HAMACK: a Honeynet Architecture against MAlicious Contents in KAD

Thibault Cholez, Isabelle Chrisment, Olivier Festor

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

HAMACK: a Honeynet Architecture against MAlicious Contents in KAD

Thibault Cholez — Isabelle Chrisment — Olivier Festor

N° 6994

April 2009

ISSN 0249-6399 ISRN INRIA/RR--6994--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
NANCY - GRAND EST

HAMACK: a Honeynet Architecture against MAlicious Contents in KAD

Thibault Cholez* , Isabelle Chrisment* , Olivier Festor*

Thème : MAnagement of DYnamic NEtworks and Services
Équipes-Projets MADYNES

Rapport de recherche n° 6994 — April 2009 — 22 pages

Abstract:

In this paper, we propose a new P2P Honeynet architecture called HAMACK that bypasses the Sybil attack protection mechanisms introduced recently in KAD. HAMACK is composed of distributed Honeypeers in charge of monitoring and acting on specific malicious contents in KAD by controlling the indexation of keywords and files. Our architecture allows to: (1) transparently monitor all the requests sent to the targeted contents in the network, (2) eclipse malicious entries of the DHT, and (3) attract the download requests of peers searching for malicious contents towards the Honeypeers by poisoning the DHT references with fake files and sources. Early results on the KAD network demonstrate the applicability and the efficiency of our approach.

Key-words: P2P networks, DHT, Sybil attack, honeypot, KAD, monitoring

* {thibault.cholez, isabelle.chrisment, olivier.festor}@loria.fr

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

HAMACK: une Architecture de Honeynet permettant de lutter contre les Contenus MAlveillants diffusés dans KAD

Résumé : Nous proposons dans ce rapport une nouvelle architecture de Honeynet appelée HAMACK capable de dépasser les protection récemment introduites dans KAD pour lutter contre l'attaque Sybil. HAMACK est composé de Honeypeers distribués chargés de superviser et d'agir sur des contenus spécifiques partagés dans KAD en contrôlant l'indexation des mots-clés et des fichiers. Notre architecture permet de: (1) superviser de manière transparente toutes les requêtes destinées aux contenus ciblés dans le réseau, (2) éclipser les références malveillantes de la DHT, et (3) attirer les requêtes de téléchargement des pairs cherchant des contenus malveillants vers les Honeypeers en polluant ces références de la DHT avec des fichiers et des sources falsifiés. Les résultats du déploiement sur le réseau KAD réel montrent que notre méthode est applicable et très efficace.

Mots-clés : réseaux P2P, DHT, attaque Sybil, honeypot, KAD, supervision

1 Introduction

Peer-to-peer (P2P) networks are now commonly used to share files within the Internet. They offer lots of advantages compared to the client-server scheme by giving possibility to gather and share a large amount of resources with the collaboration of many individual peers. However, peer-to-peer networks also provide support for harmful and malicious activities that can voluntarily propagate strongly undesirable contents[1]. As peer-to-peer systems are self-organized, dynamic and do not have a centralized infrastructure, it is not obvious to collect information to measure them and to observe the behavior of malicious users.

With passive monitoring we can observe, from one point, the P2P traffic without sending additional data into the network. [6] collected 24 hours traces from an eDonkey server and pointed out the correlation between the set of peers active for some given data. [10] used passive monitoring at several routers to monitor flows in FastTrack, Gnutella and Direct-Connect. However, these approaches do not allow to study specific contents at the network scale. Active monitoring removes this drawback but is more intrusive in the sense that some traffic (queries, files) is injected in the network to gather more information concerning the P2P system. Many crawlers have been used to study the different P2P protocols like Gnutella [13], Napster [9], e-Donkey [4] [17] and KAD [15] [11]. Alone, a crawler can just observe the network without acting on it. In the case of KAD, a crawler just discovers the peers but not the shared contents. To have a better view of the network and to control it, [12] associated to the crawler a Sybil attack which consists in creating a very large number ($\sim 2^{16}$) of fake peers, controlled by one computer, and placing them actively in the part of the DHT to observe. Recent protection mechanisms have however been introduced [2] in KAD to make this intrusive approach inefficient.

In this paper, we propose to apply the concept of honeypot in the context of the KAD network, to monitor and act on specific malicious contents[2] and users in a widely deployed P2P system. A honeypot is classically used to attract malicious users trying to penetrate illegally in an unauthorized system. In the existing research works related to P2P honeypots [5] [1] [7], the solutions consist in advertising fake files as normal users in order to log the download queries received for these files, but without any guaranty to attract all peers looking for the studied content. Our solution surpasses classical honeypots because it allows to attract absolutely all the users searching for a given content, and also to control these malicious contents. Our approach is based on a P2P honeynet architecture composed of fully distributed honeypots we defined as Honeypeers. Its originality with regards to active monitoring is that we can transparently monitor all the requests sent to the targeted contents in the network, eclipse malicious entries of the DHT, and attract the download requests of malicious peers towards our Honeypeers by poisoning the DHT references with fake files and sources. Our strategy to take the control over DHT entries does not rely on Sybils injection and bypasses all the protection mechanisms introduced recently in KAD to fight against Sybil attacks.

This document is structured as follows. Section 2 describes the background of KAD, the Sybil attacks and the last inserted protections. We then present,

[1]This work is funded by the French ANR Rechearch Project MAPE (Measurement and Analysis of Peer-to-peer Exchanges for pedocriminality fighting and traffic profiling)

[2]given by a law enforcement agency

in Section 3, the approach to design HAMACK and its features allowing to monitor, eclipse and poison malicious contents of the network. The section 4 assesses the efficiency of our architecture by modeling the KAD search process. Section 5 and 6 respectively present the implemented architecture of HAMACK and the results of experiments done on the real KAD network. Finally, Section 7 concludes the paper and outlines our future works.

2 KAD and the Sybil attack

2.1 Overview of KAD

KAD is a structured P2P network based on the Kademlia distributed hash table routing protocol [8]. KAD is implemented by the eMule and aMule open source clients, allowing users to share files on this network. Mainly used in European countries and in China [11], the estimated number of concurrent online users is around 2.5 millions, which makes KAD the widest deployed structured P2P network today.

Each node of KAD has a 128bits "KADID" setting its position in the DHT. All the routing tasks are based on the XOR metric used to evaluate the distance in the DHT between two peers, or between a peer and a content. Two types of requests are used to discover the network: the *Hello_REQ* is used by a peer to announce itself, and the *Kademlia_REQ* is used to discover new peers close to a specific address. Some contacts are selected to fill the routing table following a specific scheme. The routing table is composed of groups of K-contacts (called a K-bucket), organised like a tree so that the group at level **i** has contacts at a distance between 2^{128-i} and 2^{127-i} from the current peer, regarding the XOR metric. In other words, the deeper the contact is in the tree, the closer it is to the current node and the better is its knowledge of this part of the DHT (knowing the same number of contacts for a zone always smaller). Routing is done in an iterative way and with parallel lookups, and will be described in detail further.

As a file sharing application, the purpose of the KAD DHT is to index files and keywords. When a new file is shared, the raw data and all the keywords composing its name are hashed separately with a MD5 function generating a KADID for each piece of information. Those KADIDs are then published in the DHT. The peers able to index a file or a keyword are all those that are close enough to the published hash. This distance is called the "tolerance zone" of a KADID, and is set to the first common 8bits (most significant). The double indexing allows the retrieval of a file, being given a set of keywords. To publish a file, two types of requests are sent:

- *KADEMLIA2_PUBLISH_KEY_REQ* requests are sent towards the hash of the keyword to associate a keyword with a file
- *KADEMLIA2_PUBLISH_SOURCE_REQ* requests are sent towards the hash of the file to associate a file with a source (a peer sharing the file)

After accepting a publication request for a given resource, a peer is in charge of indexing this specific content, and to answer to the related Search requests.

2.2 The Sybil attack

The Sybil Attack, as described by Douceur [3], consists in creating a large number of fake peers called the "Sybils", and placing them in a strategic way in the DHT to take control over a part of it.The efficiency of the Sybil attack is tight to the number of successfully injected Sybils, in spite of possible protective mechanisms of the P2P network. We will show that, in the case of KAD, another strategy can achieve the same results as a Sybil attack without triggering any protection.

Recent investigations showed that KAD was highly affected by the Sybil attack, resulting in important outages in the network by using few resources. Steiner et al [12] were the first to successfully launch a real Sybil attack on KAD, resulting in the full control of a part of the DHT. The attack is divided in two steps. The first step consisted in crawling the P2P network thanks to many route requests (*Kademlia_REQ*), to progressively discover more and more peers in the network. When almost all nodes of a zone are known by the crawler, the second step consisted in polluting the routing table of each node discovered with Sybils. Steiner et al injected 2^{16} Sybils from a single computer in a small zone of KAD. They were able to catch most of the Publish and Search requests within this zone. They achieved an eclipse attack making some keywords indexed on the Sybils disappear from the DHT.

Since, the latest versions of the major KAD clients have introduced new protection mechanisms based on local detection of suspicious behaviors, in order to limit the efficiency of the attacks. We described and evaluated these mechanisms in our previous work [2]. First, sending many requests from a single source is now detected and considered as flooding, which reduces the crawler efficiency. Second, it is no longer possible to infect a peer's routing table with Sybils showing the same IP address or very close IP addresses (same /24 subnetwork). Finally, the identity of a peer is now checked before being added or updated in the routing table. The IP address of a new peer is checked by a three-way handshake to avoid Sybils to use spoofed IP addresses. Moreover, KADID spoofing is also prevented by associating a public key to each KADID newly announced. KADID spoofing exploited in a large scale allowed to partition the DHT [16].

The new protection mechanisms avoid the injection of several Sybils from a single IP address. Doing a massive Sybil attack on KAD becomes extremely expensive regarding the number of IP addresses involved. Even the more localized attacks described in [12], which used a limited number of Sybils (32 Sybils to eclipse the particular keyword "the") is no longer possible. Indeed, the Sybils were directly injected in the routing table, which is now protected. Moreover, the crawler used to discover the network is now limited by the flooding protection. To design HAMACK, we had to adopt a novel strategy, neither relying on a crawler, nor on Sybils injection in routing tables.

We will show that localized attacks can still be effective with a reasonable amount of distributed resources. In fact, the main weakness allowing localized attacks to be performed is the possibility left to the peers to freely choose their KADID, which remains possible. Our assumption is that if a few numbers of modified clients choose their KADIDs very close to a given targeted hash (i.e. closer than any other peer randomly chosing its KADID), and considering the

Filename	Nb of Sources	Size
"Matrix 2 Reloaded dvdrip.avi"	362	695.21MB
"Matrix Revolutions spanish.avi"	328	625.23MB
"Game Enter the Matrix DVD.iso"	298	681.19MB
"Matrix Realoaded cd1 divx.avi"	8	679.74MB

Table 1: 4 results returned by a search request for the keyword "matrix"

very efficient KAD search algorithm "return the X closest nodes...", the requests will arrive at these nodes with a very high probability.

3 HAMACK features

This part aims to describe our honeynet architecture and its features. As they are distributed and autonomous, we prefer to call our modified clients "Honeypeers" rather than Sybils.

3.1 Our approach to design an attractive Honeypot

When designing HAMACK, our objective was to make the most attractive Honeypot for malicious users in KAD. A naive honeynet approach would contain several clients that announce fake contents. But expecting a good visibility for the files promoted by such honeynet would involve a huge amount of resources. In fact, when a user searches for a given content, several different pieces of information are returned to help him find the right file. Table 1 shows an example of a search results for a typical shared content.

The name of the file and its size are important data that can be controlled by a classical Honeynet, whereas the number of sources can not be handled easily. Unfortunately, this information is capital because it allows to sort the search results, the best files being those with the highest number of sources. Files with a high number of sources are very popular, and consequently, more trustworthy and downloaded faster than the others. Typically, the last result of the search shown in table 1 should be discarded by users. While in eDonkey the number of sources is returned by the server, in KAD, it is computed by the peers in charge of indexing this file.

Previously, the easiest solution to increase the estimated number of sources displayed for a given file was to publish it with many Sybils; but it has become really resource consuming since IP address limitations have been set in KAD. Another solution consists in taking control over the indexing scheme of KAD to control all the information returned about the file, including the estimated number of sources.

Finally, by controlling the indexation mechanism, many more applications are possible. The features of HAMACK are the following, all appearing transparent for the final user:

- Passive monitoring: monitoring all the Search and Publish requests for malicious keywords and files

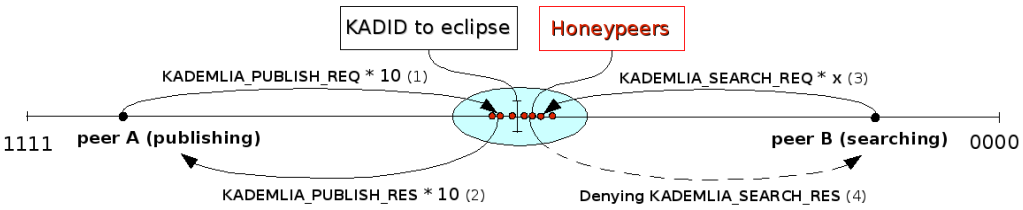

Figure 1: Message exchange when eclipsing content

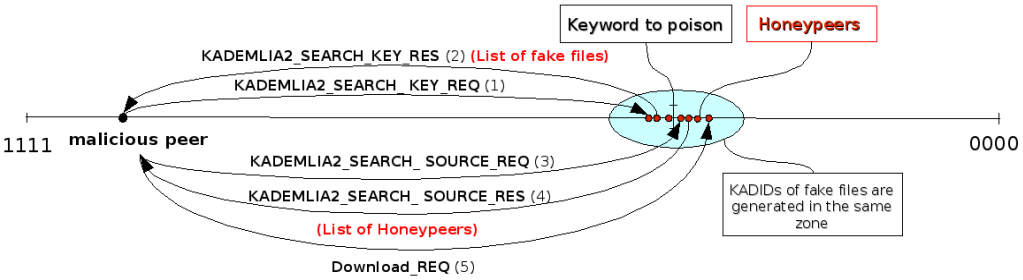

Figure 2: Message exchange when poisoning a keyword

- Eclipsing contents: removing malicious keywords and files from the network
- Index poisoning: replacing the malicious files linked to a keyword with fakes showing a high number of sources
- Promoting Honeypots: replacing the peers sharing a file with Honeypots

They are described in details in the forthcoming subsections and in figure **??**.

3.2 Passive monitoring and eclipsing contents

The simpliest and quietest behavior consists in transparently monitoring the network by logging all the requests passing through the Honeypeers, particularly those concerning the targetID. Then, the Honeypeers answer to each request like a normal client, not disturbing the network. When a *KADEMLIA2_PUBLISH_KEY_REQ* message is received, HAMACK stores the following information from the request message: the sender IP address and port, the keywordID, the list of fileIDs and for each fileID the list of tags containing the file properties (full name, size). A *KADEMLIA2_PUBLISH_SOURCE_REQ* message contains fewer fields: the sender IP address and port, the fileID, the KADID, IP address and port of the source. This feature allows HAMACK to monitor the activity of malicious files (what peers are sharing them) and of malicious keywords (what are the new published files).

The second behavior aims to eclipse a target ID from the network as described in figure 1. To eclipse a keyword or a file, the Honeypeers have to behave differently. First, the Honeypeers have to bypass two constraints concerning the maximum number of references that a peer can store. The first

constraint limits the total number of references that any peer can store in order to limit the weight of a single peer in the network. The second constraint limits the maximum number of references stored for a particular KADID (either keyword or file) to avoid that too popular entries become over-referenced against the others. Our Honeypeers bypass these limitations in order to always answer positively to any Publish request. Otherwise, if these limits were reached, denied requests can be sent to normal peers and escape to HAMACK. Second, the Honeypeers have to deny all the incoming Search requests for the target ID. As HAMACK manages to acknowledge all the Publish requests, denying the Search requests simply removes the contents of the DHT. This feature is interesting because it allows HAMACK to remove malicious contents from the network, and by doing so, to prevent users from accessing it.

3.3 Index poisoning and promoting Honeypots

The index poisoning is the most interesting feature of HAMACK. It makes possible to capure all requests emitted by a malicious peer from the beginning, with the keyword search, to the end, with the final download request. The first step is the same as eclipsing a keyword by attracting all the Publish requests for the targeted keyword. Then, instead of denying the Search requests, all the Honeypeers composing HAMACK will answer with a list of fake files when requested. Because HAMACK has a total control of the responses, each fake file will appear to be very attractive, with a high number of sources. We can choose whether all the list is composed of fakes, or only a part of it. In the first case, setting properly the parameters of the list of fake files is important in order not to be detected, as all the good references will be eclipsed.

Then, when a malicious user selects a fake file, HAMACK must receive the Search sources requests to finally answer with the KADIDs of the Honeypeers, as shown in figure 2. To capture easily the Search Source requests for all the fake files, the fake FileIDs are generated to be extremely close (96bits) to the targeted keyword. With this optimization, every Search Source request will be attracted in the same way than the initial search of keyword, the Honeypeers staying in the same place of the DHT. When sending the Search Source responses, HAMACK provides the list of the Honeypeers (KADID,IP,port) in order to attract the final Download request of the malicious user. The relevancy of this feature is to assess the intention of the malicious user. HAMACK does not only assume that a user is malicious based on the initial search of keywords, but proposes fake files characterised by full malicious names, and checks the malicious behaviour with the final Download request.

4 HAMACK exploiting the KAD search procedure

The key point of our architecture is the control of the indexation mechanism. This part aims to demonstrate how efficient our architecture is against KAD publication and search functions. We will analyse in the forthcoming section the overall publish and search procedures in KAD, and describe in details how our Honeypeers take advantage of it. In opposite to [12], we do not consider the

direct injection of Sybils in routing tables, but we focus our study on controlling the sequence of events and messages happening during the search process, without abnormally affecting the routing tables.

4.1 Positioning the Honeypeers

The first step to settle our honeynet is to position precisely the Honeypeers according to the keyword or the file over which we want to take the control. So, the main parameter to consider is the targeted KADID. To compute it and get the 128bits address, we need to apply the same MD5 hash function as the one used in a KAD client to the keyword or the file. When this KADID is known, it is given as a parameter to each Honeypeer which can then derivate its own KADID from it. This derivation function simply copies the first 96bits from the targeted KADID and chooses the remaining 32bits randomly.

Even if the Honeypeers do not synchronise themselves when choosing their KADID, the risk of collision for few Honeypeers on 32bits is insignificant. Let N be the mean number of peers closer to the target than one of the Honeypeers. N is given by the probability for a peer to randomly choose a KADID with the same 96bits than the Honeypeers multiplied by the number of peers in the network. According to the latest estimations, the number of peers participating in KAD remains by much under 5 millions. The result in (1) shows that 96bits is a sufficient prefix to be sure that no other peer will be placed closer to the target that our Honeypeers.

$$N = \frac{2^{32}}{2^{128}} \times 5 \times 10^6 = 6.31 \times 10^{-23} \quad (1)$$

4.2 Attracting the requests during the search procedure

As our Honeypeers are the closest to the targeted content, the critical point is to be sure that they will be able to attract all the requests, given the publish and search functions of KAD. We will show in this subsection that KAD is very efficient in routing, and that is its great weakness when considering localized attacks.

All services in KAD involving lookups in the DHT are achieved thanks to the "Search" object. Every keyword or file that has to be published, or every search done by a user, will generate an autonomous "Search" object in charge of all the process until its ending. Several Search objects can be managed by a client at the same time for different purpose, but all are distinct and autonomous.

The search process can be divided in two separate parts. The first part consists in finding online peers in the tolerance zone of the target to fill the array of "*possible*" contacts in the Search object. This array contains all the suitable contacts found and ordered by distance. This part is done by sending several *KADEMLIA_REQ* in parallel with the target ID as parameter. Routing is done in an iterative way with parallel lookups, as described in figure 3: asking at first the 3 closest contacts of the routing table for even closer nodes toward the target ID, waiting for their responses and then reiterating by asking the best of the newly received contacts. Only the contacts that have successfuly answered to a *KADEMLIA_REQ* are considered as available and can be used for the next step. While closer contacts can be found in the last responses,

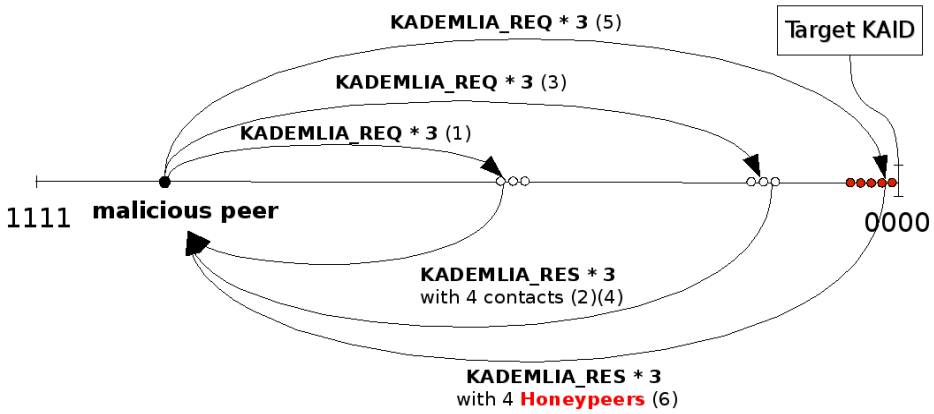

Figure 3: Message exchange in the search procedure

the Search keeps looking for even closer contacts, without sending the specific service requests. In the last version of the KAD clients (eMule 0.49c and aMule 2.2.4), the contacts received from a *KADEMLIA_RES* are now verified in the same way than contacts added in the routing table, in particular: only one KADID is allowed per IP address, and no more than 2 IP addresses are allowed from the same /24 subnet. These new constraints also motivate our distributed architecture.

The second part consists in sending specific service messages to the "*possible*" peers, regarding the type set for the search object (for example *KADEMLIA2_PUBLISH_KEY_REQ* to publish a keyword), and then analysing the responses to continue or stop the Search. The effective starting condition of the function sending service requests is important. In fact, this function is executed when the Search has not received any new contact within the 3 last seconds, which means that the last found peers fail to provide any closer contacts to keep the Search going on. Then, the service requests are sent to the closest peers discovered. The stop conditions of the Search are the following: if it got more positive responses for the service than the replication rate, or if it triggers a timeout. To protect from churn, a publication is considered as successful when at least 10 peers have acknowledged it.

When a *KADEMLIA_REQ* arrives on a Honeypeer for the targeted hash, it answers with the list of all the other Honeypeers composing HAMACK and activated for this content. As the list of contacts managed by the Search object is always sorted by distance, all the Honeypeers are placed in the best positions to receive service requests. So, as soon as a publishing peer gets one of the Honeypeers among the routing responses, all the following specific requests emitted by this peer will be sent to HAMACK. Therefore, the probability to attract all the requests emitted by a peer is approximated by the probability to find at least one of the Honeypeers before the Search object begins the transmission of service requests, we calculate this probability in the following paragraph.

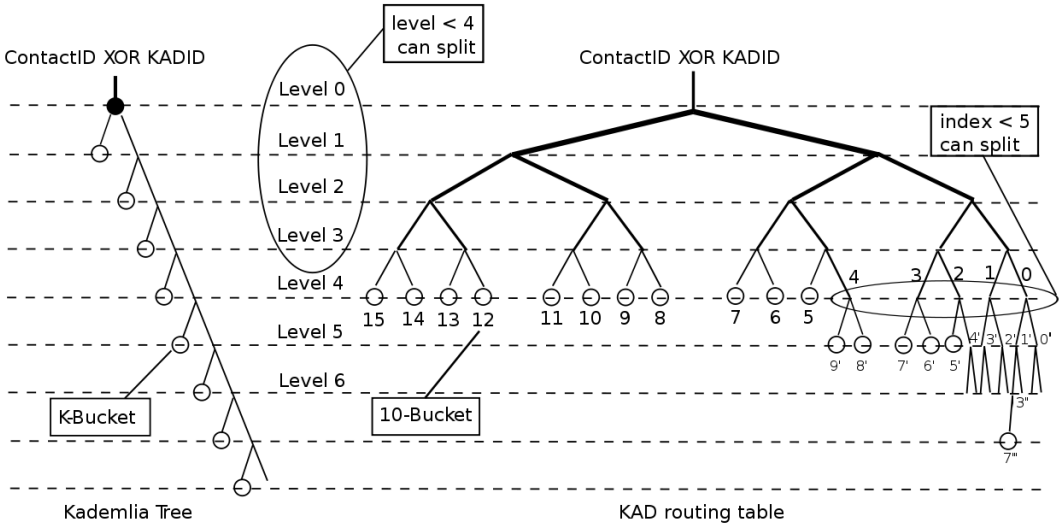

Figure 4: KAD routing table scheme

4.3 Analysis of HAMACK efficiency

Since the first study on the KAD routing scheme [14], it has been confirmed [16] [2] that the routing table of KAD is slightly different from the proper binary tree defined in Kademlia [8]. Figure 3 compares the two trees. Basically, the routing table of KAD is far more efficient because more contacts are stored at each level. With the knowledge of the routing table and of the search procedure, we are able to estimate the probability to attract the first service request for a target of HAMACK, considering the distance of the source.

We assume that the initial peer and each intermediate step have a full routing table filled according to their respective place in the DHT. Moreover, we consider that the initial peer is far from the tolerance zone of the target so that it will need more hops before finding the Honeypeers. We also make the assumption that there is no partition in the P2P network and that the Honeypeers are well referenced by the other peers.

So, let H be the number of Honeypeers returned at a given step of the Search, and $(1 - P_n(H = 0))$ or $P_n(H \geq 1)$ be the probability to find at least one Honeypeer at the n^{th} step of the Search. As the Honeypeers know each others and collaborate through *KADEMLIA_RES*, we consider that the fact to find at least one Honeypeer leads to the capture of all following requets by HAMACK. Let NH be the number of Honeypeers used by HAMACK and NP_n the number of potential peers that can be chosen at step n of the Search. At the begining of the lookup, the peer has to bootstrap the Search by requesting the 3 closest contacts of its routing table (equation 2). Then at each step (equation 3), 3 routing requests are sent towards the closest contacts and each will return 4 responses among a choice of NP_n closer contacts, so that the Honeypeers have to be found among the 12 responses. To find $P_n(H)$, we can compute the probabilities to choose i Honeypeers among NH at each step, which is represented by the hypergeometric law with parameters (12, NH, NP_n+NH)

and to sum all the positive values of i.

$$P_0(H \geq 1) = \sum_{i=1}^{i=3} \frac{C_{NH}^{i} \times C_{NP_0}^{3-i}}{C_{NP_0+NH}^{3}} \tag{2}$$

$$P_n(H \geq 1) = \sum_{i=1}^{i=12} \frac{C_{NH}^{i} \times C_{NP_n}^{12-i}}{C_{NP_n+NH}^{12}} \tag{3}$$

Let NP be the total number of peers in the network. Looking at the real KAD routing table, the initial peer has at least one 10-bucket to cover 1/16 of the network. So, the first 3 contacts used to bootstrap the Search are chosen among $NP/2^4$ peers. Within a 10-bucket, the contacts are not ordered but we have to translate the fact to pick up the 3 closest contacts in probability. We assume that the KADIDs within a 10-bucket are uniformly distributed, so that choosing the 3 closest contacts among 10 improves, in average, 2 more bits of the target. So a more realistic value is:

$$NP_0 = \frac{NP}{2^6} \tag{4}$$

Then, looking at the figure 3, the contacts requested for the next step have a view of the zone considered for the target at least 3 times better. In fact, the inital peer, being far away from the target, picked up the contacts to bootstrap the Search from one of its top 10-bucket with an index between 5 and 15. These bootstrap contacts, when requested, will then look in their bucket with index 0 to find closer contacts. In the worst case, the targetID will hit the following 10-buckets with the indexes 1'-3"-7"' to find the next contacts. That means that in the worst case, considering the routing table of KAD, the Search will move forward by 3bits. As the structure of the routing table does not change with the depth, this reasoning is true for every step. Finally, if we want the worst case to be realistic by adding 2 more bits when choosing the best 3 contacts in a bucket, each step improves the Search of 5 bits in our model.

$$NP_n = \frac{NP}{2^6 \times 2^{5 \times n}} \tag{5}$$

The numerical application of these probabilities with the parameters ($NP = 5 \times 10^6$,$NH = 20$) are displayed in table 2. We see that at least one Honeypeer will be returned with nearly 10% of probability after the first step, and with 93% after the second. This result demonstrates that the routing in KAD ensures that the Honeypeers will be discovered during the Search process and attract all the next service requests. Practical experiments will confirm these results.

5 Implementation

This part describes how HAMACK is implemented to achieve its purpose.

5.1 Honeypeers

The main constraint to be addressed by HAMACK comes from the IP address limitation inserted in the lastest version of the major KAD clients. So, all the

n	$NP_n + NH$	$P_n(H \geq 1)$
0	78145	0.00077
1	2461.5	0.0931
2	96.5	0.928
3	22.4	1

Table 2: Probability to find at least 1 honeypeer at the n^{th} step of the search

Honeypeers running for a specific target ID must have different IP addresses in order not to be denied. Moreover, no more that 2 Honeypeers can belong to the same subnet. To fit with these constraints, we chose to run HAMACK from a slice of PlanetLab Europe, as described by figure 5.

The Honeypeers execute a modified aMule client in its daemon version (aMuled) which is light and remotely manageable. This client has been modified in order to achieve the features described in section 3. Basically, many functions processing the incoming packets and many parameters of KAD were adapted. The behavior to adopt for a Honeypeer (spying, eclipsing, poisoning), the target ID and other parameters are set in a specific configuration file read by the modified client.

Several Perl scripts were written to easily deploy HAMACK on PlanetLab and change the behavior of the Honeypeers when needed. They are gathered and executed from a separate computer called the Manager. Writing these scripts was facilitated by the homogenous architecture of PlanetLab, so that each Honeypeer runs the same task in the same environment. The first version of HAMACK only manages one target ID but the scalability is not an issue. Given the very low resource consumption of aMuled, a single node can easily handle dozens of clients on different ports, so that HAMACK will investigate many contents at the same time in the next release.

5.2 Database

The second major component of HAMACK is the database that stores the information captured by the Honeypeers. The database is implemented in a PostgreSQL server and secured by an additional SSL module to avoid unauthorised connections. Most of the access to the database are writing but the Honeypeers can also read the different tables if needed: for example to get the KADID,IP and port of the other Honeypeers in charge of the same target ID to collaborate. We designed the different tables to easily anonymize the IP address of the peers if needed. When a new IP address is seen by HAMACK, the database attributes a new index to it and remembers this reference for a whole execution. Finally, the IP address field can be dropped without loosing too much information.

6 Measurements

To develop and evaluate HAMACK, we made several experiments on the real KAD network. As we absolutely wanted to not disturb the network, we chose the keyword "document" as our test target ID because it is quite well indexed but insignificant when doing a research on KAD. In a first time, we study

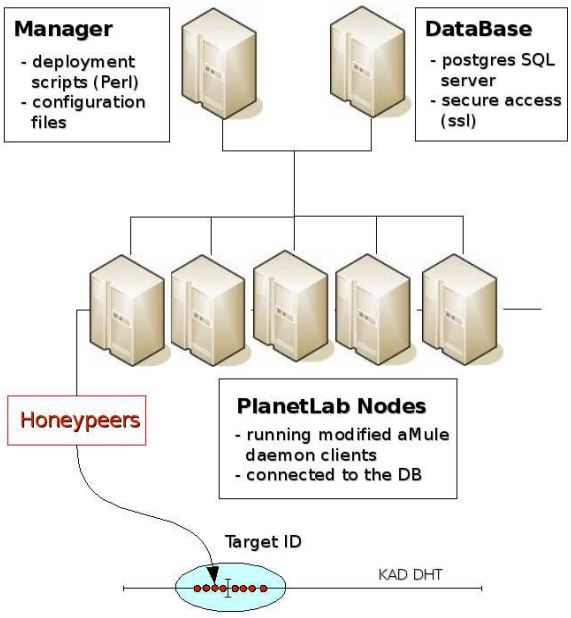

Figure 5: Network architecute of HAMACK

different parameters affecting HAMACK efficiency before evaluating the final architecture.

6.1 Setting the parameters

The execution time needed for an experiment was the first parameter to study for two reasons. On one hand the uptime of a peer is a classical parameter used to distinguish which peer should remain in the routing table to improve its stability, so that peers staying for a long time in the network are better known. On the other hand, our Honeypeers implement a function that makes them periodicaly announce themselves in the network. Graph 6 shows, for every interval of six hours, the number of Publish requests received by HAMACK over 4 days. We waited two hours before counting the requests to have the Honeypeers known in the network. We can clearly see that on the medium and long run, the time has absolutely no impact on HAMACK efficiency. We also see that every day, the number of requests received during the afternoon ($\sim$ 40000) between 12h-18h or 18h-00h (GMT+1), is higher that in the morning ($\sim$ 30000) between 00h-06h or 06h-12h. This result is coherent with the fact that KAD is mainly used in European countries, as presented in [11]. According to this result, the next experiments are based on a full day capture.

The number of Honeypeers is also an important parameter to be able to efficiently use our resources. The minimum number of Honeypeers to involve should be 10, like the replication factor to publish a content. But usually, more than 10 service requests can be sent and all have to be captured. The reasonable minimum number of Honeypeers tried in our experiment is 15. Graph 7 shows the evolution of HAMACK efficiency when the number of Honeypeers

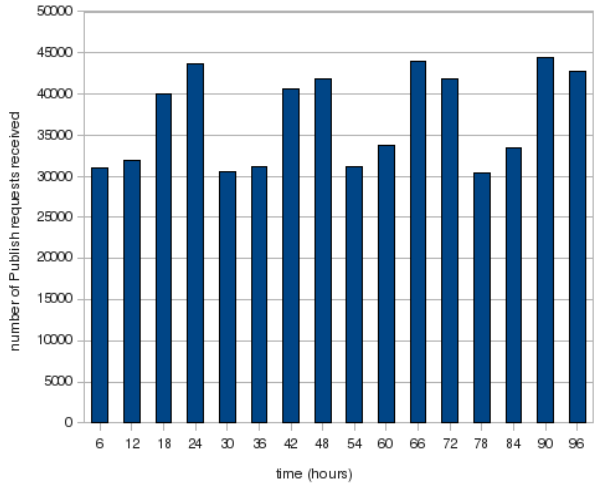

Figure 6: Impact of the execution time on HAMACK efficiency

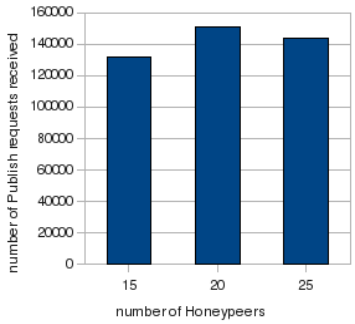

Figure 7: Impact of the number of Honeypeers on HAMACK efficiency

varies. The results show that 15 Honeypeers achieve a very good attractivity for HAMACK but using 20 Honeypeers improves the efficiency. Using 25 Honeypeers shows an insignificative decrease of performances, simply meaning that there is no purpose to use more that 20 Honeypeers.

After studying the environment parameters, we investigated three parameters related to the Honeypeers behavior. Firstly, we condidered the impact of the collaboration among Honeypeers. In the first case, the Honeypeers do not have a priori knowledge of the others and simply discover themselves through KAD mechanisms that fill their routing table. In the second case, the Honeypeers explicitely know each other and always promote themselves in *KADEMLIA_RES*. Figure 8 shows how this modification performs. We can see that the collaboration of Honeypeers increases the efficiency of HAMACK ($\sim 22\%$). This means that, by default, few normal peers were announced in the *KADEMLIA_RES* returned by the Honeypeers for the target ID, decreasing the performance of HAMACK.

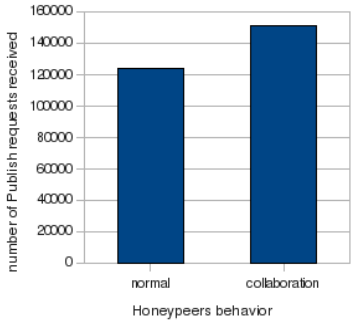

Figure 8: Impact of the collaboration between Honeypeers

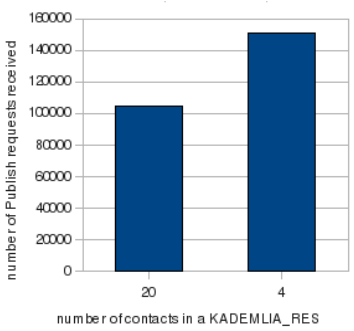

Figure 9: Impact of the size of Kademlia_RES

Secondly, we defined the strategy of collaboration to use. Two strategies are possible. The first is compliant with the very last version of the KAD clients (eMule 0.49c ans aMule 2.2.4) which drops any *KADEMLIA_RES* containing more that 4 contacts. The second uses the full list of Honeypeers in the *KADEMLIA_RES* which should give better results with oldest clients. This protection being very recent (less than 2 months), we thought that most of clients would not have been updated. Figure 9 shows how many Publish requests are received in one day with both strategies. When fitting the protection against the flooding of contacts in *KADEMLIA_RES*, the number of Publish requests received increases a lot ($\sim 45\%$). This means that clients are quickly updated by the users after a new release and that this new constraint has already to be considered.

Finally, we measured the impact of active announcements. The result of figure 10 does not show a significative improvement of HAMACK efficiency when forcing the Honeypeers to announce themselves actively in the network, to gain visibility. The basic KAD behavior is sufficient to be assured that a peer is well known in the network.

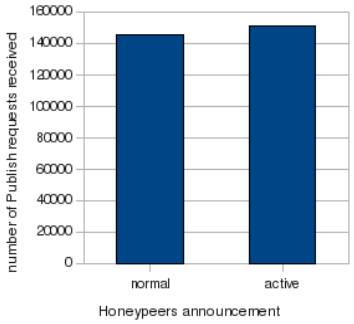

Figure 10: Impact of active announcement on HAMACK efficiency

6.2 Evaluation of HAMACK efficiency

Evaluating the efficiency of our architecture is a real challenge. How to be sure that HAMACK effectively attracts all the emitted Publish requests? As we can not have a probe behind each peer of the KAD network during our experiments, we designed two experiments to give us a good indication of the efficiency of the architecture.

The first experiment consisted in publishing a keyword managed by HAMACK from 6 different sources that we monitored. Each source had a different place in the DHT and was bootstraped from a different set of contacts, in order to avoid any misleaded measurements due to unproper initial conditions. Then, we can compare for every publication, how many sent requests were captured by HAMACK and if the distance of the publishing peer affects the performance of the architecture. Figure 11 shows that all the Publish requests sent by our monitored clients are captured by the Honeypeers. Moreover, these very good results were confirmed by several search attempts performed on the targeted keyword "document" when eclipsed by HAMACK: all the Search requests were captured by the Honeypeers, and the global search finished without finding a single result.

The second experiment was to consider how many Publish resquests are seen by HAMACK from the same peer in a short period of time. We know that the publication process waits for 10 positive responses before stopping to send Publish requests. Given that, each time that HAMACK sees less than 10 times the same request from a peer during the short period of time of a publication process, it means that the other requests have been received by normal peers. As our architecture also attracts the Search requests, HAMACK can still control the content even if few Publish requests are missed. This experiment is a good indicator of how HAMACK performs. The results displayed in figure 12 are extremely encouraging. Very few distinct publications are seen with less than 8 requests (509/5669). As expected, a very high number of distinct publications are seen around 10. The high number of publications for which more that 10 requests are captured can seem to be strange. In fact, during the publish process, many *KADEMLIA2_PUBLISH_KEY_REQ* messages can be sent when the peer has almost finished the publication (for example 9 acknowledgments

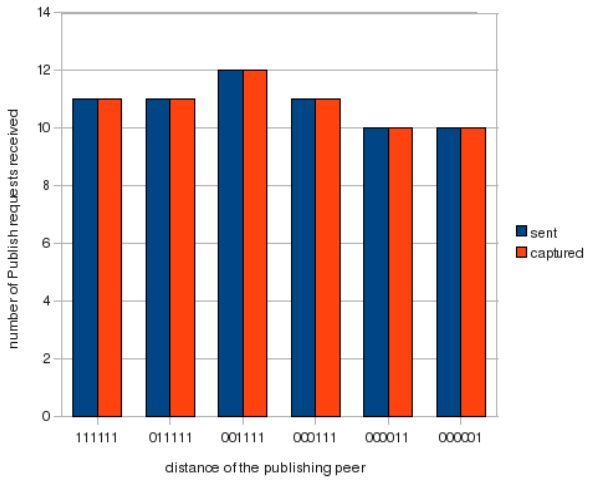

Figure 11: Number of Publish requests sent and captured considering the distance of the publishing peer

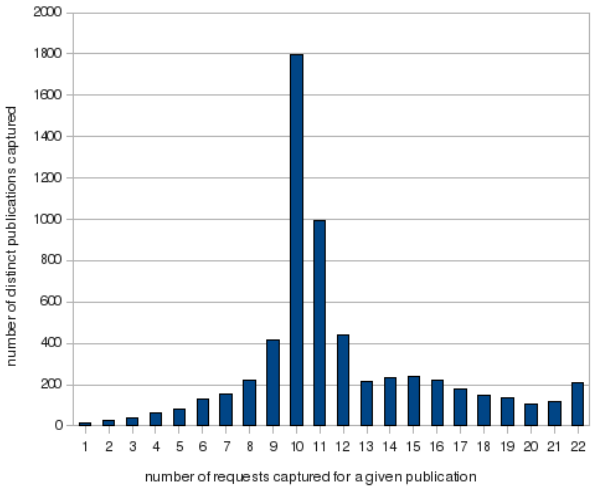

Figure 12: Number of Publish requests received for each replication rate

out of 10 needed). When receiving the 10^{th} response, the Search will stop but all the previously sent requests will still be considered and processed.

We also evaluated how well HAMACK is distributed on the Honeypeers. To do so, we counted the number of Publish requests captured by each Honeypeer in one day. We found out that the load distribution between Honeypeers is related to the distance between each Honeypeer and the target. Figure 13 shows the load on every Honeypeers ordered by their distance to the target. Even if the Honeypeers share 96bits with the target, the freedom to choose their KADID in the left 32bits is sufficient to see the efficiency of the KAD search algorithm

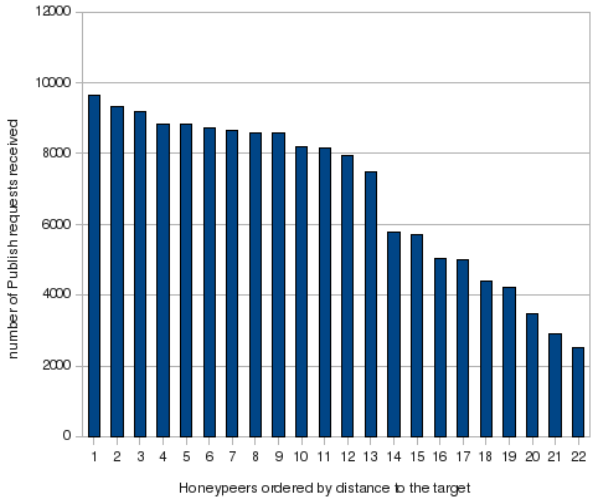

Figure 13: Distribution of the load on the Honeypeers composing HAMACK

trying to find the closest contacts. We can clearly see in the results that the Honeypeers with IDs (1-13) chose the same 97th bit than the target, while the Honeypeers (14-22) chose the other value, resulting in a sensible decrease of their attractivity compared to the others.

Finally, we evaluated the attractivity of fake files announced by HAMACK. In this last experiment, HAMACK poisoned the keyword 'spiderman' with 4 files, 2 well shared and 2 with a low number of sources, while eclipsing the other references for this keyword during one day. The search results returned by a KAD client during our test are displayed in figure 15. We can see that our attack succeeds to eclipse the real entries and to replace them with our 4 fake files. Figure 14 shows for each of these fake files, the proportion of first Search Source requests received from a distinct peer. The results show that the well shared files are chosen in first by 96% of the users, which confirms the great importance of the number of sources. This experiment also illustrates how malicious users will be cheated by HAMACK in future real applications.

7 Conclusion

We have described HAMACK, an efficient honeynet that bypasses the most recent protections of KAD against the Sybil attack. HAMACK uses several Honeypeers set very close to malicious references of the DHT and these Honeypeers are able to take control over them. Our appproach does not rely in the injection of Sybils and is absolutely non intrusive for the network besides the targeted contents. Quiet monitoring of all the incoming requests and eclipsing the malicious contents are some interesting features of HAMACK, to study and protect the network. But the most accomplished feature is the possibility to announce many files for a given keyword with realistic and attractive attributes, in particular the number of sources. Through the announcement of fake files, HAMACK is able to attract and capture all the requests of a malicious peer:

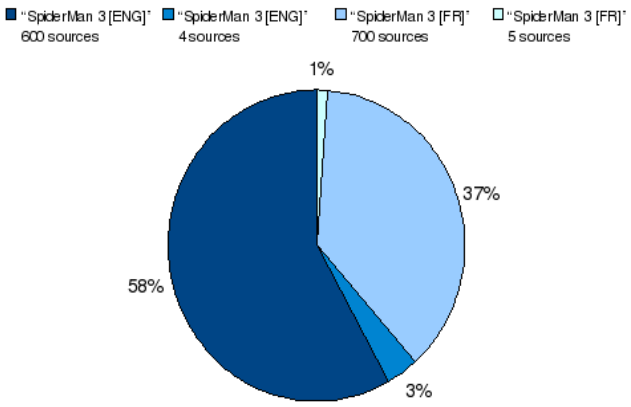

Figure 14: Proportion of first Search Source requests received for each fake file announced by HAMACK

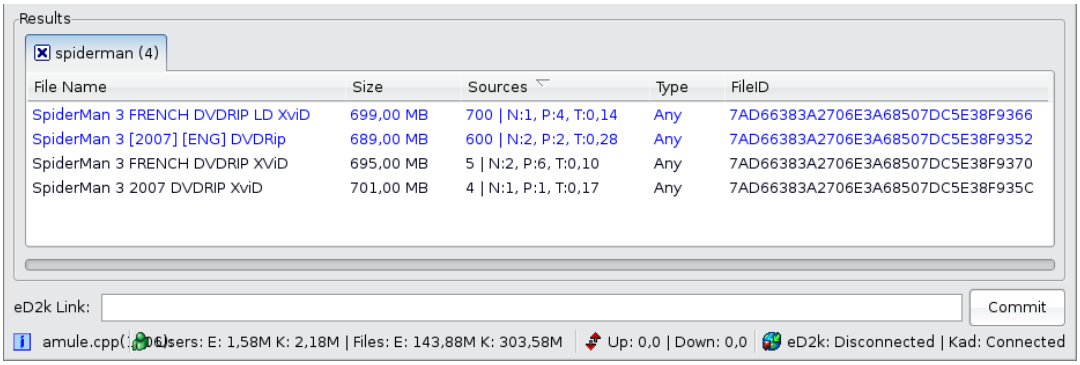

Figure 15: Result of a search for "spiderman" under eclipse and poison (4 fake files)

from the search of a keyword, to the final download request, assessing the actions of malicious users.

To achieve these features, HAMACK exploits the weakness of KAD allowing to freely choose the KADID of a peer and relies on the very efficient search process of the KAD DHT. As described in our model, a search launched on a target of HAMACK will be captured by the Honeypeers with a very high probability ($\geq$ 93%). Our work highlights a new dilemna of KAD which has to choose between its routing efficiency and the safety of its indexed contents. HAMACK is implemented by a lightweight architecture and fully fonctionnal. It uses modified aMuled clients deployed on PlanetLab nodes, and coupled with a secured database. The first experiments run on the real KAD network helped to set the parameters of the architecture. They showed 3 important results: 1- the coordination between Honeypeers increases the efficiency of HAMACK, 2- the architecture has to fit with the latest constraints inserted in KAD and 3- a low upper bound of needed Honeypeers. Then, several experiments were run and showed that HAMACK is extremely efficient to attract all the requests

of the target IDs, resulting in the total control of the contents. Finally, our final experiment poisoning a real content confirmed the great importance of controlling the number of sources to make an efficient honeypot.

Our future work will consist in enabling HAMACK to target many IDs at the same time, by running several Honeypeers by node. Then, we will use HAMACK to study and fight against different types of malicious contents spreading in P2P networks, like viruses, or more generaly to study any phenomenon related to contents in P2P networks. To close the loop, we work on a revocation mechanism to protect the network against malicious users detected by HAMACK.

Acknowledgment: This work is partially supported by the French Ministry of Research project, MAPE, under contract ANR-07-TLCOM-24 and by the EC IST-EMANICS Network of Excellence (#26854).

References

[1] R. Badonnel, R. State, I. Chrisment, and O. Festor. A management platform for tracking cyber predators in peer-to-peer networks. In *ICIMP 2007. The Second International Conference on Internet Monitoring and Protection*, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[2] T. Cholez, I. Chrisment, and O. Festor. Evaluation of sybil attacks protection schemes in kad. In *AIMS 2009: 3rd International Conference on Autonomous Infrastructure, Management and Security*, Twente, The Netherlands, 2009. Springer-Verlag.

[3] J. R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[4] F. Le Fessant, A.-M. Kermarrec, and L. Massouli. Clustering in peer-to-peer file sharing workloads. In *in Proceedings of IPTPS*, 2004.

[5] M. Latapy, O. Allali, and C. Magnien. Measurement of edonkey activity with distributed honeypots. In *HOTP2P 2009. Sixth International Workshop on Hot Topics in Peer-to-Peer Systems*, Rome, Italy, 2009.

[6] S. Le-Blond, J.-L. Guillaume, and M. Latapy. Clustering in p2p exchanges and consequences on performances. In *IPTPS*, pages 193–204, 2005.

[7] H. Lee and T. Nam. P2p honeypot to prevent illegal or harmful contents from spreading in p2p network. volume 1, pages 497–501, Feb. 2007.

[8] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[9] S. Saroiu, K. P. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *MMCN '02*, San Jose, CA, USA, January 2002.

[10] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, 2004.

[11] M. Steiner, T. En Najjary, and E. W. Biersack. A global view of KAD. In *IMC 2007, Internet Measurement Conference, October 23-26, 2007, San Diego, USA*, Oct 2007.

[12] M. Steiner, T. En Najjary, and E.W. Biersack. Exploiting KAD: possible uses and misuses. *Computer communications review, Volume 37 NÂ°5, October 2007.*

[13] D. Stutzbach and R. Rejaie. Capturing accurate snapshots of the gnutella network. In *8th IEEE Global Internet Symposium*, March 2005.

[14] D. Stutzbach and R. Rejaie. Improving lookup performance over a widely-deployed dht. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*, Barcelona, Spain, 2006.

[15] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202, New York, NY, USA, 2006. ACM.

[16] P. Wang, J. Tyra, E. Chan-Tin, T. Malchow, D. Foo Kune, N. Hopper, and Y. Kim. Attacking the kad network. In *SecureComm 2008: the 4th International Confere on Security and Privacy in Communication Networks*, Istanbul, Turkey, 2008. ACM.

[17] J. Yang, H. M., W. Song, J. Cui, and C. Zhou. Crawling the edonkey network. In *GCCW '06: Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops*, pages 133–136, Washington, DC, USA, 2006. IEEE Computer Society.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399