covers $\frac{1}{4}$ of the ID space and so on. The buckets' space does not overlap and together they cover the maximum ID space. Thus, a full Kademlia routing tree would consist of 159 buckets with k contained nodes, one for each possible prefix length. The bucket tree then looks similar to Figure 2.1, where the dashed rectangles represent interior nodes in the tree that do not contain any nodes, and the full rectangles present k-buckets.

Not all of those buckets are created from the beginning. Instead, new buckets are created "on demand": Whenever a new node is added, the target bucket is found using the shared prefix length formula above. If that bucket is not full, the node is simply inserted. If it is full, and if this bucket is the one that contains the local node ID, the bucket is split and the nodes contained within it are reorganized according to their distance (or, equivalently, prefix length). The full insertion algorithm is described in detail in Algorithm 1.

It can be seen from Algorithm 1 that nodes which are still alive are never removed from the buckets. This is of advantage for a stable network, since the probability of nodes staying in the network for a longer time increases the longer those nodes have already been in the network (see [MaMa02]).

Normally, bucket contents are kept fresh (i.e. containing mostly alive and reachable nodes) by regular request traffic. Every time a request is received, all nodes contained in that request are added into the buckets. This leads to regular updates of the node contents, and dead nodes being quickly ejected from buckets. In certain cases, when no lookup is received for a specified zone, bucket refresh is accomplished by simply issuing a $FIND\_NODE$ for a random ID contained in

Algorithm 1 Kademlia: Insertion of nodes into routing table

```
procedure INSERTNODE(startnode, ID)
    length ← 0
    curnode ← startnode
    while not isLeaf(curnode) do
        if ID[length] = 1 then                          ▷ length'th bit of ID
            curnode ← children[1]
        else
            curnode ← children[0]
        end if
    end while
    if contains(curnode, ID) then          ▷ Node seen, move to back of bucket
        moveToBack(curnode, ID)
    else if size(curnode) < k then
        append(curnode, ID)
    else if contains(curnode, ID_L) then          ▷ Bucket contains local node
        split(curnode)
        insertNode(curnode, ID)                              ▷ Recursive insert
    else
        leastseen = getFirst(curnode)                ▷ Least recently seen node
        if ping(leastseen) then                                ▷ Send PING RPC
            moveToBack(curnode, leastseen)     ▷ Node seen, move to back of
bucket
        else
            delete(curnode, leastseen)
            append(curnode, ID)
        end if
    end if
end procedure
```

those buckets. Due to the symmetry of the XOR metric, this leads to the local node node being inserted in remote nodes' routing tables, and remote nodes being inserted into the local routing table. The Kademlia authors recommend that every bucket not accessed for an hour should be refreshed using the above method.

The Kademlia paper also mentions some optimizations to reduce bandwidth requirements for pathological cases and for increasing routing performance. Some of these optimizations will be reviewed in chapter 2.2.1. For more details, see the original Kademlia paper[MaMa02].

2.1.2 Routing in the Kademlia network

P2P networks can be divided into two categories: *Structured* and *unstructured* networks.

In unstructured networks (like Gnutella), storage and network layout is not constrained in any way. While this allows simpler algorithms and higher network performance, unstructured networks do not generally scale as well as structured networks.

Kademlia, however, is a *structured* network: Content is not stored distributed on random nodes in the network, but in a structured fashion. This means that the topology of the network as well as the storage of information is constrained by certain rules.

Kademlia uses the XOR metric described above to impose structure on the network. Keys are stored on nodes whose ID is close (i.e. within a certain distance) to the content's hash value. This eases searching for specific content and greatly decreases search effort since only certain parts of the network needs to be searched. However, it also requires a more complex routing algorithm.

Instead of just the `GET()` and `STORE()` RPCs described in chapter 2.1, four RPCs are defined in the original Kademlia paper: `STORE`, `PING`, `FIND_NODE`, and `FIND_VALUE`. `STORE` is the equivalent of `PUT`, while `FIND_VALUE` is the Kademlia equivalent of `GET`. `PING` is used to check the reachability of nodes, and `FIND_NODE` is used to find nodes instead of keys.

Whenever an RPC like `FIND_NODE`, `FIND_VALUE` or `STORE` is executed, a routing phase is started. For this example, let us assume a `FIND_NODE` RPC has been sent.

The routing phase commences by having the local node select the α closest nodes to its search target from its buckets. α is a system wide concurrency parameter that controls how many queries can be in progress at once. In the Kademlia paper, α is set to 3.

It then sends `FIND_NODE` requests to those α nodes, and adds them to the "best" list (b in Algorithm 2).

When a node receives a `FIND_NODE` call, it selects the k closest nodes to the target from its buckets, and returns them to the querying node.

Then, the repeated routing step starts: When the replies arrive at the searching node, they are inserted into a list of length k, the "possible" list. This list always contains the k closest known nodes sorted by distance. Insertion maintains sort order and the length of the list. Also, the nodes are inserted into the "best" list.

If new contacts have been inserted into the "best" list, the `FIND_NODE` request is again sent to the new nodes from that list.

When a reply fails to return any contacts closer to the target than the contacts currently in the "best" list, nodes from the "possible" list that have not been queried yet are queried. The routing algorithm continues above, when replies are received.

For an overview of the routing process, see Algorithm 2.

This routing step continues until all k contacts in the list have been contacted and have replied or until a timeout has been reached. The list then contains the k closest nodes to the target, and the `FIND_NODE` call is finished.

In the case of a `FIND_VALUE` RPC, the routing proceeds identically. However, when a node has a stored value for the requested key, it immediately returns that value. Note that Kademlia allows caching of values on intermediate nodes on the way.

A `STORE` is executed by first executing a `FIND_NODE` request and then sending a `STORE` request to the k closest nodes found. These nodes then store the key-value pair locally.

`PING` is used for aliveness checks in the routing tree as described in Chapter 2.1.1. This RPC contacts a node in the underlay, and sends the Kademlia ID. In turn, the receiving node sends his ID as a reply. This call is useful for checking liveness of nodes and sharing routing information (when normal traffic is not sufficient).

As seen in Algorithm 2, the routing is iterative and parallel. Iterative routing allows full control over the routing process by having the querying node decide on the next routing step. However, it decreases network performance because the latency of round-trips between queried nodes and the local node adds up quickly.

Parallelism leads to an optimized routing path because nodes that answer quickly are automatically preferred by the routing algorithm. When querying several nodes at once, failed nodes do not impose a timeout on the searching node.

The three RPCs described above (and the `PING` RPC) are sufficient to provide DHT operation. However, due to the loose nature of the network, nodes might leave and join at any time, taking their stored data with them. This is counteracted by storing the information on k nodes with a `STORE` request and periodically refreshing the information by issuing `STORE` requests for all stored key-value pairs. Several optimizations are suggested for the republication mechanism in [MaMa02], as otherwise the traffic caused by republishing would be prohibitively expensive.

2.1.3 Eclipse Attacks in the Kademlia Network

It is clear that the flexibility of this network is also a security risk. With the node IDs chosen randomly, a malicious user can position nodes at arbitrary positions in the network. Also, the lack of a central authentication server (which would defeat the decentralized nature of the system) allows malicious users to introduce several nodes under their control.

This strategy, where an attacker introduces several nodes into the network is called "Sybil attack". The nodes are then called Sybils, and can cooperate to disturb network operations. This allows an attacker attacker to greatly amplify the damage he can inflict on the network. Also, Sybil attacks cannot prevented easily in a network without central authentication, as proven in [Douc02].

Kademlia, as described in the original paper, has no defenses against Sybil attacks. IDs are chosen randomly, and can not be verified to be constructed correctly. The authors suggest that IDs could also be constructed using some information unique

Algorithm 2 The Kademlia routing algorithm. *target* is the search target. Algorithm returns k closest known nodes to *target*.

```
procedure ROUTE(target)
    curnode = startnode
    l = []                          ▷ List of k closest nodes, sorted by distance
    b = []                          ▷ List of α closest nodes, sorted by distance.
    q = []                          ▷ List of already queried nodes
    while not isLeaf(curnode) do
        if ID[length] = 1 then
            curnode = children[1]
        else
            curnode = children[0]
        end if
    end while                       ▷ Find correct bucket
    for i = 0, i < α, i++ do        ▷ Get alpha closest nodes
        send(FIND_NODE, curnode[i])
        insert(b, curnode[i])
        insert(q, curnode[i])
    end for
    insert(l, curnode)              ▷ Add all nodes to list
    finished = false
    while r = receiveReply() and !finished do   ▷ r is list of contacts
        insert(l, r)
        n = insert(b, r)            ▷ insert() returns number of inserted nodes
        if n > 0 then               ▷ Closer contacts were returned
            for j = 0, j < α, j++ do
                if b[j] ∉ q then
                    send(FIND_NODE, b[j])   ▷ Query new nodes
                    insert(q, b[j])
                end if
            end for
        else                        ▷ No closer contacts returned
            for i = 0, i < k, i++ do
                finished = true     ▷ Finished when all k nodes have been queried
                if l[i] ∉ q then
                    send(FIND_NODE, l[i])
                    insert(q, l[i])
                    finished = false
                end if
            end for
        end if
    end while
    return l
end procedure
```

to each host (such as the IP), which would allow verification of node IDs. However, this is not done in the implementations Kademlia and Kad, for various reasons.Constructing the node IDs from IP information, for example, would complicate operations through firewalls and NAT boxes and would disallow node mobility (i.e. changing IP addresses, but keeping the same overlay address)

An attack that can be executed by first introducing Sybils into the network is the eclipse attack. The central idea is described in [SNDW06]: by positioning nodes in suitable locations, a victim can be made to communicate mostly with malicious nodes. Thus, the attacker can control most (or all) communication with the victim node simply by positioning his nodes in certain locations. Also, by returning incorrect or biased neighbor information, a small amount of nodes can efficiently eclipse large sections of the network.

Within Kademlia, there are two possible targets for an Eclipse attack: Either, the content stored in the network is eclipsed, or the nodes themselves are eclipsed.

Eclipsing content in a Kademlia network is relatively easy, given the large ID space. The attack is started by positioning at least k nodes nearer to the target than any other non-malicious node (which should always be possible, given the node IDs are chosen at random). When storing content in the network, the initial node search will eventually return the k Sybils, as described in Chapter 2.1.2. The subsequent `STORE` RPC can simply be dropped, as seen in figure 2.2. This way, only the original node will be storing the data, and it will be lost when that node leaves the network. The only hurdle for an attacker is making sure that his k malicious nodes are found during the search.

This attack works partly due to the fact that responsibility for storing values is passed to the k closest nodes to the target. Neither successful storage and availability nor the integrity of the data is checked.

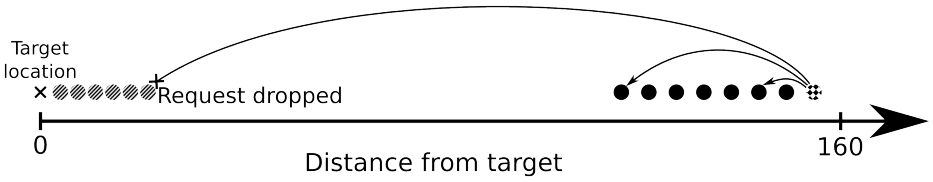


Figure 2.2: Eclipsing content: eclipse nodes positioned near the target hash value

Eclipsing nodes, however, poses more difficulty. In this case, the target is a node ID instead of a content key. When trying to eclipse a node with ID ID_T, the goal is to never see the node with ID_T appear in the search results list. This means the node with the ID ID_T will not receive any traffic from the network. However, it can still send queries normally.

Eclipsing a node is nonsensical from a pure DHT standpoint, since not finding a node is the same as not storing or retrieving information on it. Node IDs are only used to correctly distribute content over the network, and no RPC calls operate on node IDs (the `FIND_NODE` call is only used for storing and retrieving data). However, when using Kademlia to implement non-DHT overlays (such as a P2P routing overlay mechanism), eclipsing a node has great influence on the results returned by the network.

The basic pattern of an eclipse attack on nodes remains the same as when attacking content: the attack is executed by again positioning the malicious nodes around

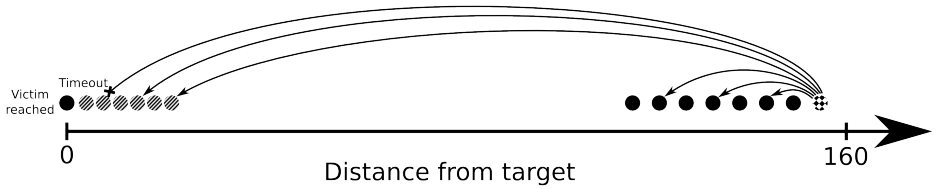


Figure 2.3: Eclipsing nodes: By not returning the victim, the search fails with a timeout

the target. The Sybils are then configured to return other nodes, but not the target node, to the querying node. This leads to the process seen in Figure 2.3: the Sybils are reached, but do not return the victim. After a time, the lookup fails with a timeout error. Only looking at the routing algorithm, this seems to be sufficient to eclipse nodes.

However, the nature of the Kad routing algorithm makes this attack more difficult than expected. Whereas content is only stored on the k closest nodes to the target, nodes are stored in other nodes' routing tables. Since the target node is not suddenly removed from these routing tables (and it is still reachable in the underlay with a `PING` request), regular nodes that knew about the target before node still do during the attack. If the routing algorithm should ask a node that knows about the target node, it is immediately found (as in Figure 2.4). Without control over the underlay network, there is no possibility to remove nodes from the k-Buckets. It is thus the goal of an attacker to keep the searching node querying malicious nodes for as long as possible.

Also, whereas content is "inactive", a node is an active part of the network that can send and answer requests. A node regularly sends out requests and receives replies as part of regular protocol operation. This means that it is added into other buckets, which again increases the chances of being found. The presence of a node can always be verified by sending a `PING` RPC (if its underlay contact data is known). If content is lost from the network by the actions of a malicious attacker, this can only be noticed by trying to retrieve that data.

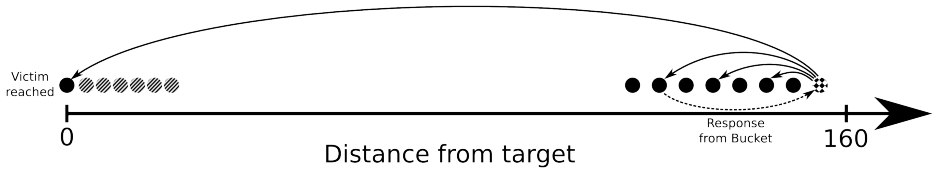


Figure 2.4: Failing eclipse attack on node

Another difficulty when eclipsing nodes is the preference for "old" nodes: recall that in the routing algorithm described in Chapter 2.1.2, newly found contacts only replace old contacts when the old contact is no longer reachable. In order for the malicious nodes to receive a large amount of search traffic, they first need to be added into remote buckets. With the aforementioned restriction, this is not easy. While the malicious nodes may appear in the search path, they will not be added into the buckets of remote nodes.

However, the publicity of Sybils in the network is a very important factor in attack success. The better known the malicious nodes are, the higher the probability that a Sybil will be returned instead of the target node.

The final factor hindering an attack is the iterative nature of the Kademlia routing algorithm. With recursive routing, a query can simply be dropped by malicious nodes. The querying node receives no notification of this, and all it can do is resend the query (which will most likely reach the malicious node again, and is dropped again). With iterative routing, a node keeps control over the routing process at all times. Instead of simply dropping requests, the queries must be misdirected.

All these complications stem naturally from Kademlia design choices. While the authors mention the fact that the routing algorithm resists "certain basic denial of service attacks"[MaMa02], note that security was not a primary design goal in Kademlia.

2.2 The Kad implementation

Kademlia implementations are used in a variety of file sharing networks as additional serverless components. For example, most BitTorrent clients use a Kademlia implementation (called "mainline DHT") to share sources without a central coordination server, and Gnutella uses Kademlia to share additional file sources. eDonkey uses a Kademlia implementation called "Kad". All these implementations of Kademlia are incompatible. Since the eDonkey network is the largest P2P networks existing today, we decided to use Kad for evaluating impact of the eclipse attack.

Kad is used in the eDonkey network for serverless operations. It supports operations such as requesting and storing file sources, retrieving bootstrap nodes from other nodes and searching for certain file properties such as file name, size and type. Kad has been in use in the eDonkey network for a comparatively long time (since about 2004), and two versions of the Kad protocol exist. The most popular client is called eMule and its source has been available on-line since 2002. However, for this experiment, aMule was chosen. aMule and eMule both use the same Kad implementation, but aMule is tailored to run on Unix and Linux systems.

Since the eDonkey network has at least one million users at all times (and more than 2 million users most of the time) according to [Ed2k11], and almost all eDonkey clients support Kad, this makes Kad one of the largest publicly accessible DHTs today. Since a large network size is one of the primary motivations for using a real network, this makes Kad a good subject for this work.

2.2.1 Differences to Kademlia

The integration of Kad with eDonkey clients has a slew of consequences which require changes in the core Kademlia protocol to make it suitable for large-scale file sharing operations.

Two versions of the Kad protocol exist that are both currently used in the network. Kad 2 is used to communicate with newer clients, while Kad RPCs are used to interact with older clients. Since this work uses a modified aMule client and not a new implementation of Kad, no special precautions needed to be taken to ensure compatibility with both protocol versions. Both versions are used transparently and interchangeably, and do not influence the experiment results. While Kad 2 has several new features (such as the cryptographic verification of IDs and a three-way handshake), the executed experiments are not influenced by these new features.

First, the length of IDs is different. Instead of 160 bit SHA-1 hashes, 128-bit MD4 hashes are used. This is due to the fact that eDonkey uses MD4 hashes in the

server-based network. This does not require computation of a second hash value, which allows easier integration in the centralized eDonkey network. Also, it avoids having to compute a second set of hashes for every file, a potentially expensive operation.

Also, 128 bit numbers are better suited to fast computation (since they can be handled as 2 64-bit or 4 32-bit integers). Decreasing the length of the hash does not significantly change the network performance: With 2 million participants and 300 million shared files (numbers taken from [Ed2k11]) collision probability is still negligibly small ($\frac{302*10^6}{2^{128}} \approx 8.87 * 10^{-31}$).

Since Kad node IDs are still chosen randomly as in Kademlia, the assumptions about distribution of nodes over the ID space from Chapter 2.1.1 still hold. With a size of about two million peers, the distance between adjacent nodes is about $2^{128}/(2 * 10^6) \approx 3 * 10^{32} \approx 2^{108}$. This means that on average, the closest client to a randomly chosen node will have the first differing bit at position 108. While badly written clients and incorrect configurations do introduce certain discontinuities into the network (see [StENB07a]), we will disregard them for the purpose of this work. Also, the average prefix length two nodes share calculated by $\int_0^{128} n * 0.5^n \mathrm{d}n$ again gives an average prefix length of 1.67 bits.

Another change concerns the routing tables. While the Kademlia routing table is indexed by the ID of the nodes, the Kademlia table is indexed by the distance of the nodes. This makes it easier to find the correct bucket during the routing process.

Also, the tree is more branched than the Kademlia bucket tree: The first four levels of the tree are always split, and the buckets in deeper levels are split when the prefix taken in the tree to reach that bucket is numerically less than 6. This results in a tree looking like Figure 2.5, where double circled nodes will no longer be split. Compare this to the Kademlia tree in Figure 2.1. Stutzbach et al. call this approach “Discrete Symbols” in [StRe06]. The added buckets greatly decrease the routing path length, and lead to an average path length of only 3 hops in the Kad network.

The insertion algorithm is essentially the same as Algorithm 1, except for the split condition being different. Instead of only checking if the local node ID is contained in the bucket, the numerical value of the prefix is tested and the depth in the tree is tested. If the numerical value of the prefix is less than six, or the depth in the tree is less than five, the bucket is split.

Also, some filters are used to prohibit the addition of certain nodes into buckets (for example, PeerGuardian block lists, or bogus IP addresses). Bucket refresh is done with special Kad RPCs that will be described in a later section.

Also, nodes are saved to a file on shutdown of the client. While this is more a feature of the client implementation than of the protocol, it is relevant for evaluating attacks on joining clients. On quitting the Kad client, all buckets of level 5 or less are saved, plus one bucket chosen randomly. Looking at figure 2.5, this makes up to 170 saved contacts. These contacts are used to refill the tree when joining the network again. Since the Kad ID is also persistent, these cached nodes will fill up a large percentage of the bucket tree (which will also contain many nodes that are no longer in the network). For a comparison, see Figure 2.6. The labels in the nodes describe the branch taken in the parent node. Double circled nodes are buckets, and their label describes the number of contacts contained in the bucket. It is

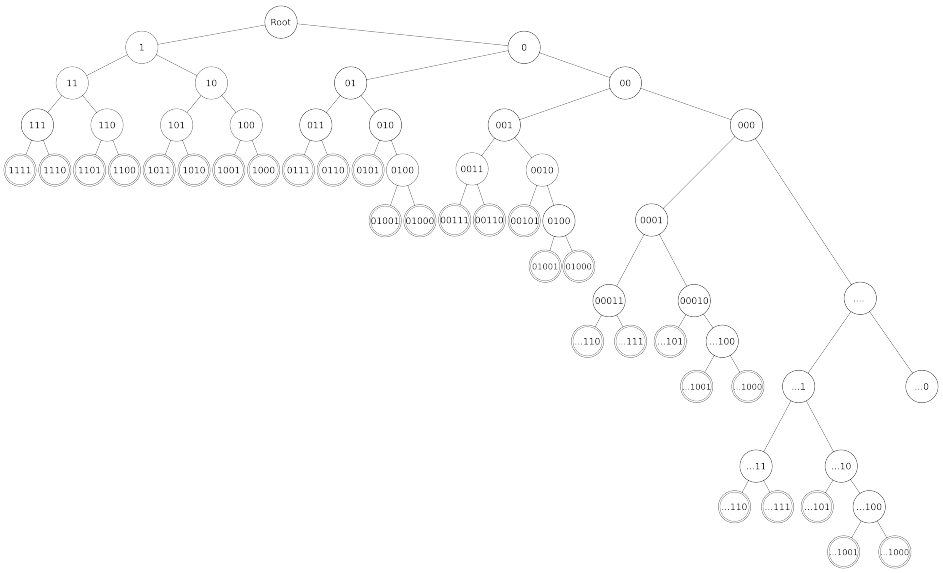


Figure 2.5: The Kad bucket tree. Labels in the nodes indicate the ID prefix. Double circled nodes are k-buckets.

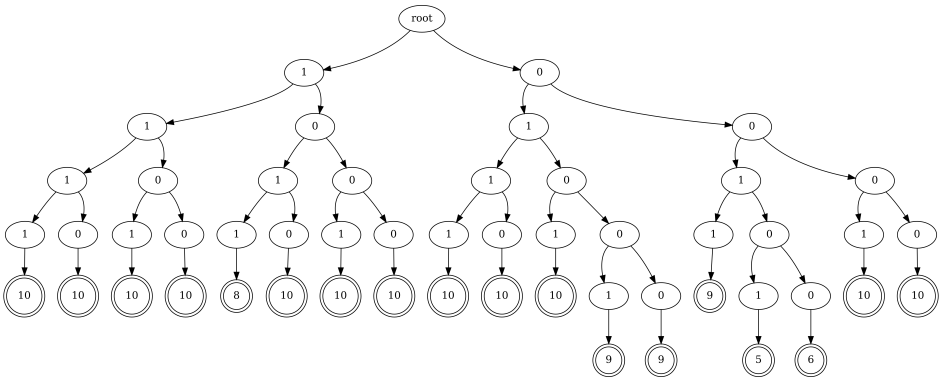


Figure 2.6: Saved nodes in the Kad tree. Double circled nodes are k-buckets, the number in the k-buckets shows the number of contacts contained in the bucket. Figure was created from the bucket dump of a freshly started Kad client.

immediately apparent that the first four levels of the tree are almost filled. This greatly decreases the time after which the new node is reachable in the network, and allows a fast integration into the network when joining.

Kademlia as described in the preceding chapters is tailored towards being a DHT. However, Kad in eDonkey is used as a file sharing protocol. Also, the extended functionality provided by Kad for searching files and tunneling through firewalls and NATs requires a variety of API calls instead of only the ones proposed in Kademlia.

The Kad API is realized through different “search types”: When starting a Kad operation, a new search is started. This search issues different RPCs over the network. Each search has its own lifetime and maximum results, and maps to different kinds of network RPCs.

There is a wide variety of client API calls that result in different network RPCs. For easier classification, we have divided these RPCs into three types: DHT, administrative and basic. DHT RPCs are RPCs that execute a node search beforehand

(like STORE in the original Kademlia implementation). Administrative RPCs are RPCs that require no DHT interaction (like the PING request). Finally, basic RPCs are the RPCs that are used for basic protocol interaction, like FIND_NODE. For a short description of these different RPCs, see tables 2.1 and 2.2. These RPCs are used by different searches on the client side (i. e. Kad API calls). For a short description of available Kad client side search types, see table 2.3. To explain the use of these RPCs and search types, a file search will be explained in more detail.

A file search in Kad is executed in two stages: First, the file hash is found by searching for keywords or other properties, and then sources for the file hash are retrieved with a second search.

The first search is a so-called keyword search. The search is started when the user searches for a file name. Then, a KEYWORD search is started in the client for the entered keywords. The keywords are hashed to a key, and routing for finding this hash begins. Using KADEMLIA_REQ/RES requests, the closest nodes to the hash are found. Then, the SEARCH_KEY RPC is sent to these nodes, and the nodes return stored file hashes for the keywords.

Then, the user selects a file from the results list, and a source search with the search type FINDSOURCE is started. Again, using KADEMLIA_REQ/RES RPCs, the closest nodes to the hash are found, and then a FIND_SOURCE RPC is sent to these nodes. The nodes then return sources for the file, and the client starts downloading the requested file from the found sources.

Despite the extended API, the core routing algorithm stays the same. Iterative routing is still done. However, RPC calls are interleaved. Recall that on a STORE in Kademlia, a node search is done, and then the STORE RPC is sent to the k found nodes after the node search is finished. To differentiate this "store" phase from the first "routing" phase, we will call it phase 2, and consequently the routing phase will be called phase 1. In Kademlia, phase 1 and 2 happen sequentially: First, the node search is executed, and then the store RPC is called. In Kad, the two are interleaved: If no response is received to a search for 3 seconds, the query is "jumpstarted": This means that the closest nodes that that have not yet been contacted are queried, *and* a "store" type RPC is sent to the current closest target that has already been queried. The search then continues until enough "store" RPCs are sent or a timeout has been reached. This "jumpstart" behavior will become important later on when executing an attack.

2.2.2 Eclipse Attacks in the Kad network

Executing an eclipse attack in the Kad network is not much different from eclipsing content in Kademlia in principle, and eclipse attacks on content were executed by Steiner et al. in [StENB07c] for Kad.

The attack by Steiner followed the description on eclipsing content in Chapter 2.1.3. Positioning 32 Sybils close enough to the target value, and then dropping incoming store requests made it possible to eclipse arbitrary hashes in the network with very low bandwidth requirements.

The clients that executed the attacks were all run on the same host. This was possible because Kad clients had no protection against Sybil attacks at that time. However, more recent versions (since eMule 0.39c) have integrated a defense known as subnet guard: whenever a node is added to the buckets, several conditions are checked. These conditions are:

Name	*Type*	*Description*
BOOTSTRAP	administrative	Asks nodes to return a random selection from their buckets
HELLO	administrative	Used to check whether nodes are alive
SEARCH	DHT	Asks node to return keyword (i.e. file names) matches and sources
PUBLISH	DHT	Asks node to store keywords and file sources and matches
SEARCH_NOTES	DHT	Asks node to return stored notes for keyword
PUBLISH_NOTES	DHT	Asks node to store notes for keyword
FIREWALLED	administrative	Used for firewall tunneling
FINDBUDDY	administrative	Used to find a firewall tunneling partner
KADEMLIA_RES	basic	Basic routing primitive request
KADEMLIA_REQ	basic	Basic routing primitive response

Table 2.1: Kad 1 RPCs, from [aMul09], `src/kademlia/net/KademliaUDPListener.cpp`

Name	*Type*	*Description*
BOOTSTRAP	administrative	Asks nodes to return a random selection from their buckets
HELLO	administrative	Used to check whether nodes are alive
SEARCH_KEY	DHT	Asks node to return keyword (i.e. file names) matches
SEARCH_SOURCE	DHT	Asks node to return sources for keyword
SEARCH_NOTES	DHT	Asks node to return stored notes for keyword
PUBLISH_NOTES	DHT	Asks node to store notes for keyword
PUBLISH_KEY	DHT	Asks node to store keyword matches
PUBLISH_SOURCE	DHT	Asks node to store sources for keywords
FIREWALLED	administrative	Used for firewall tunneling
FINDBUDDY	administrative	Used to find a firewall tunneling partner
PING	administrative	Check the “aliveness” of a contact.
KADEMLIA_RES	basic	Basic routing primitive request
KADEMLIA_REQ	basic	Basic routing primitive response

Table 2.2: Kad 2 RPCs, from [aMul09], `src/kademlia/net/KademliaUDPListener.cpp`

- Is there a node with the same ID, but a differing IP address in any bucket?
- Are there more than 10 IPs from the same /24 subnet in all buckets?
- Are there more than 2 IPs from the same /24 subnet in any bucket?

Whenever one of these conditions is true, the node is not added to the buckets. This protection greatly decreases the impact Sybils can have on the network, and makes Steiner's attack harder: each of the 32 Sybils needs a different IP address, and at least 20 have to be from different subnets.

Also note that Steiner attacked content. As mentioned previously, attacking content is easier, since responsibility for storing key-value pairs is essentially passed to the other nodes.

After the release of eMule 0.49a, Kohnen et al. executed a refined Sybil attack that was able to bypass the new filter infrastructure [KoLR09]. This is accomplished by introducing many Sybils, but making only one Sybil (the "Master Sybil", or Sybil 0) advertise itself in the network. This Sybil is also the Sybil farthest away from the target. All other Sybils are positioned between the master Sybil and the target. This layout is shown in Figure 2.7, where S_0 is the only Sybil actively advertising itself in the network.

The prefix routing still causes Sybil 0 to receive routing requests for the target. However, now Sybil 0 returns another Sybil (the next closer one) and a fake node (This is required because Kad expects two nodes in the response). The next Sybil is then the closest node, and will again be queried. To disrupt the publishing process (such as `PUBLISH` or `PUBLISH_NOTES`), this continues until 10 Sybils have been returned. The searching node then sends store requests to the found Sybils. The the requests can simply be dropped, and the publish process is disrupted.

This strategy works because Kad does not check nodes that are selected in the search against its buckets. This is probably a bug, and fixed easily enough. However, in the current aMule version (2.2.6), this has not been done yet.

When Kad searches for file sources (with `SEARCH` or `SEARCH_KEY` requests), simply returning 10 Sybils is not enough. Depending on the client side search type, different timeout and maximum result values can be used as seen in Table 2.3. In the case of `SEARCH` or `SEARCH_KEY`, Kad continues the search until either 300 results have been found or the search has run for 45 seconds. Returning 300 fake results or sending garbage files would quickly alert the user.

So, a timeout must be forced instead. The Sybil chain is constructed as above. Now, however, a delay is introduced between receiving a request and answering. When the chain is long enough, the timeout is reached, and the search is stopped.

Then, a timeout can be forced with a low number of Sybils. This can be seen in Figure 2.7: one Sybil returns the next, and the cumulative delay finally forces the query to be aborted.

With several other modifications that depend on the Kad code edge cases, Kohnen managed to disrupt the source search mechanism for various targets with less than 10 Sybils.

However, these attacks were all targeted on eclipsing content, i.e. stopping Kad from storing or retrieving a certain target key. This work focuses on exploring

Name	*Lifetime*	*Used RPCs*	*Description*
KEYWORD	45 s or 300 results	SEARCH, SEARCH_KEY	Regular keyword search, i.e. find file hashes for keywords
NOTES	45 s or 50 results	SEARCH_NOTES	Search for notes
FINDBUDDY	100 s or 10 results	FINDBUDDY	Find buddy for firewall tunneling
FINDSOURCE	45 s or 20 results	SEARCH_SOURCE, SEARCH	Find sources for file hash
NODE	45 s or 1 result	KADEMLIA(2)_RES	Find new entries for routing table. Queries 50 nodes at once
NODECOMPLETE	45 s or 10 results after 10s	KADEMLIA(2)_RES	Used for bootstrapping and bucket refresh
NODESPECIAL	45 s or target found	KADEMLIA(2)_RES	Used for special client side requests.
STOREFILE	140 s or 10 results	PUBLISH_SOURCE, PUBLISH	Stores sources for file hash
STOREKEYWORD	140 s or 10 results	PUBLISH_KEY, PUBLISH	Stores files matching keywords
STORENOTES	100 s or 10 results	PUBLISH_NOTES	Stores notes

Table 2.3: Kad client search types, data taken from [aMul09], `src/kademlia/kademlia/Defines.h`

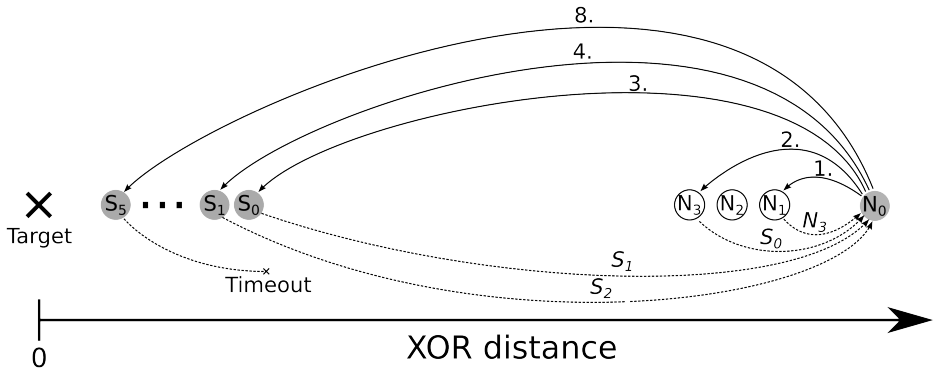


Figure 2.7: The Sybil chain forcing a timeout

eclipse attacks on nodes, which entails several difficulties as described in chapter 2.1.3.

This has been evaluated in a simulation by Boie in [Boie09]. The attack executed was the regular Sybil chain introduced above: Several Sybils, with only one being published in the network, try to force a timeout on all node lookups for a special victim node. This leads to the victim becoming unreachable for all Kad RPCs.

While Boie and Kohnen were able to consistently execute successful attacks, attack success is much lower when eclipsing nodes in the network. Depending on churn, network size and activity of the various nodes, a maximum of 44% of disrupted searches was reached.This was in a small network of 500 users with a relatively high churn. All other attacks were markedly less successful. Parameters with a large influence were churn and network size, both of which cannot be influenced in the Kad network.

These results indicate that the differences between node and content attacks mentioned in chapter 2.1.3 have a large influence on the success possibility.

3. Related work

[MaMa02] describes the original Kademlia implementation. The basic elements of Kademlia, such as the XOR metric and its advantages, as well as the iterative routing algorithm, have survived in Kad almost unchanged. However, the Kad implementation differs in several points that are important for executing an eclipse attack.

Most importantly, the layout of the bucket tree has changed. The routing table optimizations used by Kad are described in [StRe06]. The information contained in these sections was confirmed using the aMule source code [aMul09] and its Kad implementation (which is identical to the eMule implementation).

The structure of the network and several of the network conditions, such as size, distribution of IDs and node characteristics have been researched by Steiner in [StENB07b] and [StENB07a].

The initial eclipse attack executed by Steiner ([StENB07c]) made use of the fact that Kad allowed running several protocol instances on one host. This, combined with Kad delegating source storage, allowed eclipsing content with very low resource requirements. Running 32 Sybils on one host allowed even eclipsing a very popular keyword such as “the” with low incoming bandwidth. The attack was basically executed as described in Chapter 2.1.3: By positioning enough Sybils close to the keyword, all queries eventually ended on malicious nodes. The queries were dropped, and the content lost.

An important difference from our work is that Steiner attacked the source search mechanism, and our work evaluates eclipse attacks on nodes. As described in chapter 2.1.3, disruption of the publishing mechanism is very easy if enough Sybils are available. Also, the introduction of subnet filtering as described in chapter 2.2.2 into Kad (as of eMule version 0.49/aMule version 2.2.1) severely limits the practicability of this attack. While the basic principle of the eclipse attack does not change, a large amount of Sybils and IP addresses would be required to circumvent the blocking behavior.

Kohnen et al. in [KoLR09] circumvented the new protection mechanisms by using the concept of a Sybil chain (described in Chapter 2.2.2). This allowed eclipsing content despite the address restriction placed on Kad nodes. It was shown that

content publication can be disrupted with as few as 10 nodes under malicious control in the network. However, not only content publication was disrupted: The Sybil chain was also used in order to force a timeout on other RPCs (namely `SEARCH`). This was also possible using less than 15 Sybils in most cases.

Several optimizations used by Kohnen are not applicable when attacking nodes. Reversing the Sybil chain had a large impact on the number of required Sybil for a successful attack, because it led to the search process repeatedly transitioning between phases one and two of the routing algorithm (see Chapter 2.2.1). This made it possible to reach a timeout much earlier. The number of required Sybils was reduced by more than half compared to the unreversed chain. However, when attacking nodes, phase 2 of the routing algorithm is not used. Since no "store"-type RPCs are sent, phase 2 is never entered, and thus the effects of a reverse chain do not apply.

Kohnen and Boie executed their attacks in the real (i.e. not simulated) Kad network with Kad client implementations. Their aim, however, was to attack the data stored in a network. This was done by positioning the nodes to be responsible for the data, and then discarding all store requests as described in Chapter 2.1.3. This approach is not feasible for executing an eclipse attack on nodes, since node responsibility is not passed to other nodes. However, the second method used by Kohnen, forcing a timeout during the RPC call, is useable for attacking a node. This is because it relies not on the publication mechanism of Kad (i.e. publishing data on the 10 closest nodes in the network), but attacks the lower-level routing algorithm.

Boie used the findings by Kohnen to execute an eclipse attack on nodes in [Boie09] in a simulated network. Again, the Sybil chain mechanism from Kohnen is used.

Boie had markedly less success on eclipsing nodes than both Kohnen and Steiner: A maximum of 45% of searches for the victim were ended with the victim not being foun despite it being in the network. This was with the most favorable network configuration. Success decreased markedly with changing network parameters. Increasing the network size and regular node activity led to decreasing attack success. Also, increasing churn increased attack success.

The simulations executed by Boie used the Kademlia implementation of the OverSim [BaHK07] network simulator. Many differences exist between the OverSim Kademlia implementation.

Also, Boie's attack was only simulated, and this prohibits transferring any results to the real Kad network. While simulation is an easy way to create an environment to evaluate the impact of an attack, it also differs from the real network in several important aspects. First, the simulation implementation (a Kademlia implementation from the Omnet++ project) differs from the actual Kad implementation used in clients the same way the standard Kademlia differs from Kad (see Chapter 2.2.1).

Also, the parameters of the simulated network may greatly differ from the parameters of the real network. While almost all parameters of the simulated network are controllable, parameters in the Kad network are not controllable, and also measurable only with high effort. Important factors that are modifiable in the simulation and have great influence on attack success are not modifiable in the real network. Two parameters are churn and network size, both having high impact on attack success in Boie's work. Also, the knowledge of regular nodes about the Sybil nodes, an important parameter for a successful attack, is not easily measured.

The cluster manager used for coordination of attacks in the paper uses TCP/IP sockets for communication, and is easily adaptable for use in the Kad network.

4. Framework Setup

To execute experiments in the Kad network in a repeatable fashion, a standard experiment layout was devised. This layout allows repeatedly executing experiments and gauging the success of the attack. The layout is described in detail in Chapter 4.1.

Because no standard framework exists for executing experiments in the eDonkey Kad network, the code of a eDonkey client had to be modified to allow instrumentation of the network, and control of the search mechanism. These modifications are described in Chapter 4.3.

4.1 Experiment Setup

The use of Kad as a secondary routing layer for TCP/IP networks serves to increase robustness and flexibility of these networks. A common attack against hosts is making their services unavailable to clients, a Denial-of-Service (DoS) attack.

In a traditional TCP/IP network, this is easily done by either disabling the host providing the service, or making that host unreachable by disrupting intermediate servers (for example, gateways or central routers). Then, the victim is no longer reachable for clients.

When using a secondary P2P routing layer, the disruption of intermediate nodes no longer plays a large role: Because of the multitude of available paths, a different routing path can simply be selected when the current path fails. The victim is then still reachable by clients.

An eclipse attack negates that advantage, since the routing paths in the overlay network can be easily disrupted as well. So, the resistance against eclipse attack is important for the use of a P2P network as secondary routing layer for TCP/IP networks.

To evaluate the feasibility and impact of an eclipse attack, a setup containing three different types of nodes is used: A tester node, a victim node and several malicious nodes.

First, the tester is responsible for testing the reachability of the victim. In the attack scenario above, the tester is an arbitrary node evaluating the reachability of

the victim. This is done by periodically searching for the victim and determining if a result is returned. This information is logged to allow evaluation of attack effectiveness. Several experiment parameters can be modified on the side of the tester: The location (i.e. distance from the victim) of the tester may have impact on the search results, and can be easily varied. Also, the interval at which victims are searched for can be adjusted.

The victim, on the other hand, is the attacked host. The aim of the attacker is making the victim unreachable for the tester node. The victim participates normally in the network and has a configurable ID. As in all other nodes, useful request information is logged. Also, activity of the victim is configurable: this means that the interval and amount of random queries that are sent to the network can be adjusted. In regular intervals, a configurable number of queries for a random key is started. This feature was not used in the experiments, however.

Finally, the malicious (or "Sybil" nodes) are the nodes that actually execute the attack. They are configurable for different delays between the routing steps and also configurable for different amounts of activity, as above.

In addition, the Cluster Manager is used, a centralized component for sharing information between nodes taken mostly from Boie's work [Boie09]. The cluster manager calculates the positions of the malicious nodes and the victim node, and calculates the next Sybil chain element for each node. Different Sybil distributions can be implemented in the cluster manager.

Note that these experiments use only one tester node, i. e. one node searching for the victim. This means reachability of the victim is only checked from a single node. To evaluate the reachability of victims from arbitrary nodes, more tester nodes would need to be used. However, due to technical and time constraints, reachability of the victim from the rest of the network is not tested in this work.

Host name	*IP address*	*Role*
vz198.rz.uni-frankfurt.de (ffm)	141.2.38.198	Tester
olbia.net.in.tum.de	131.159.15.6	Victim
guantanamo.net.in.tum.de	213.239.199.116	Malicious
istrukta.net.in.tum.de	131.159.14.169	Malicious
lipari.net.in.tum.de	131.159.14.104	Malicious
thenybble.de	83.169.39.92	Malicious
pisa	134.2.172.133	Malicious
alcatraz	131.159.15.50	Malicious
stromboli	131.159.20.56	Malicious
vulcano	131.159.15.55	Malicious

Table 4.1: Nodes used in experiments

The nodes used are shown in table 4.1. Note that apart from olbia, alcatraz and vulcano, no addresses are used that can potentially trigger the subnet protection.

The fact that these three hosts are in the same subnet is unlikely to influence experiment outcome, since the victim is connected to the network first. This makes it likely that the insertion of one of the malicious nodes into the routing table is prevented by the subnet guard. This does not significantly change the experiment outcome.

4.2 Experiment execution

With these prerequisites, an experiment is executed as follows: First, the cluster manager is started. It outputs a victim ID, which is chosen randomly, and the different Sybil IDs. These IDs are calculated according to algorithm 3:

For Sybil ID i, the $100/n * i$th bit of the victim ID is flipped, where n is the number of Sybils. Since the closest regular node has a common prefix length of 20 bit with the victim, this means all resulting IDs have a lower distance to the victim than any legitimate node. This algorithm leads to the Sybils being distributed logarithmically over the available ID space.

Algorithm 3 Calculation of Sybil IDs

```
procedure CALCULATEIDS(targetID, nrSybils)
    distance ← ⌊100/nrSybils⌋
    result ← []
    for i ← 0, i < nrSybils, i++ do
        n ← i * distance
        newkey ← targetID ⊕ (1 << (n − 1))          ▷ Flip nth bit
        result[i] ← newkey
    end for
    return result
end procedure
```

The tester ID is then configured to have a shared prefix of 8 bits with the victim. The rest of the ID is chosen randomly. This is significantly closer than the average calculated in Chapter 2.1.1, but enough nodes exist between the tester and the victim to allow an attack (as shown in the preliminary experiments in Chapter 5.1.2).

The victim is started, connects to the network, and is left running for 10 minutes. This gives the victim time to properly connect to the network. This time is verified in a preliminary experiment in Chapter 5.1.1.

Then, the tester and the malicious nodes are added into the network. The tester then starts regularly querying the network for the victim key. This is used to simulate normal search activity with the victim as target. The results of these searches will be used to analyse the results of the attack later.

The malicious nodes are then responsible for executing the actual attack: whenever a query for the victim is received, the next Sybil is retrieved from the cluster manager (and cached), and returned to the requesting node after a short timeout. This effectively implements the attack mechanism of the Sybil chain described in Chapters 2.2.2 and 3.

Note that all nodes are active in the network. This is a difference to the setup described in Chapter 2.2.2 by Kohnen, where only a single Sybil (the master Sybil) is active in the network. However, because enough addresses were available, all Sybils were kept active. This increases the publicity of malicious nodes in the network, and allow higher attack success. How the use of inactive Sybils influences attack success was not evaluated, and needs to be researched in another work.

The experiment is then run for a certain time. Most basic attack experiments were run for one hour, except when noted. Since the bootstrap phase takes only 10

minutes, and searches normally terminate very quickly, this time should suffice to evaluate success of an attack. Also, longer experiments with a duration of 18 hours were run, in order to estimate if the eclipse attack is sustainable for a longer period of time.

After the experiment time is over, all programs are stopped, and the log files are retrieved from the hosts.

Analysis consists of applying several analysis tools: plotting the search process using gnuplot and a visualization script is used to visually gauge the success of the attack and detect recurring patterns in the search process. Also, several other tools analyze the log file and allow calculation of statistics such as the length of searches, the average number of routing hops and the number of successful and failed searches.

4.3 The code

In order to execute eclipse attacks in the Kad network, the Kad implementation of aMule was extensively modified. Logging of the Kad protocol operation was implemented for analysis of the attack, and is described in chapter 4.3.1.

Several modifications were made to enable the tester node to correctly search for the victim. These modifications are described in detail in chapter 4.3.2.

Finally, the malicious Sybil nodes were modified to implement the Sybil chain mechanism used by Kohnen. These modifications are described in chapter 4.3.3.

4.3.1 Protocol Operation Logging

Firstly, logging for the basic Kad requests was implemented. For this, the basic Kad packet handling was modified in order to log the following RPCs on package level: `KADEMLIA_RES`, `KADEMLIA_REQ` and the corresponding Kad 2 RPCs `KADEMLIA2_REQ` and `KADEMLIA2_RES`. For each, the search target, source IP and port is logged, as well as the contacts contained within the RPCs. These contacts are logged with their target Kad ID, IP and source port.

Note that the basic Kad RPCs do not contain the ID of the source node. To find the ID of that source, post-processing of the log files is needed. For that, the sending of `KADEMLIA(2)_REQ` packets is logged, since at sending time, the target ID is known. The IP-ID combination is then read from the log file and used as a mapping for received Kad requests and responses. Note that an IP/ID combination is normally not unique. The UDP port of a contact would make this combination unique. However, almost no duplicated mappings were encountered during the experiments.

Also logged is the start and end time of a search: This allows to find search timeouts and successful searches that stem from successful or unsuccessful attacks. The search target is logged to allow correlation with the RPCs above.

Several tools use the logging data to allow analysis of attack progress. First, a visualization for search packets was written. It displays the contents of `KADEMLIA(2)_RES/REQ` packets with the included nodes, their status (malicious/regular) and the source of the packet. This allows optimizing attacks and finding reasons for failing attacks (like delay between routing steps being too high, or non-malicious nodes being queried).

Also, a search length visualization was created. It calculates the length of a search and a running average of search lengths, and displays them in a graph. This allows a quick analysis on how effective the attack run was.

Also, the actual attack code had to be implemented. While most of the actual attack code is encapsulated in a single source file (`src/eclipse.cpp`), which was taken from Boie[Boie09] and modified, the interface to the aMule client had to be implemented. For this, the patches implemented by Boie were used and modified to suit the new aMule version.

4.3.2 Tester Node Modifications

On closer inspection of the Kad code, several corrections and improvements proved to be necessary to execute an optimized attack. In particular, the RPC used for finding the victim node from the tester node turned out to not function like expected. Both the Kad client search types `NODE` and `NODECOMPLETE` terminate very quickly, and thus are not suitable for finding a certain node (see Table 2.3). They are normally used to periodically refresh the routing table (as described in Chapter 2.2.1), and as such are not suitable for finding specific targets. `NODECOMPLETE` is used in the bootstrapping phase to determine when Kad is connected to the network, and exits after either 10 answers have been received after 10 seconds, or after 45 seconds. When the search finishes, the Kad implementation is treated as "online" in the aMule code (i.e. ready to publish data and ready to receive published data). The `NODE` RPC queries a large amount of nodes at once, but exits within 15 seconds after the first reply packet has been received. It is used for quickly finding new nodes to put in buckets when the bucket contains less then 8 nodes.

Because both of these RPCs exit quickly in certain circumstances, they are not optimal for executing an eclipse attack. So, the aMule patches were changed to use the `NODESPECIAL` search type for finding the victim. This search is intended for special, non-Kad operations by the developers and thus fits exactly with what we want to accomplish. Note that this call is not used for any protocol operations in the standard client, and thus does not influence the network in any way. Also note that the `NODESPECIAL` searches for a node explicitly, i.e. not as a preparation for executing a different RPC, like `PUBLISH` or `FIND_SOURCE`. This makes an attack harder, since the search only stops when the victim has been found or the timeout has been reached.

This method also allows easy logging of search success or failure, since a callback is called to indicate finding of the target node or a timeout.

4.3.3 Malicious Nodes Modification

Also, several modifications were made to the malicious node behavior. First, since the Sybil chain is static and does not change over the course of the experiment, the next Sybil can be cached. When a request for the victim is received for the first time, the next Sybil in the chain is fetched from the cluster manager and saved. This decreases the time necessary for getting the next Sybil, and allows more precise configuration of the timeout value.

This is important while executing an attack, since the jumpstart mechanism starts contacting additional nodes when no answer is received from any node within 3 seconds. Precise calculation of the delay allows a better utilization of this three second timeout in order to force a search timeout as soon as possible.

Also, the delay on receiving a `KADEMLIA_RES` was made adjustable in order to easily execute several experiments with different Sybil delays. The effect of the Sybil delay on attack success is relatively clear: if the Sybil delay is too short, the timeout will not be reached with the available number of Sybils. If it is too long, other non-malicious nodes will be queried. It is interesting how close to the three seconds the timeout can be configured without introducing other nodes into the search.

The Kad code was also modified to implement facilities for writing out the contents of buckets to a file. In adjustable intervals, all nodes in the bucket tree are logged with their position in the tree and their contact data (IP and ID). Since the contents of the buckets are a major influence on the routing path, this allows better diagnosis of the attack results.

5. Experiments

During the course of this work, several experiments were executed. First, to confirm the boundary conditions of the network, preliminary experiments were executed. They are described in chapter 5.1.

With the findings from the preliminary experiments, attack experiments could be executed with settings that enabled a successful attack. These attack experiments and their results are described in chapter 5.2.

5.1 Preliminary Experiments

Before executing the attack experiments, several boundary conditions of the Kad network needed to be explored. This was done with several preliminary experiments.

In Chapter 5.1.1, the bootstrap time of a node will be examined. This allows experiments to be designed so the victim node is fully integrated in the network.

Then, the performance of the Kad routing algorithm will be explored. While Stutzbach calculates average hop length of the Kad network in [StRe06], the network characteristics may have changed in the meantime. Thus, in Chapter 5.1.2, the influence of tester-victim distance on the routing path length will be examined.

Finally, in Chapter 5.1.3, the inclusion of Sybil nodes in the routing path will be checked, since inclusion of the Sybils is important for the success of a Sybil attack,

5.1.1 Evaluation of Bootstrap Time

Firstly, several experiments were executed to evaluate the length of time after which a node is integrated into the network closely enough to commence regular operation. Until the joining node has been included in some buckets on other nodes, it cannot be reached over the network. If experiments are started before the routing tables of the victim have been adequately filled, they do not accurately reflect the behavior of a normal network. Thus, a certain delay is required before starting attack experiments.

Normally, nodes can participate in the network relatively quickly, since the buckets from the last run are saved in the `nodes.dat` file. However, since the Kad ID

changes with every one of our experiments (which is required in order to keep the experiments independent from each other), the routing table has to be refreshed , and this can take some time. This routing table refresh is accomplished through the use of `NODE` and `NODECOMPLETE` searches. `NODE` searches are regularly started for random keys in order to fill buckets that are not filled enough yet, and `NODECOMPLETE` queries are started periodically for the own key. Since the XOR metric is symmetric, these queries also lead to insertion into the other nodes buckets. For a closer description of these search types, see Table 2.3 and Chapter 2.2.1.

This ad hoc bootstrapping does not occur on regular joining of the Kad network, since the buckets are already filled from the `nodes.dat` file (see chapter 2.2.1). Executing a regular bootstrap phase from a bootstrap nodes file (retrieved from www.nodes-dat.com, for example) would however lead to the addition of the same nodes in several nodes' routing tables in the experiment setup. Since this would falsify experiment results, no bootstrap nodes file is used, and the bootstrapping phase above is used.

To evaluate the required delay, only the tester and the victim were used. First, the victim client is started and connected to the network. Then, after a certain time, the tester client is started and connects to the network. It then starts searching for the victim in 15 second intervals. By analysing the log file, the time after which the victim is first found is calculated. The results are plotted in Figure 5.1.

Date	*Experiment Nr.*	*Description*
2011/03/23	27	1 Minute bootstrap time
2011/03/23	28	5 Minutes bootstrap time
2011/03/23	29	10 Minutes bootstrap time
2011/03/23	30	15 Minutes bootstrap time
2011/03/23	31	20 Minutes bootstrap time
2011/03/23	32	Repetition Experiment 27
2011/03/23	33	Repetition Experiment 28
2011/03/23	34	Repetition Experiment 29
2011/03/23	35	Repetition Experiment 30
0211/03/23	36	Repetition Experiment 31

Table 5.1: List of bootstrap time experiments

Every bootstrap experiment was repeated once, as seen in the experiment list in table 5.1. As can be seen by the results in Figure 5.1, the time until the victim is first found decreases dramatically after 5 minutes have passed between tester and victim addition, and does not decrease any further.

It can safely be assumed that after 10 Minutes, the victim's bootstrap phase is finished and it is fully integrated into the Kad network. Also note that the bootstrap status of the tester node has no influence on the time until the victim is found (otherwise, the overall results would be much higher).

The time of 10 minutes between addition of the victim and the tester nodes will be used in most other experiments. This allows the victim node to settle into the network in a stable state before commencing the attack.

5.1.2 Influence of Tester-Victim Distance

In order to judge the effect of an eclipse attack in delaying a search, the regular network performance without an attack has to be measured first. Several experi-

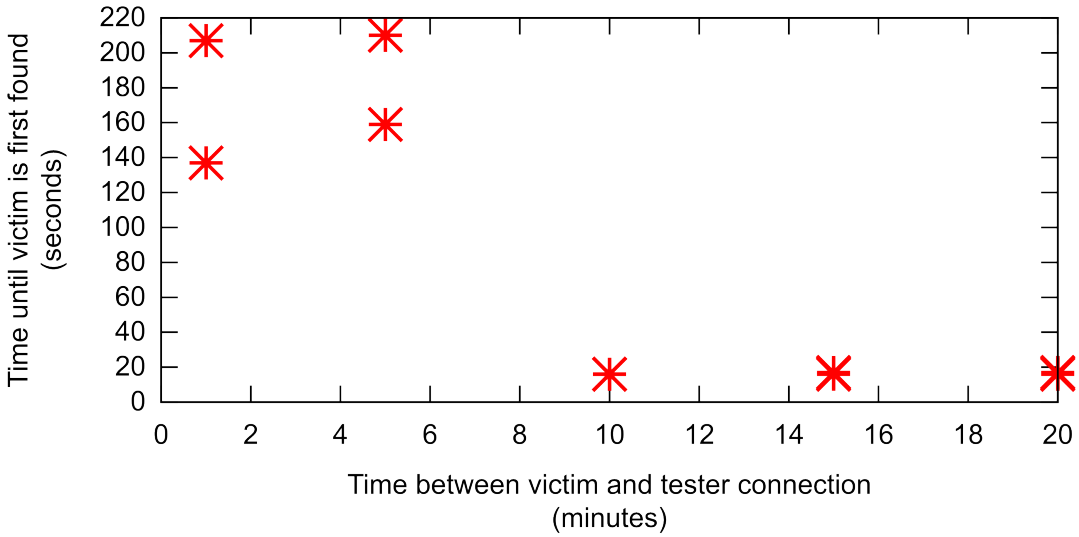


Figure 5.1: Evaluation of bootstrap time: Time until victim is first found vs. delay between tester and victim additions

ments were executed to judge the average routing path length and time depending on the distance between searching node and target.

This allows comparing the results from the attack experiments with results in an undisturbed network. Then, the degree to which the Sybil attack influences regular network performance can be calculated.

The length of the total routing path has great influence on the execution of an eclipse attack: the more hops a search takes before hitting our Sybils, the easier it is to force a timeout to exit the RPC call (because some time has already passed). However, when more hops are taken, more non-malicious nodes are contacted. When one of these nodes returns the victim, our attack has failed. Since a single routing step in Kad takes only a very short amount of time and thus has little influence when trying to reach a timeout, a short routing path is more beneficial for our attack. However, note that the Sybil nodes still need to be included in the routing path for the attack to succeed.

To evaluate the routing path length (without Sybils), a tester and a victim node were introduced into the network within ten minutes of each other (as suggested by Chapter 5.1.1). The tester has a varying distance from the victim, varying between 126 bits and 0 bits of shared prefix.

Then, the tester node starts searching for the victim in regular intervals of 60 seconds. The experiment is run for 30 minutes.

The number of hops taken between the tester and the victim is analyzed. Also, the average time a search for the victim takes is analyzed.

The experiment results are show in figure 5.2. The figure shows that average routing length does increase slightly with the distance between tester and victim, but not significantly for our experiments. Note that on all distances except 120 bit, the average path length even stays below the average step length calculated by [StRe06]. Also note that the length of the routing path only increases markedly when there is no shared prefix.

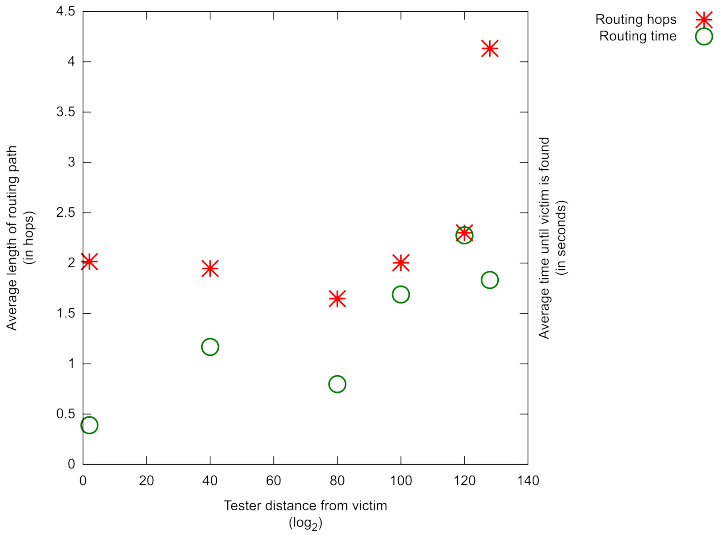


Figure 5.2: Influence of the distance between tester and victim on search time and routing hops.

Date	*Experiment Nr.*	*Description*
2011/03/29	40	Distance 2 bits
2011/03/29	41	Distance 40 bits
2011/03/29	42	Distance 80 bits
2011/03/29	43	Distance 100 bits
2011/03/29	44	Distance 120 bits
2011/03/29	45	Distance 128 bits
2011/03/31	50	Repetition Experiment 45
2011/04/11	59	Repetition Experiment 40
2011/04/11	60	Repetition Experiment 41
2011/04/11	61	Repetition Experiment 42
2011/04/11	62	Repetition Experiment 43
2011/04/11	63	Repetition Experiment 44
2011/04/11	64	Repetition Experiment 45

Table 5.2: Routing Length Experiments

Also note the parallel nature of the Kad routing algorithm. At a common prefix of zero bits, slightly more than 4 routing steps are taken on average. However, the time needed for the search does not increase. This is because several routing steps are executed in parallel.

5.1.3 Inclusion of Sybils in Route

The relatively short length of the routing path between tester and victim shown in the previous experiment raises the question whether Sybils will be found before the victim. If the victim is always returned before the Sybil nodes, no attack is possible.

To check for inclusion of the Sybils in the routing path, another preliminary experiment was executed. It consists of an “active” tester, i.e. one that actively searches for the victim, several inactive testers and the victim. As in the experiment before, the testers are added 10 minutes after the victim is introduced into the network.

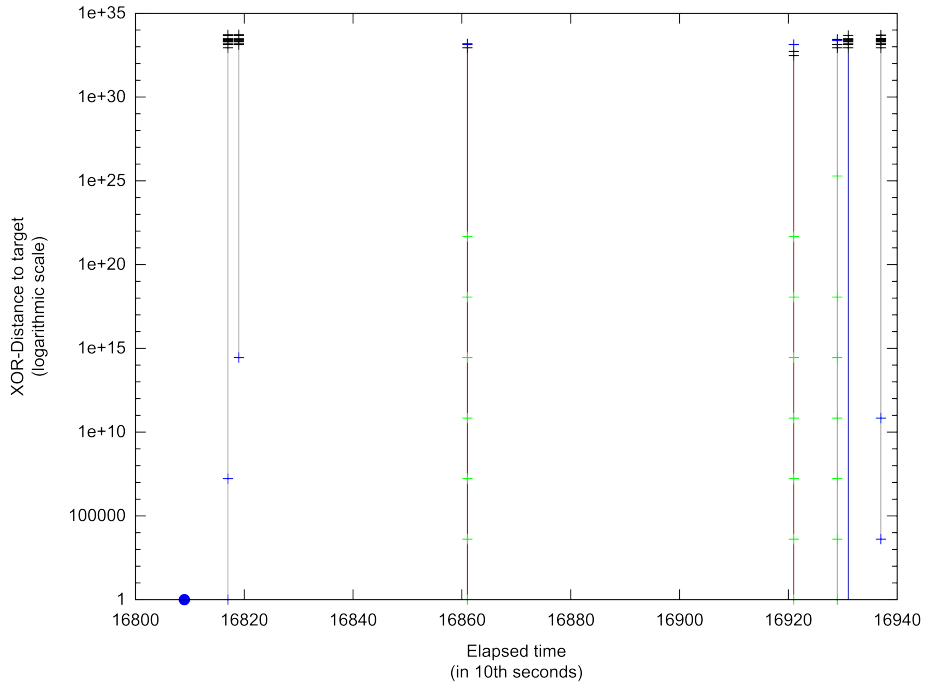


Figure 5.3: Inclusion of Sybils in the routing path

The active tester ID has a common prefix of 120 bits with the victim, while the inactive tester IDs are calculated with algorithm 3. This algorithm should create only IDs that are closer to the target than any other legitimate node. This structure is identical to the structure that will later be used in the attack experiments.

Then, the results are plotted, and checked for the inclusion of inactive testers. If the inactive testers are regularly included in the route, chances are good that malicious Sybils will also be included, and thus an attack is possible.

The visualization of two searches towards the end of the experiment is shown in Figure 5.3. Since this is the visualization that will be used in later experiments, the plot format is described here in short.

The diagram shows the packets received by the tester in a search for the victim. Each vertical line connecting several crosses is a `KADEMLIA(2)_RES` packet, and the crosses are the nodes contained in the packet. The nodes are ordered by descending XOR distance to the target. The color of the node shows if the node is under malicious control or was the source of the current packet:

Black crosses Uncontrolled regular nodes

Green crosses Nodes under control of the attacker

Blue crosses Source of the displayed packet

The color of the line indicates whether the victim was contained in this packet:

Grey line Victim was not contained in this answer packet

Red line Victim was contained in this answer packet

Blue line The victim was the source of this answer packet

Finally, the blue circles indicate the start of a search. This visualization allows following the progress of a search, and allows a quick visual check whether and how the Sybils were contacted and the attack worked. It is also very useful to find out why the attack has failed.

In this example, it can clearly be seen that all 8 Sybils eventually appear in the routing answer (the two red lines have 8 green crosses in total). This is advantageous for executing an attack, since it means the Sybils will definitely be found and included in the routing answer. However, all Sybils were returned in the same packet as the victim. Since the Sybils were not configured for malicious behavior, this is hardly surprising. However, it shows that the Sybils and the victim are both inserted into the same k-buckets in regular nodes, a fact that complicates a successful attack.

5.2 Attack Experiments

With these preliminary experiments, the useful parameters for executing an eclipse attack have been found: The preliminary experiment in Chapter 5.1.1 suggests a bootstrap time of 10 minutes between adding the tester and the malicious nodes to the network.

In Chapter 5.1.2, it is shown that the influence of distance is not a big influence on the routing performance of the network. While the number of routing hops increases markedly at a prefix length of 0, hop count stays between 1.5 and 3 hops for all other distances.

Thus, a shared prefix of 8 bits between tester and the victim was chosen, since it only slightly increases the total routing length. Also, this prefix is markedly shorter than the prefix of the closest regular node, which shares a common prefix of length 20 with the victim (see Chapter 2.2 for how this was calculated).

Thus, the next experiments will be executed with a tester-victim distance of 120 bits, and 10 minutes between adding the victim and the tester.

First, a basic attack will be executed with a low number of Sybils to gauge the possibility of an attack. This experiment is described in chapter 5.2.1.

To judge if the attack can be successful for a longer time, a longer attack will be run. This experiment will be described in Chapter 5.2.2.

Finally, since the low number of Sybils from Chapter 5.2.1 will not be sufficient for a successful attack, the number of Sybils will be increased. This will hopefully allow execution of a successful attack. This experiment will be described in chapter 5.2.4.

5.2.1 Basic Attack

The first attack experiment will gauge the impact an eclipse attack can have in the network. The hosts listed in Table 4.1 will be used. With 8 Sybil nodes, the attack will probably not be executed successfully. However, the impact of the Sybil chain can be estimated. The visualization described in chapter 5.1.3 can be used to find out whether the Sybil chain worked as expected, and a visualization of search duration can be used to estimate impact of the chain.

In this experiment, the basic attack will be tested as described in Chapter 4.1: One victim node, one tester node and 8 Sybil nodes. The cluster manager is started, and the tester is configured to have a distance of 120 bits from the victim ID (i.e. 8 bits shared prefix length). This distance was chosen arbitrarily. When executing the attack in a real network, the exact position of the searching node cannot be determined beforehand, and the searching node ID is chosen randomly. In this experiment, positioning of the tester node allows easier analysis and repeatability of the experiment.

The malicious nodes are positioned as described in Chapter 4.1. With 8 Sybils, each Sybil will have a common prefix of $20 + i * 12$ bits with the victim, where i is the number of the Sybil. This leads to a exponential distribution over the ID space. Every Sybil is closer to the target than the closest non-malicious node, which has only about 20 bits of prefix length with the victim (see Chapter 2.1.1). This means all Sybils are closer to the target than legitimate nodes, which is a prerequisite for an eclipse attack.

Then, after 10 minutes, the tester and the malicious nodes are added into the network. The tester starts searching for the victim in regular intervals of 60 seconds, and the malicious nodes execute the Sybil chain algorithm from Chapter 2.1.3 with a Sybil delay of two seconds.

After 60 minutes, all Kad clients are closed and the log files are retrieved for analysis. Also, the k-buckets of the tester node were logged to be able to follow routing decisions.

A number of experiments were executed as seen in Table 5.3 with differing results.

Date	*Experiment Nr.*	*Results*
2011/02/23	18	3 second Sybil delay. Used NODECOMPLETE search type. No success.
2011/02/23	19	Repetition of experiment 18, no success.
2011/03/10	23	Now using NODESPECIAL search. No success.
2011/03/17	26	Regular experiment. No success.
2011/03/24	37	Regular experiment. Success.
2011/03/25	38	Repetition of experiment 37. No success
2011/03/30	48	Repetition of experiment 37. No success
2011/03/31	49	Tester distance 126 bits. No success.

Table 5.3: Executed basic attack experiments

In most of the experiments, the search process was not influenced (such as in Experiments 26, 38 and 49). The average search length did not change, and was always around one second.

When the malicious nodes were added to the buckets of the searching node, as in experiment 37, they were able to influence the search duration, and an average search duration of about 15 seconds was reached. Compared to the average search lengths in Chapter 5.1.2, this is a notable delay. To illustrate this point further, two sample experiments will be examined in detail.

The difference between these two experiment runs is that in Figure 5.5, no malicious nodes were into the buckets of the searching node, while in Figure 5.4, several malicious nodes were added into the buckets. This makes a huge difference in

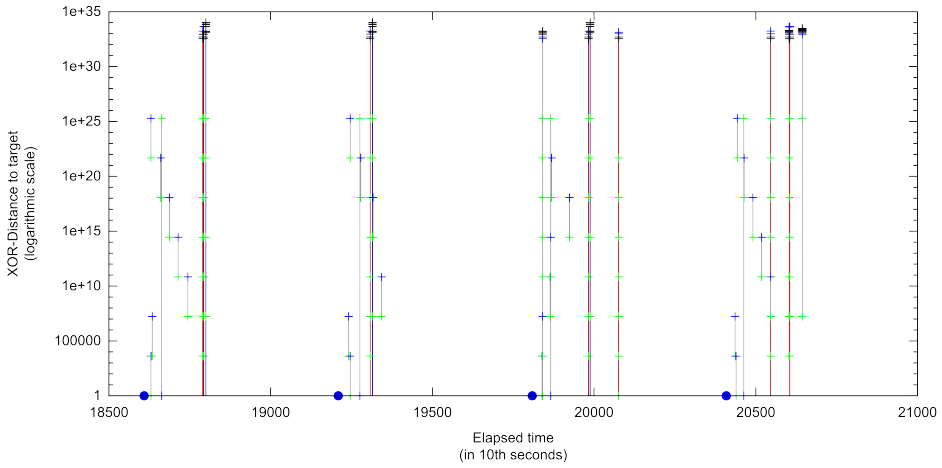


Figure 5.4: Experiment 37: Basic attack, Sybil added into buckets

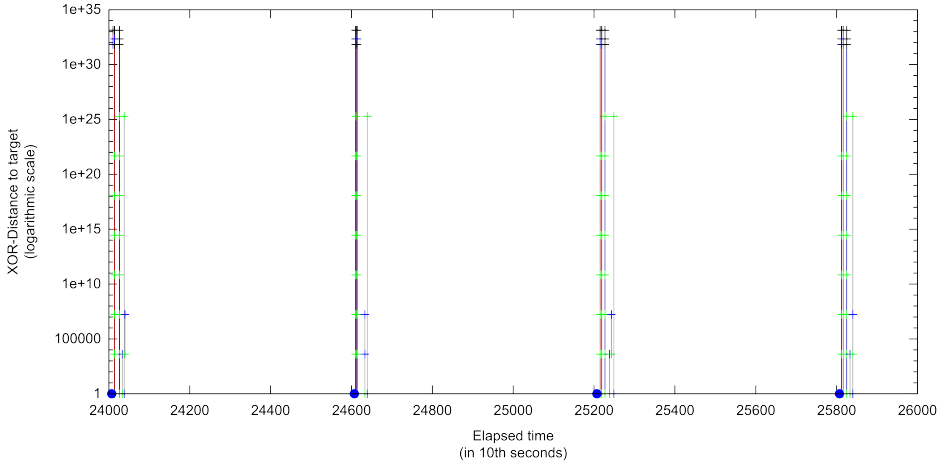


Figure 5.5: Experiment 38: Basic attack, Sybil not added into buckets

regard to the attack success: in the first experiment, the Sybils are only ever queried after the victim has already been found. This means there is no chance of starting the Sybil chain algorithm, and the victim will always be found, no matter what the number of Sybils is.

In the second experiment, a successful attack would have been possible with more Sybils: all of the malicious nodes are queried, and the work of the Sybil chain can be seen. One Sybil (in blue) returns the next closer Sybil (in green). Then, the searching node queries the next Sybil, which again returns the next closer node and so on. Eventually, the chain ends. Since all Sybils have been queried, a regular node is queried for the victim. This leads to the victim being found. With more available Sybils in the second experiment, the eclipse attack would most probably have succeeded.

In the diagrams, the parallel search behavior of Kad can be seen: at the beginning of the search process, three nodes are queried in rapid succession. This also means that three Sybils need to be added into the buckets for maximum attack success. This was the case in Experiment 38, as proven by looking at the bucket contents of the tester node. If less than three Sybils are included in the buckets, non-malicious nodes are included in the search.

The victim and the malicious nodes are all saved in the same buckets on regular nodes (since they all have the same prefix). This means that nodes who have both malicious nodes and the victim node in their routing tables return both when queried for the victim. The fact that the malicious nodes and the Sybils are inserted into the same buckets makes it very unlikely that a node will return malicious nodes, but not the victim node. This, of course, causes the attack to fail.

The effects of the Sybil chain can also be shown in another visualization: In diagrams 5.7 and 5.6, the x axis measures the time from the beginning of the experiment to the start of the search, while the y axis represents how long the search took. The green line displays the running average of the search time over the last 20 searches.

It is immediately recognizable in Figures 5.5 and 5.4 that all searches in experiment 38 took less than 3 seconds, while in experiment 37, the search time fluctuates between 20 and 10 seconds. While this is not long enough to cause a timeout, it shows the relative effectivity of the attack. With more Sybils, an attack would have been successful.

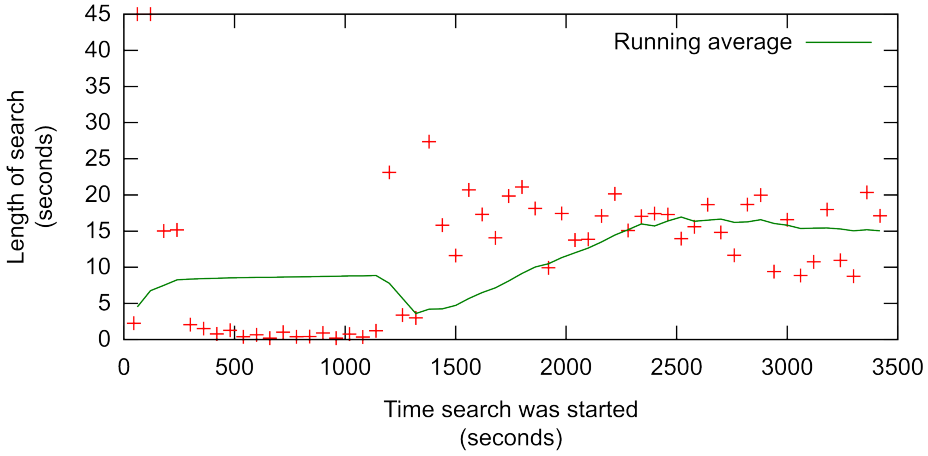


Figure 5.6: Experiment 37: Search duration

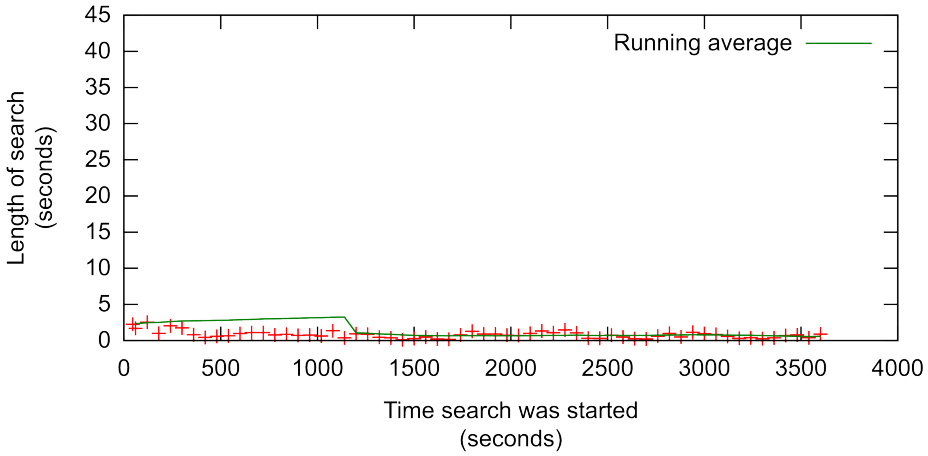


Figure 5.7: Experiment 38: Search duration

The core requirement for executing an eclipse attack is thus insertion of malicious nodes into the searching nodes' routing tables. However, this is difficult. Because

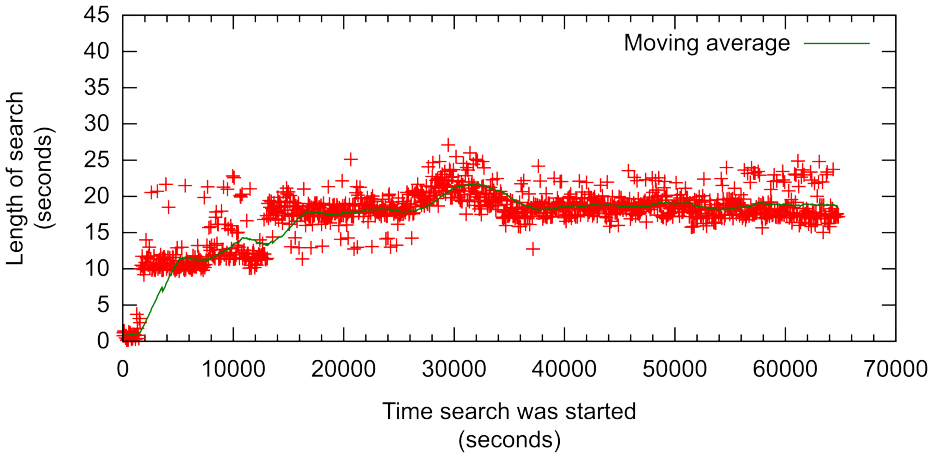


Figure 5.8: Experiment 53: Search duration

Kad has relatively static routing tables (due to preference of nodes with long uptime), the malicious nodes will only rarely be added into the testers buckets. Also, the saving of nodes between client executions (see chapter 2.2.1) makes bucket insertion even more difficult.p

As seen in Table 5.3, only one out of eight experiments was successful in adding malicious nodes to the tester buckets and influencing the search duration by a noticeable amount.

However, no exact data was collected on the likelihood of bucket addition, as this would have required a radically different setup and a large amount of experiments.

5.2.2 Increasing Length of Attack

With the above knowledge, it is interesting whether the attack will be successful when conducted for a longer period of time. Churn could lead to the addition of malicious nodes into the testers' buckets, or the activity of the victim could lead to more knowledge of the victim in the network. Also, if the malicious nodes are correctly added to the buckets, the approximate delay per Sybil can be calculated.

To evaluate this, a longer experiment was run. While the last experiments had a duration of 60 minutes, now a longer period of 18 hours will be used. The rest of the experiment setup is unchanged from the experiment setup described in Chapter 5.2.1: The victim is added 10 minutes before all other nodes and the tester has a distance of 120 bits from the victim.

Date	*Experiment Nr.*	*Results*
2011/03/25	39	Success
2011/03/31	53	No success. Only 4 Sybils configured correctly.

Table 5.4: Long attack experiments

The search time visualization of experiment 53 is shown in Figure 5.8. It shows that the Sybil attack is possible in this experiment. After about 30 minutes, the average search time increases to about 12 seconds. However, after a longer run time of about 4 hours, the average delay increases again to about 19 seconds.

Using the average routing times of 12 and 19 seconds, this corresponds to a per-Sybil delay of 1.5 and 2.375 seconds per Sybil. To successfully force a timeout of 45 seconds, about 19 Sybils would be needed towards the end of the experiment.

Also note that the attack does not become less successful over time. The average search duration does not decrease again. This means the traffic caused by the victim by its regular protocol operations (like refreshing buckets and receiving RPCs) does not suffice to make a victim well enough known in the network to defend against a Sybil attack.

However, until the Sybil nodes are known well enough in the network to cause a timeout, the victim is reachable. Addition into buckets due to churn is thus not desirable for an attacker, since the victim has time to publish its content in the network until enough Sybils are added. By joining the network with malicious nodes before the victim connects, this can be avoided.

With these results, an experiment with an increased number of Sybils will be executed. It should be possible to force a timeout on searches if enough Sybils are available.

5.2.3 Inserting Malicious Nodes Before Victim

In the previous experiments, the malicious nodes were always added after the victim. This allowed the victim to connect to the network, and allowed the victim to add itself into the buckets of regular nodes.

Regular nodes returning the victim contact data and not the contact data of malicious nodes is the major cause of a failing attack (see figure 5.5, where regular nodes return the victim contact data). This is caused by the Sybils not being well known enough in the network.

To increase the number of buckets into which the Sybils were inserted, the Sybils were connected to the network a certain time before the victim. This allows the Sybils to be added into the buckets of regular nodes. Then, the experiment was run for one hour with the standard settings also used in Chapter 5.2.1.

Date	*Experiment Nr.*	*Description*
2011/03/12	25	Sybils added 16 hours before victim. Success
2011/04/01	54	Sybils added half an hour before victim. Success

Table 5.5: Malicious nodes added before victim connection

Two experiments were executed, as seen in Table 5.5. Both experiments succeeded in delaying the search for the target by a noticeable amount.

However, in Experiment 54, the Sybil nodes were added to the buckets of the tester node. This is similar to the basic experiments executed before, and not surprising. Similar results were returned by the delay statistics, as seen in figure

However, when the Sybils were added to the network 16 hours before the connection of the victim, a significant delay of 30 seconds was reached with only 8 Sybils, as seen in figure 5.10.

The cause of this large delay is the better “publicity” of the Sybils. Because of their long uptime, the Sybils were added into the buckets of several non-malicious nodes. When searching for the victim, several non-malicious nodes repeatedly

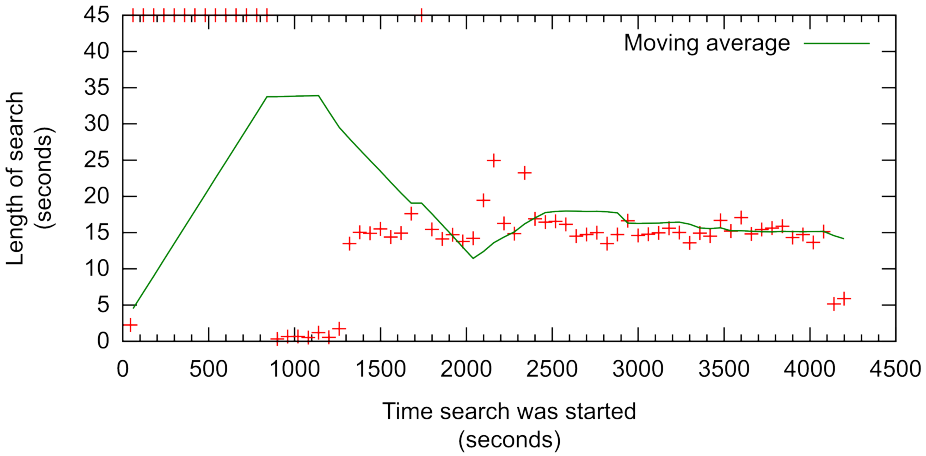


Figure 5.9: Experiment 54: Sybils added to buckets

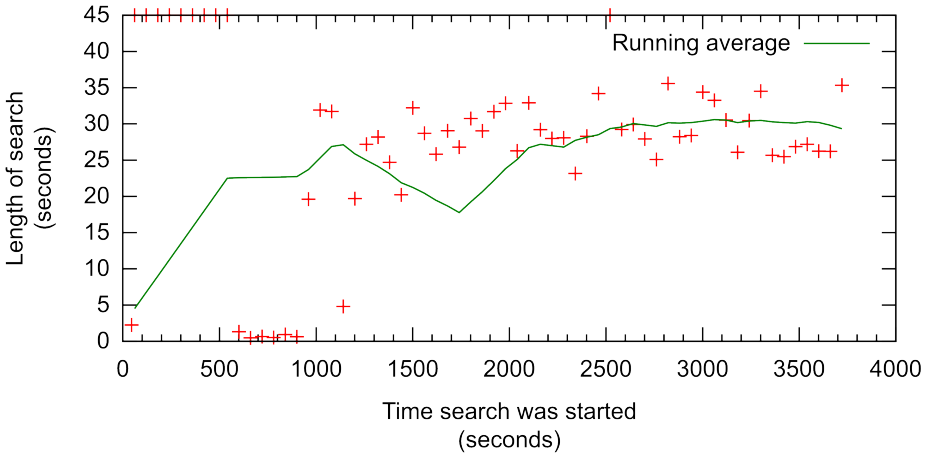


Figure 5.10: Experiment 25: Sybils added 16 hours before victim. A large delay is reached with a low number of Sybils.

returned Sybil nodes but not the victim node in their answers. This can be seen in Figure 5.11.

This is different from the experiments executed in Chapter 5.2.1, where a non-malicious node almost always returned the victim directly. This is very advantageous for the attack, as the Sybil chain is no longer dependent on the fact that no regular nodes can be queried.

While the number of Sybils was too low in this experiment to cause a search timeout, the total search delay reached in this experiment is larger than in the experiments before. It is therefore advantageous for an attacker to add his nodes to the network before the victim is introduced, and thus allow higher knowledge of the Sybils in the network.

5.2.4 Increasing Number of Sybils

The previous experiments show that an eclipse attack in the Kad network is definitely possible under the right conditions. The most important prerequisite is addition of at least three Sybil nodes into the tester buckets or a high publicity of

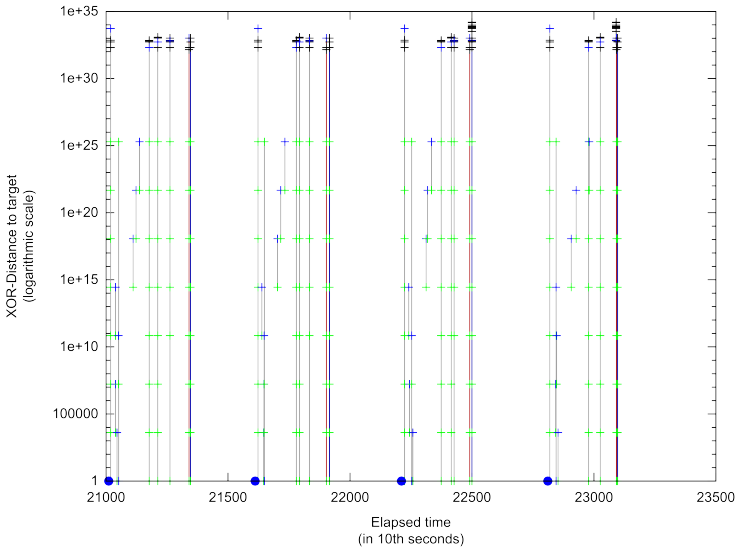


Figure 5.11: Experiment 25: Regular nodes returning malicious nodes

the Sybils, as in the experiments in Chapter 5.2.3. Then, the Sybil chain can be used to delay searches for the victim.

However, the numbers of Sybils in the last experiment was obviously not sufficient to actually reach the timeout of 45 seconds. To evaluate the actual possibility of reaching a timeout, the number of Sybils will be increased.

Date	*Experiment Nr.*	*Description*
2011/04/02	55	Sybil delay of 3 seconds too large. No success
2011/04/04	56	Success.
2011/04/05	57	No success.
2011/04/05	58	No success.

Table 5.6: Executed experiments with increased Sybil number

In these experiments, 44 Sybil nodes where used. These nodes are all part of the PlanetLab network test bed. The choice of PlanetLab nodes ensures that the subnet protection of Kad will not trigger.

With the average delay of 1.5 seconds seen in experiment 5.2.2, this means a timeout will be reached. The experiment has a run time of 80 minutes. other settings (tester-victim distance, time between victim and tester addition, cluster manager configuration) were kept from the last experiment. Again, the length of a search will be used as indicator for the attack success.

Figure 5.12 that Experiment 56 has clearly succeeded. After about 10 minutes, the search reaches a timeout of 45 seconds. In 80 minutes experiment time, 12 searches for the victim were successful, and 68 searches failed due to a timeout. Again, three Sybils were added to the tester buckets.

In Figure 5.13, the working Sybil chain can be seen: Each node, after a timeout, returns the next closer node to the querying node. Due to an error in the experiment configuration, one of the sybils does not correctly answer with the next chain element. However, one of the regular nodes queried returns a chain element, and the attack can continue.

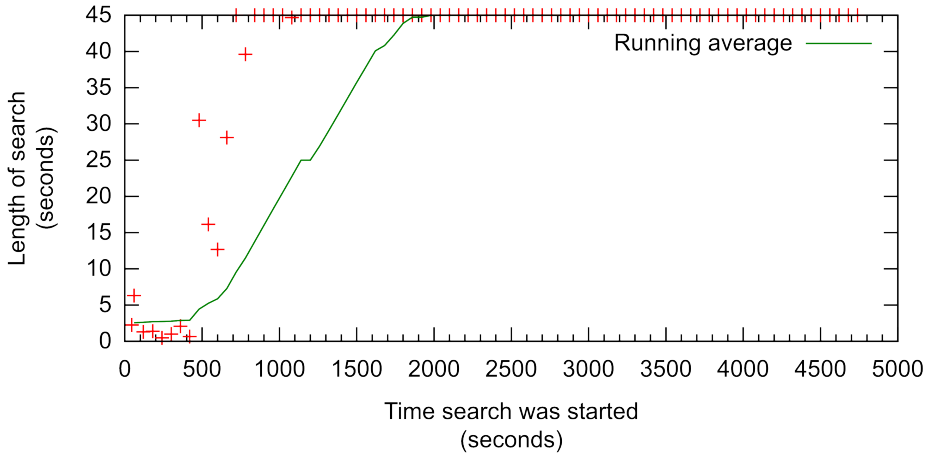


Figure 5.12: Experiment 56: Search duration

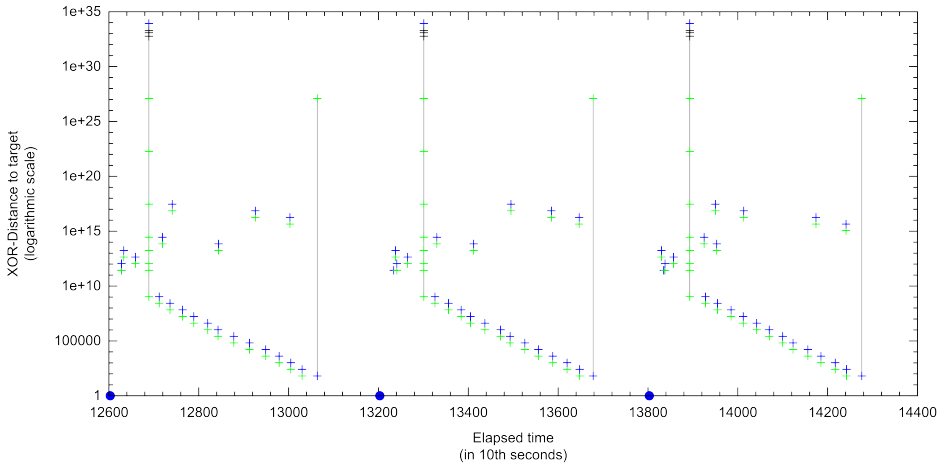


Figure 5.13: Experiment 56: Successful Sybil chain

However, even with a high number of Sybils, the attack can fail, as in Experiment 57. Here, only one Sybil was added to the tester's buckets. Figure 5.14 shows that this is not sufficient to cause a timeout or even influence the search duration significantly.

Experiment 56 shows that a successful attack is definitely possible. However, a large number of participating nodes does not guarantee success, as shown by the failing experiments. The addition of at least three Sybils into the tester's buckets is still a crucial point.

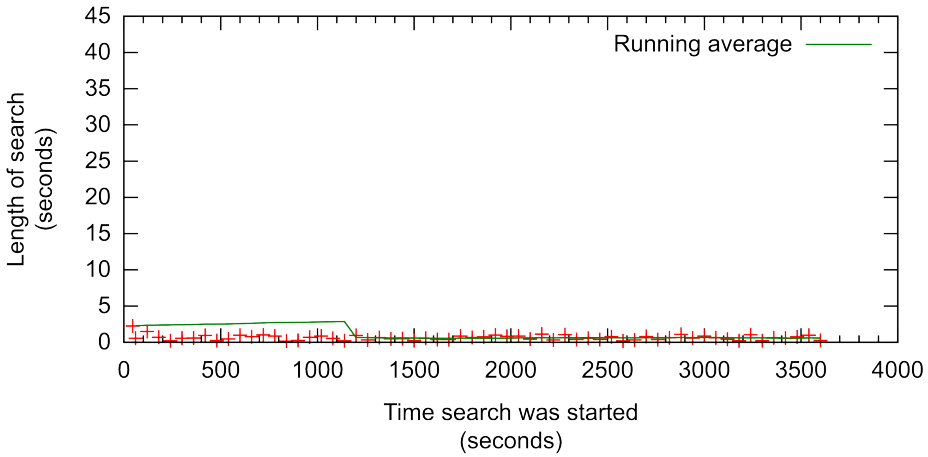


Figure 5.14: Experiment 57: Search duration

6. Evaluation

In the previous chapters, the eclipse attack has been executed with varying parameters. Results from these experiments will be described in chapter 6.2. Then, several improved attacks strategies are described that can improve attack success in Chapter 6.2. Finally, a mitigation strategy for the eclipse attack is described in chapter 6.3.

6.1 Experiment results

During the course of this work, a large amount of experiments were executed. Not all of these experiments could be used. In particular, most early experiments needed to be repeated after the selection of the `NODESPECIAL` search type for the tester. However, all executed experiments, whether failed, misconfigured or executed successfully, helped to understand the inner workings of the Kad implementation in aMule.

In the next chapters, the results of the experiments executed in chapter 5 will be summarized.

6.1.1 Preliminary Experiment Results

Several preliminary experiments were executed to understand the performance parameters of the Kad network as necessary for an eclipse attack. The three preliminary are shown in Chapter 5.1.

The bootstrap time for our experiment setup was evaluated by testing how long the victim needed to run in order to be fully integrated into the network. After 5 minutes of running, the time after which the victim was first found decreased sharply from more than 140 seconds to about 20 seconds. This time did not decrease any more when the victim was allowed to run longer before addition of the tester.

This means the time necessary for full integration into the network is about 10 minutes.

Then, a second row of experiments was run to explore the performance of undisturbed searches in the network. The average number of received replies until the search ended and the time a single search for the victim took were both measured with differing distances between tester and victim.

It was found that regular searches using the `NODESPECIAL` search normally terminate very quickly. A single search for the victim takes less than 4 seconds on average, regardless of distance.

Finally, the inclusion of the sybils on the routing path between tester and victim was tested. Here, a central problem when executing an eclipse attack can be seen: While the sybils are returned by the routing algorithm, the victim is almost always contained in the same packet. This means the attack will fail, even with inclusion of the sybils.

6.1.2 Basic attacks

With these results, an attack could be executed. An eclipse attack with the algorithm described in Chapter 2.2.2 was executed with the settings found by the preliminary experiments.

The executed attack experiments then showed that executing an eclipse attack in the Kad network is definitely possible. When at least three malicious nodes are added into the buckets of the tester node, 8 Sybils are able to delay a search by at least 15 seconds.

With less than three Sybils added to the buckets, the attack does not succeed in markedly delaying the search. Non-malicious nodes that are queried during the course of the search generally return the victim directly. Due to the malicious and victim node sharing a relatively long prefix, they are all added into the same bucket in regular nodes. This means that a search for the victim will generally return the victim (and, possibly, several Sybil nodes). However, at that point, the attack has already failed.

The requirement of at least three Sybils stems from the parallelism of the Kad routing algorithm. Because three nodes are queried of the start of the search, three Sybils need to be included in the k-buckets. Otherwise, regular nodes are queried.

6.1.3 Increasing Attack Length

The search process can be strongly influenced in the Kad network, as shown by the previous experiment. However, it is not known if the attack will still be successful over a longer period of time.

Due to the influence of churn and regular victim activity, the victim might become better known in the network, and the attack might fail. To evaluate this, experiments with longer attack duration were executed.

While the previous attack experiments were run only for one hour, these experiments were run for about 18 hours.

three malicious nodes were again added into the tester nodes' buckets, and searches were delayed markedly. The delay imposed on the search process even increased with increasing experiment duration.

While after the first 10 minutes, a delay of about 12 seconds was reached, this increased to about 19 seconds after 4 hours of experiment run time.

This shows that a longer attack does not decrease in effectiveness, and that the traffic caused by regular protocol operations (receiving publish results, sending bucket refresh packets, checking aliveness of nodes) is not enough to prevent attacks on a victim. The regular activity increases the publicity of the victim in the network, but this is not enough to disrupt the Sybil chain.

6.1.4 Adding Malicious Nodes before Victim

In Chapter 5.2.3, the malicious nodes were added before the victim. This allows the malicious nodes to be inserted into buckets before the victim node. While adding the Sybils half an hour before the target node had no notable effect, addition of the Sybils into the network 16 hours before the victim had a remarkable impact.

With only 8 sybils, a search delay of 30 seconds was reached. After the 8 Sybils of the Sybil chain have all been contacted, regular nodes are queried. They again return the Sybil nodes, and not the target node. This leads to a long delay in the search process.

The long residence of the Sybil nodes in the network leads to the insertion into many regular nodes' buckets. Also, since the victim has only recently joined the network, it is not included in many buckets. This means that the Sybil nodes are better known than the victim. Regular nodes then return only the Sybil nodes when queried, because they have not yet heard of the victim.

This is of great advantage to an attacker, since it removes the need for the malicious nodes to be added into the tester nodes' buckets. Since the attacker knows the ID of the victim beforehand, adding malicious nodes into the network a longer time before the victim joins the network is not a problem. The experiments also prove the importance of Sybil publicity: With a large enough publicity, the Sybils od not even need to be added to the tester nodes' buckets to execute an attack.

Also note that 10 Sybils could suffice to totally block the victim from addition in regular nodes' buckets. Since the malicious nodes and the victim are all inserted into the same buckets on regular nodes (since their shared prefix length is far longer than that of regular nodes), addition of 10 malicious nodes into the buckets would stop the victim from being included in the buckets. Then, an eclipse attack is much easier to execute.

6.1.5 Increasing Number of Sybils

While searches for the victim were strongly influenced in the previous chapters, none of the searches actually reached a timeout because of our attack.

This was due to the low number of Sybils. While a maximum delay of 19 seconds could be reached with 8 Sybils, the timeout for a `NODESPECIAL` search is 45 seconds. The delay factor of 8 Sybils is thus not enough to force a search timeout.

In order to reach a timeout, the number of Sybils used in the attack was increased to 44. PlanetLab nodes were used to avoid the subnet restrictions placed on the routing table

With this increased number of nodes, a timeout was reached. After 15 minutes, searches continually reached a timeout. This means that communication between victim and tester is now disrupted, and the victim can no longer be found by the tester.

6.2 Improved Attack Strategies

The previous chapters have shown that attack success relies greatly on the addition of malicious nodes to the tester's buckets. In the experiments executed in this work, this addition has taken place several times. However, the low number of

repeated experiments does not allow assessment of the exact probability of Sybil node addition into the buckets.

However, it is immediately apparent that the probability of Sybil node addition into the tester buckets is strongly influenced by the degree to which the Sybil nodes are added into the buckets of regular nodes (we will call this "publicity" from now on). When Sybils nodes are added into the buckets of regular nodes, this increases the chances the tester will receive the Sybil node contact data at some point in the future. This may lead to the addition of the Sybil nodes into the tester buckets.

Increasing the publicity of the Sybil nodes is most easily done by having the Sybil nodes query regular nodes. By starting searches for random keys in regular intervals, more regular nodes come into contact with the Sybil nodes, and insert them into their routing tables. This increases the chance that the Sybil nodes will eventually be added to the tester's buckets.

By issuing `HELLO` requests (for example) in regular intervals, addition of the Sybils into remote buckets can be facilitated. However, care must be taken to not trigger the Kad flood protection, since this will drop packets or even ban the source IP if the number of packets per minute is too high. With `hello` requests, a maximum of 3 packets can be received per minute. When more than 15 packets are received, the IP is banned permanently. However, not only the `HELLO` request can be used. The Kad implementation needs to be analyzed in detail to find RPC calls that allow an insertion into the buckets with high probability.

When the malicious nodes have been added into the tester buckets, further optimizations can be used to reduce the number of required Sybils to reach a timeout. In Chapter 5.2.2, a delay of 19 seconds was eventually reached with 8 Sybils. This corresponds to a per-Sybil delay of $19/8 = 2.375$ seconds. To reach a timeout of 45 seconds, 19 Sybils are then required.

However, the attack configuration used in this attack was probably not optimal. Several optimizations can be used to increase the delay that is imposed on the search for each step in the Sybil chain.

An increased delay between chain members can decrease the number of required Sybils. In the experiments, a delay of 2 seconds between receiving the routing request and answering with the next chain member was used. By increasing this delay, the number of Sybils can be reduced. However, a routing step can be delayed for at most 3 seconds, since after that time, the jumpstart mechanism leads to additional regular nodes being queried.

Also, the addition of fake nodes into answers from the Sybils could increase the delay between members of the chain. This is described by Kohnen in [KoLR09], but not done with the current patch set. By including nodes that do not exist, but are still closer to the victim than any known node, additional time is lost querying these non-existing nodes. While this timeout will not be large due to Kad's usage of the UDP protocol and the inherent parallelism of the routing algorithm, it still increases the delay between querying one malicious node and the next one. Also, a large number of contacts can potentially be included in the RPC (up to 9 fake nodes, and one Sybil node). This can well lead to a large increase in the per-Sybil delay, and thus a decreased number of required Sybils.

6.3 Mitigation Strategies

To prevent execution of eclipse attacks, the subnet protection described in Chapter 2.2.2 is not sufficient. While it increases the required address amount for executing a successful eclipse attack, acquiring a sufficient amount of IP addresses (between 20 and 50, depending on optimizations) from different subnets is not an impossible task.

While it may be a challenge for an "independent" attacker, most larger organisations have enough addresses to support such an attack. Even single attackers can amass vast amounts of addresses by the use of botnets. Thus, the Sybil protection in Kad can be circumvented easily.

To prevent Sybil attacks, either explicit or implicit certification of the node IDs needs to be implemented. While explicit certification would require massive changes in the Kad code (either for the introduction of a central certification authority, or a decentralized web of trust), implicit certification could be implemented easily. One possibility would be the construction of node IDs.

While currently, Kad node IDs are chosen randomly, they could equally well be constructed from certifiable parameters such as the IP address and port, or DNS names. This way, it would be very hard for an attacker to position nodes correctly in the network. However, it would also complicate the operation of the Kad network.

The introduction of constructed node IDs would disallow IP mobility (i.e. disallow participation in the network with the same ID and different IPs), and complicate operation over systems such as firewalls and NATs.

However, the eclipse attack can be detected easily. Recall that regardless of distance, the node search finishes within a maximum of 5 seconds. In chapter 5.1.2, searches for the target node generally terminated after less than 5 seconds regardless of distance between target node and searching node. Thus, when the node search for a certain node takes very long, it is probable that either the node is being eclipsed, or it is currently not connected to the network.

Also recall that the eclipse chain is relatively "fragile": Whenever regular nodes are queried, the chain is quickly broken and the victim is found. The eclipse attack relies on the querying node not having contact with any non-malicious nodes during the search.

Thus, it should suffice to query a set of random nodes after the search has almost ended and the victim has not been found yet. Since the victim is regularly found in just one or two step when the eclipse attack fails (compare Figure 5.5), this should suffice to find the victim if it is available in the network.

Note that this strategy is only useful on explicit node ID searches, something the Kad network does not usually do. However, the mitigation technique works against the attack as it was executed in Chapter 5.2.

Another technique for discovering Sybil attacks is auditing the node IDs of the nodes seen in the search process so far: When a significant number of nodes in the returned results are uncommonly close to the search target, it is very probable that an eclipse attack is being executed. The nodes whose IDs are so close to the target are probably trying to execute an eclipse attack on the target.

For regular protocol operation, a suitable minimum distance would have to be chosen that minimizes false positives. 30 bits of common prefix with the victim is

suitable, given that the probability of a randomly chosen node ID having 30 bits of common prefix with the target is $0.5^{30} \approx 9.313 * 10^{-10}$.

When a significant number of (say, 8) nodes with a common prefix of 30 bits or more with the search target appears, these nodes are trying to execute an eclipse attack, and need to be blocked. Since the number of Sybils required for a successful eclipse attack is higher than 8, an eclipse attack will always be detected before it is successful.

One useful factor is that this restriction on node IDs would also stop eclipse attacks on content: When publishing, care must be taken to not publish on nodes that are too close together. That way, an attacker has no way to position his Sybils so they are found in the publish process.

While both these techniques would protect against eclipse attacks as executed in this work, the effort spent to implement these strategies would probably be better invested into switching Kad to construct node IDs from IPs. This way, the insertion and positioning of Sybil nodes would be severely hindered, and eclipse attacks would become very hard in the Kad network.

7. Conclusions

The goal of this work was the evaluation of Sybil attacks on nodes in the Kad P2P network.

To this end, the Kademlia and Kad DHT systems were first examined. The aMule Kad implementation and has several important differences to the Kademlia implementation. However, most of these differences were of no consequences to the execution of an eclipse attack.

After the intricacies of the Kad implementation were understood, modification of the Kad client for our needs was started. Of particular importance was a logging infrastructure for easy examination of the experiment results. Also, the attack behavior needed for execution of the eclipse attack was implemented in the aMule client. This work resulted in a "native" Kad client implementation that is suitable for evaluating eclipse attacks in the Kad network. The logging implementation allows flexible analysis of the protocol operations during an attack.

Several preliminary experiments were executed to assess the boundary conditions of the network. In several experiments, the required bootstrap time in the Kad network, the influence of distance on the process of a search and the inclusion of the Sybil path was evaluated. The experiments have shown that a bootstrap time of 10 minutes is sufficient for regular protocol operations. Also, the influence of the distance between searching node and target is negligible. Neither hop distance nor average routing time increased significantly with increasing distance. Finally, the Kad routing algorithm leads to the inclusion of Sybils when they were positioned close enough to the victim.

With these preliminary experiments, regular attack experiments were executed. By executing eclipse attacks under varying conditions and with varying parameters and analyzing the logged data, several results were found.

The basic attack experiments demonstrated that when the Sybil nodes are added after the victim, it is imperative for the malicious nodes to be inserted into the tester nodes' buckets. Without addition, regular nodes that were queried almost always answered with the victim, and a successful attack was not possible.

To test if it was possible to execute the attack for a longer amount of time, several longer experiments were executed. In one experiment, the Sybils were added to

the tester nodes' buckets. Attack success did not decrease over time, as could be expected. To the contrary, the delay reached in searches increased to about 19 seconds. This shows that the activity of the victim does not have a large influence during attacks. Even though the victim had regular protocol activity, this did not influence the attack success.

In the previous experiments, all malicious nodes were added to the network after the victim. However, an attacker could as well add the malicious nodes to the network before the victim connects. This allows Sybil nodes to become better known in the network, and potentially execute a more successful attack. The results showed that adding Sybil nodes before the victim joins the network massively benefits the attacker. The Sybils become well known in the network because of their long uptime (16 hours were used in the experiment) and this causes regular nodes to return the Sybil nodes.

Finally, since all previous experiments failed to reach a timeout because of the low number of malicious nodes used, the number of Sybils was increased from 8 to 44. When the malicious nodes were added to the tester nodes' buckets, a timeout was caused repeatedly and the victim was eclipsed.

So while eclipse attacks are feasible in the Kad network, they depend on two important conditions: The addition of at least three sybil nodes into the querying nodes' buckets, and a sufficient number of sybils (around 19).

However, due to limited time, this work was only able to explore these conditions qualitatively, and not quantitatively. The likelihood of the sybil nodes being inserted into the tester buckets is not known, as well as the influence of sybil publicity on the attack. Also, the exact number of required sybils could not be measured.

For exploring these parameters (likelihood of addition into the testers' buckets, exact needed sybil number and evaluation of sybil publicity), additional research and different experiments are needed.

Bibliography

[aMul09] aMule Dev Team. aMule 2.6 source code. Published online, 09 2009. http://www.amule.org/files/download.php?file=189.

[BaHK07] I. Baumgart, B. Heep and S. Krause. OverSim: A flexible overlay network simulation framework. In *IEEE Global Internet Symposium, 2007*. IEEE, 2007, S. 79–84.

[Boie09] Rainer Boie. *Empirische Untersuchung von Angriffen auf strukturierte P2P-Netze*. Dissertation, Universität Tübingen, August 2009.

[Douc02] J. R. Douceur. The Sybil attack. In *Peer-to-Peer Systems: 1st International Workshop, Cambridge, MA, USA (IPTPS 2002). Revised Papers*, Band 2429/2002 der *Lecture Notes in Computer Science*. Springer, 2002, S. 251–260.

[Ed2k11] eDonkey network statistics. Published online, March 2011. http://ocbmaurice.dyndns.org/pl/ed2k_stats.pl.

[KoLR09] Michael Kohnen, Mike Leske and Erwin Rathgeb. Conducting and Optimizing Eclipse Attacks in the Kad Peer-to-Peer Network. In *Proc. 8th International IFIP-TC 6 Networking Conference (Networking 2009), Aachen, Germany, May 11-15, 2009*, Band 5550/2009 der *LNCS*. Springer, May 2009, S. 104–116.

[MaMa02] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer information system based on the XOR metric. In *First International Workshop on Peer-to-Peer Systems (IPTPS 2002), Revised Papers*, Band 2429/2002 der *Lecture Notes in Computer Science*. Springer, 2002, S. 53–65.

[SMKK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001, S. 149–160.

[SNDW06] A. Singh, T.-W. Ngan, P. Druschel and D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proc. 25th IEEE International Conference on Computer Communications (INFOCOM 2006), Barcelona, Spain*, April 2006, S. 1–12.

[StENB07a] M. Steiner, T. En-Najjary and E.W. Biersack. A global view of kad. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, S. 117–122.

[StENB07b] M. Steiner, T. En Najjary and E.W. Biersack. Analyzing peer behavior in KAD. *Institut Eurecom*, 2007.

[StENB07c] M. Steiner, T. En-Najjary and E.W. Biersack. Exploiting KAD: possible uses and misuses. *SIGCOMM Comput. Commun. Rev.* 37(5), 2007, S. 65–70.

[StRe06] D. Stutzbach and R. Rejaie. Improving lookup performance over a widely-deployed DHT. In *Proc. Infocom*, Band 6. Citeseer, 2006.