Evaluating and Improving the Content Access in KAD

Moritz Steiner, Damiano Carra and Ernst W. Biersack
Institut Eurécom, Sophia–Antipolis, France
{steiner|carra|erbi}@eurecom.fr

Abstract

We analyze in detail the content retrieval process in KAD. KAD implements content *search* (publish and retrieval) functions that use the Kademlia Distributed Hash Table for content *routing*. Node churn is quite common in peer-to-peer systems and results in information loss and stale routing table entries. To deal with node churn, KAD issues parallel route requests and publishes multiple redundant copies of each piece of information. We identify the key design parameters in KAD and present an analytical model to evaluate the impact of changes in the values of these parameters on the overall lookup latency and message overhead. Extensive measurements of the lookup performance using an instrumented client allow us to validate the model. The overall lookup latency is in most cases five seconds or larger. We elucidate the cause for such high lookup latencies and propose an improved scheme that significantly decreases the overall lookup latency without increasing the overhead.

1 Introduction

In recent years a large number of Distributed Hash Tables (DHTs) systems, such as Chord [15], CAN [11], or Pastry [12], has been proposed. There are mainly two basic approaches for solving the problems related to the search of the content: structured ones, using a Distributed Hash Table (DHT) and unstructured ones based on flooding or random walk. Despite the large effort devoted to the topic, only few systems have been successfully deployed.

In this paper we focus on the DHT adopted by different clients that accounts for million of users: KAD, the implementation of Kademlia [9] contained eMule [4], aMule [1], and AZUREUS [2]. Overnet [10] uses Kademlia as well, but an older implementation. We consider the functionalities related to content management: lookup for searching the target peers, content publishing and content retrieval. Among the different publishing schemes, KAD adopts a publishing node centric approach: the responsibility of the content and its maintenance is with the *publishing* node, while the references to it are announced and stored into the P2P system.

A major issue in P2P networks is *churn*, i.e. node arrivals and departures that make the system volatile. In order to make the references available despite node dynamics, a peer in KAD publishes multiple copies (replicas) of a reference by selecting different nodes around the target, which is determined by the key of the reference. As the time goes by, some replicas may disappear, or new peers may arrive and take place in between the peers holding replicas. The actual location of the references is then scattered: some entries in the routing tables may be missing since the peers arrived recently or may be stale since the peers already left the system.

In case of content retrieval, where these references are searched around the target where they should be published, robust search mechanisms are necessary.

The aim of our study is to evaluate the performance of the current implementation of content management in KAD. We identify its basic building blocks and we analyze the interactions among them. The main contribution of our work can be summarized as follows:

- We develop a qualitative analysis of the current implementation to understand the impact of the design parameters on the latency of the overall content publishing/retrieval process;
- We obtain through measurements many interesting properties of the KAD P2P system, such as the probability that an entry in the routing table is stale, or the round trip delay of the messages;
- We evaluate through measurements the key performance metrics, such as overall content retrieval latency, the number of hops needed, and message overhead of the content retrieval process;
- We propose an alternative approach for the content retrieval process – called *Integrated Content Lookup* – by strongly coupling the retrieval with the lookup process and we develop a qualitative analysis of this scheme.

The analysis highlights some performance issues with the current implementation: the decoupling of the lookup phase and the content retrieval phase has an adversarial impact on the performance of the retrieval process. These issues are addressed by the Integrated Content Lookup scheme we propose. The measurement-based characterization of the KAD P2P system shows that (i) a large fraction of peers in the routing table that are stale and (ii) the empirical distribution of the message delay presents a non-negligible tail. These results should be taken into account in the design of the content management process, since they have a strong impact on the overall lookup delay.

The remainder of the paper in organized as follows. In Section 2 we give some background on KAD, the architecture, the content lookup, and the content retrieval. In Section 3 we analyze the content retrieval process and identify the impact of the main parameters on the overall lookup latency. In Section 4 we present the results of our measurements and analyze the impact of the different parameters. We propose an improved content retrieval procedure in Section 5. In Section 6 we discuss the related work before we conclude.

2 Content Publishing and Retrieval in KAD

Similar to other DHTs like Chord [15], Can [11], or Pastry [12], each KAD node has a global identifier, referred to as KAD ID, which is 128 bit long random number generated using a cryptographic hash function. The KAD ID is generated when the client application is started for the first time and is then permanently stored. In KAD the distance between two nodes is measured considering their KAD IDs: in particular it is calculated as bitwise XOR, i.e. the XOR-distance $d(a, b)$ between nodes a and b is $d(a, b) = a \oplus b$.

2.1 KAD Architecture

The basic operations that each node has to perform can be grouped into two sets: routing management and content management. Figure 1 shows some of the basic building blocks of the software architecture.

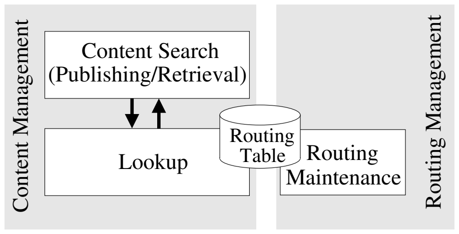

Figure 1: Software architecture of KAD.

Routing management takes care of populating the routing table and maintaining it. The maintenance requires to update the entries – called **contacts** – and to rearrange the contacts according to the distance: A peer stores only a few contacts of peers that are far away in the KAD ID space and increasingly more contacts to peers closer in the KAD ID space. If a contact refers to a peer that is offline, we define the contact as **stale**. In order to face the problem of stale contacts due to churn (departure of peers), KAD uses redundancy, i.e. the routing table stores more than one contact for a given distance. The routing management is responsible also for replying to route requests sent by other nodes during the lookup (Sect. 2.2). Since in this paper we focus on content management, we do not go into the details of the routing procedure (the interested reader is referred to [16]). The only information we use is the probability that a contact contained in the routing table is stale: p_{stale}.

Content management takes care of publishing the information about the objects the peer has, as well as retrieving the objects the peer is looking for. We summarize these two operations with the term **content search**, since they actually use the same procedure (Sect. 2.3). In both cases the peer has a *target* KAD ID (of the objects it wants to publish or it wants to retrieve) that it needs to reach. The KAD ID of an object is obtained by hashing the keywords in its filename. Since the peer routing table does not contain the KAD ID of all peers, the peer needs to build a temporary contact list, called **candidate list** or simply *candidates*, which contains the contacts that are closer to the target. The temporary list building process – called **lookup** – is done iteratively with the help of the other peers. Since the lookup process and the content search process represent the focus of our paper, we explain them in detail in the following sections.

2.2 Lookup

The lookup procedure is responsible for building the candidate list with contacts that are closest to the target KAD ID, i.e. contacts with the longest common prefix to the target. The procedure, along with the main data structures, is summarized in Procedure `Lookup`.

```
Procedure Lookup
Data: $hash(128bit)$: target = hash of the target
Data: $\underline{list}$: candidates = peers to query, ordered by their distance to target
Data: $\underline{list}$: queried = peers queried with route requests
Data: $\underline{list}$: answered = peers that replied to route requests
Data: final $\underline{int}$: $\alpha$ = initial degree of parallelism
Data: final $\underline{int}$: $\beta$ = number of contacts requested
Data: final $\underline{int}$: $t$ = seconds to wait for route request messages
Data: $\underline{timestamp}$: timeout /* timeout for route request messages
    */
Data: $\underline{int}$: lookuptime = 0 /* time the lookup process is running
    */
Data: $\underline{int}$: objectcount = 0 /* number of object references received
    */
Data: $\underline{int}$: contentreplies = 0 /* number of content replies received
    */
Initialization:
    candidates.insert(50 closest peers to target from our routing table);
    send route request(target,$\beta$) to first $\alpha$ candidates;
    queried.insert(first $\alpha$ candidates);
    timeout = now + $t$;

When route response is received do
    timeout = now + $t$;
    answered.insert(sender);
    foreach contact $\in$ response do
        if contact not $\in$ candidates and contact not $\in$ queried then
            candidates.insert(contact);
            if dist(contact,target) $<$ dist(sender,target) then
                /* approaching the target                          */
                if contact is in the $\alpha$ closest contacts to the target then
                    send route request(target,$\beta$) to contact;
                    queried.insert(contact);
```

The source peer first retrieves from its routing table the 50 closest contacts to the destination and stores them in the candidate list. The contacts are ordered by their distance to the target, the closest first. The discover process is done starting from this initial candidate list in an *iterative way*. The source peer sends a request to the first α contacts (by default $\alpha = 3$). The request is called *route request*. The source peer remembers the contacts to which a route request was sent. A route request asks by default for $\beta = 2$ closer contacts contained in the routing tables of the queried peers. A timeout is associated to the lookup process. In case the source peer does not receive any reply, it can remove the stale contacts from the candidate list and it can send out new route requests.

As soon as one route response arrives, the timeout is reset and for each of the β contacts in the response it is checked if the contact has not already been queried and it is not already in the candidate list. A route request is sent if (i) the new contact is closer to the target than the peer that provided that contact, and (ii) it is among the α closest contacts to the target. This implies that in the extreme case for every of the α incoming route responses $\min(\alpha, \beta)$ new route requests are sent out. If the returned contact is not among the α closest known contacts it is stored in the candidate list.

Figure 2 illustrates an example of the lookup process. On the top we show the evolution of the candidate list, where we use the flags 's' and 'r' to record if a route request has been sent or a route response has been received respectively. α is set to 3 and β is set to 2. The initial list is composed of contacts a, b, c and d. The distance in the vertical axes indicates the XOR-distance to the target. At the beginning, the source peer sends a route request to the top α contacts a, b and c. Contact c is stale and will never reply. The first response comes from b and contains β contacts, e and f, that the source peer does not know. The new contacts are inserted in the candidate list: since they are closer to the target than the other candidates, a route request is sent to them. At this point the response of a arrives. The new contacts, g and h, are inserted in the candidate list. Since contact h is not among the top α contacts, no route request is sent to h. After some time, the source peer receives the response of e, but only one of the contacts is inserted in the candidate list, since the other one is already present in the list.

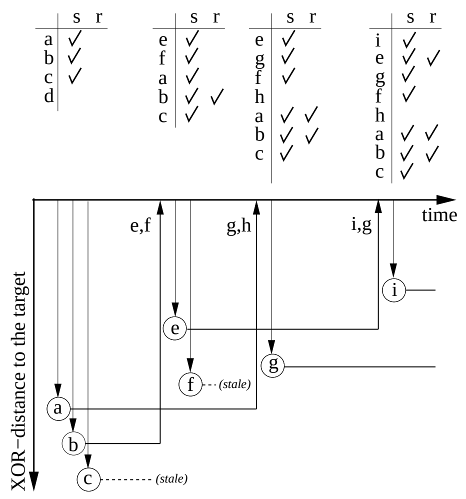

Figure 2: Example of lookup ($\alpha = 3$; $\beta = 2$).

The Procedure `Lookup` terminates when the route responses contain only contacts that are either already present in the candidate list or farther away from the target than the other top α candidates. At this point, no new route request is sent and the list becomes *stable*. The stabilization of the candidate list represents a key point for KAD. In fact, the source peer has to exhaustively search for all the contacts around the target. We show in Sect. 3 how the stabilization influences the performance.

2.3 Content Search

When the candidate list becomes stable, the source peer can start the content search process. The designers of KAD decided to consider a contact *sufficiently* close to the target if it shares with it at least the first 8 bits. The space of KAD IDs that satisfy this constraint is called **tolerance zone**. At the time KAD has been designed probably nobody thought of having such a huge success, with millions of users. However, today the 8 bit tolerance zone is too big, since it contains up to 10,000 users in the evening hours [14].

Each candidate that falls in the tolerance zone can be considered for storing or retrieving a reference. The process is described in Procedure `Content Search`.

In the implementation of KAD, there is no direct communication between the Procedure `Lookup` and the Procedure `Content Search`, i.e. when the candidate list becomes stable, the Procedure `Lookup` does not trigger the Procedure `Content Search`. The stabilization of the candidate list means that in the last t seconds no route responses are received, where t is the *timeout* set to 3 seconds by default. This can happen for two reasons: the closest peers to the target have been found or the queried peers did not reply, i.e. the top α contacts in the candidate list are stale, or overloaded, or the messages were lost.

The solution adopted by KAD to handle these two different situations is a periodic execution of the Procedure `Content Search`: every second the procedure checks if the candidate list has been stable for at least t seconds. In this case, the procedure iterates through the candidate list: a content request is sent if (i) a route request was sent to the contact, (ii) the contact replied with a route response and (iii) the contact belongs to the tolerance zone. The content request contains a '`store reference`' type in case of publishing and a '`search reference`' type in case of content retrieval (line 13). When the procedure iterates through the candidate list and finds a contact that has not been queried, it sends a (single) route request, actually restarting the Procedure `Lookup`. This is useful in case the lookup gets stuck (line 15).

When a content response is received, a counter is incremented. In case of content publishing, the maximum value for this counter is set to 10: in order to face churn each reference is published to 10 different peers that belong to the tolerance zone. In case of content retrieval, the response contains one or more objects with the requested reference and the maximum value for the counter is set to 300, i.e. at maximum 300 objects that contain the reference are accepted.

The main loop is stopped for one of the following three reasons: either the maximum search time is reached (lines 4-5), or there are no more contacts to query (lines 6-7), or enough content response have been received (lines 22-30).

3 Analysis of the Content Search Process

The content management process in KAD is divided into two procedures – `Lookup` and `Content Search`. The latter contains in a single module both content publishing and retrieval. Nevertheless, the aims of the two tasks – publishing and retrieval – are completely different. On the one hand a peer should try to publish the different replicas as close as possible to the target: this requires a candidate list to be stable, a result that can be obtained with large timeouts – note that, as explained above, a stable

Procedure `Content Search` (Publish or Retrieval)

```
Every 1 sec do
    if not stopsearch then
        lookuptime++;
        if lookuptime > 25 then
            stopsearch ← true;
        if candidates is empty then
            stopsearch ← true;
        if timeout ≤ now then
            /* candidate list is considered stable          */
            for i ← 0 to candidates.size do
                contact = candidates.get(i);
                if contact ∈ queried then
                    if contact ∈ answered and contact in tolerance zone around
                    target then
                        /* the timeout triggers the actual
                           content search.                      */
                        send content request(TYPE, target) to contact;
                        candidate.remove(contact);
                else
                    /* the timeout triggers a new route request.
                       */
                    send route request(target,$\beta$) to contact;
                    queried.insert(contact);
                    return;

When content response is received do
    contentreplies++;
    if peer is publishing then
        if contentreplies > 10 then
            stopsearch ← true;
            return;
    else
        foreach object ∈ response do
            objectcount++;
        if objectcount > 300 then
            stopsearch ← true;

```

candidate list does not necessarily mean that the contacts are close to the target. On the other hand, a peer should look for the content as soon as it is sufficiently close to the target, i.e. when it enters in the tolerance zone: in this case a stable candidate list is not necessary.

In this section we analyze the impact on the performance of the content management approach adopted by KAD. The main performance metric for the content search process is the *overall lookup latency*, i.e. how long it takes to reach the target and find the content. The delay is mainly influenced by the following parameters:

p_{stale}	probability that a contact is stale;
d	round trip delay between two peers;
h	number of iterations (*hops*) necessary to reach the target;
α	number of route requests sent initially;
β	number of closer contacts asked for by a route request;
t	time waited for route response messages.

While p_{stale}, d and h cannot be controlled by the content management process, α, β and t do depend on the implementation.

3.1 Qualitative Analysis of the Latency

Lookup Latency. For the analysis of the delay, let $F_{\text{RTT}}(d)$ be the cumulative distribution function (CDF) of the round trip delay for the single hop (see for instance the empirical CDF, found with measurements, shown in Fig. 5). At the first iteration (hop and iteration are used interchangeably) α route requests are sent. We assume that the probability that all the α contacts are stale, $p_{\text{stale}}^{\alpha}$, is negligible.

Among the initial α messages, only $\alpha(1 - p_{\text{stale}})$ replies are received. At each response, $\gamma = \min(\alpha, \beta)$ messages can be possibly sent out. The maximum number of route requests at the second hop, $\rho_{\text{2, max}}$, is then $\alpha(1-p_{\text{stale}})\gamma$. In the following hop, only a fraction of $(1 - p_{\text{stale}})$ of contacts reply and each response can trigger γ new requests. The maximum number of messages at hop i, $\rho_{\text{i, max}}$, is

$$\rho_{\text{i, max}} = \alpha[(1 - p_{\text{stale}})\gamma]^{i} \tag{1}$$

and the cumulative maximum number of messages up to hop h, $\rho_{\text{h, max}}$, is

$$\rho_{\text{h, max}} = \alpha \sum_{i=0}^{h-1} [(1 - p_{\text{stale}})\gamma]^{i}. \tag{2}$$

In practice, some contacts in the replies are already known or they are not inserted in the top α positions of the candidate list, so the actual total number of route requests sent up to hop h will be $\overline{\rho}_h \leq \rho_{\text{h, max}}$. Figure 3 shows $\rho_{\text{h, max}}$ and $\overline{\rho}_h$ for two settings for the parameters α and β. The value of p_{stale} used to compute $\rho_{\text{h, max}}$ and the value of $\overline{\rho}_h$ have been found by measurements as will be explained in Sect. 4. We consider up to three hops, since, as we will see in Sect. 4, more than 90% of the lookups reach the target in less than four hops. The actual number of messages sent is close to the maximum we computed in case of default values for α and β (3 and 2 respectively). If we increase α and β both to 4, we receive more duplicates or not interesting contacts, thus the actual total number of route request messages is far less than the maximum.

The candidate list stabilizes only *after the last response* is received, thus the stabilization time corresponds to the maximum round trip delay over all the route requests that were sent. To simplify of the analysis, we assume that all messages are sent at the beginning[1]. The CDF of the lookup delay can be found considering that the maximum

[1]This is an unrealistic assumption that provide optimistic results; for our purpose, this coarse analysis is sufficient to understand the impact of the parameters.

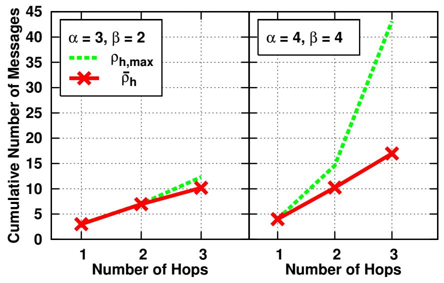

Figure 3: Number of route request messages sent during lookup.

of two random variables, which corresponds to the product of their CDFs (see [3], Eq. 2.6), thus we obtain

$$F_{\text{lookup}}(d) = F_{\text{RTT}}(d)^{\overline{\rho}_h} \ . \tag{3}$$

If we increase α, or β (or both), $\overline{\rho}_h$ increases, i.e. more messages are sent. In fact with a higher degree of parallelism, the number of message on the fly increases. With a higher value of β more new contacts may come back and correspondingly the probability that these contacts fall in the top of the list increases. The overall effect is that the client sent more messages. The drawback of this approach lies in the time for the stabilization of the candidate list. In fact the list becomes stable when contacts received in the answers are not new. If many messages are sent, the probability to receive a new contact increases; this new contact can be contained in one of the last message sent. In general, the list become stable when **all** the responses are received.

This is the contrary of what one would expect – and require during the design phase –, namely that sending more messages should increase the chances to reach the target faster. The key point is that the client has to wait for all the responses, non only for the fastest ones. With the current scheme it is not possible to reduce the lookup latency by increasing the parameters α and β.

Content Retrieval Latency. Once the candidate list is stable, the lookup process terminates. At this point the content retrieval process waits for t seconds (timeout) before starting to send the content requests. This adds to the overall latency t seconds, plus a random delay uniformly distributed between zero and one second, due to the periodic execution of the Procedure `Content Search`. Moreover there is an additional round trip delay due to the content request message.

Overall Lookup Latency. In summary, the overall latency of the content retrieval process is composed by different terms. Let $f_{\text{lookup}}(d)$ be the probability density function (PDF) of the lookup latency, i.e. the derivative of $F_{\text{lookup}}(d)$ found in Eq. (3). The PDF of the overall lookup delay, $f_{\text{overall}}(d)$ can be found by considering that the sum of two random variables corresponds to the convolution of their PDFs, denoted with the symbol '$*$'. We obtain

$$f_{\text{overall}}(d) = f_{\text{lookup}} * \delta_t * \text{Unif}_{(0,1)}(d) \tag{4}$$

where δ_t is the Dirac's delta function translated in t (the timeout value) and $\text{Unif}_{(0,1)}$ is the PDF of a random variable uniformly distributed between 0 and 1. For simplicity we do not consider the additional round trip delay due to the content request message since it can be correlated with the lookup delay. The CDF of the overall lookup delay, $F_{\text{overall}}(d)$, can be found by integrating Eq. (4). Figure 4 shows the CDFs of the overall lookup latency for different values of α and β. The input CDF of the round trip delay, $F_{\text{RTT}}(d)$, and the value of $\overline{\rho}_h$ have been obtained by measurements as we will explain in Sect. 4. As already observed, by increasing the design parameters α and β, the overall lookup latency increases. The fact that the lookup process and the content search are decoupled results in an overall delay that is strongly dependent on the value of the timeout t.

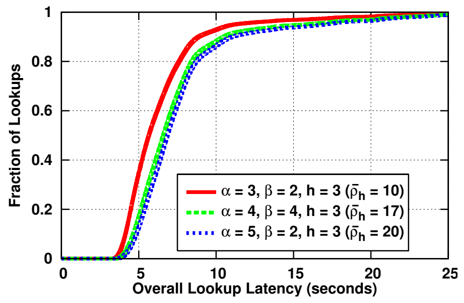

Figure 4: Overall Lookup Latency $F_{\text{overall}}(d)$: qualitative analysis ($t = 3$).

4 Evaluation

In this section we measure the performance of the content management in KAD. We first evaluate the external factors that we cannot influence: p_{stale}, the empirical CDF of the round trip delay $F_{\text{RTT}}(d)$ and the empirical CDF of the number of hops h. Then we study the current implementation and the impact of of the design parameters α, β, and t.

4.1 Measurement Tool and Methodology

For our measurements we have instrumented version 2.1.3 of aMule[1] to log all the messages related to content management: route requests and route response, as well as content search and content response. Given a keyword, the client determines the target KAD ID and starts the Procedure `Lookup` and the Procedure `Content Search`. For each message, we register the time stamp, and for lookup responses we register the contacts returned, so that we can evaluate the evolution of the candidate list.

We have extracted 1251 keywords from movie titles found on IMDB [17] and we use them as input for the instrumented client. The keywords are chosen such that the hashes of these keywords are uniformly distributed over the hash space. As explained in Sect. 2.3, the content retrieval process stops at the latest after 25 seconds. This means that we can launch the lookup for a keyword every half a minute. For a given set of values for α, β and t one experiment where we lookup all 1251 keywords takes about 10 hours[2].

The metrics derived from the collected data are:

p_{stale}: the probability of stale contacts, found as ratio between the number of requests sent and responses received;

CDF of d: empirical cumulative distribution function of the round trip delay for a single message;

CDF of h: empirical cumulative distribution function of the number of hops necessary to reach the target (the first peer that replied with the content);

CDF of the Overall Lookup Latency: empirical cumulative distribution function of the delay between the first *route* request sent and the first *content* response received;

Overhead: The number of route request messages sent during a lookup process.

The initial number of route requests launched is set to $\alpha = 3$; the number of contacts contained in the route response is set to $\beta = 2$. The timeout is equal to $t = 3$ seconds. These are the default values in aMule [1]; we perform a set of experiments by changing these values and we evaluate the impact of them on the overall lookup latency and on the overhead.

4.2 Basic Characteristics

Staleness (p_{stale}). The first parameter we analyze is p_{stale}, the probability that a contact is stale. We perform the same set of experiments with two different access networks and we have found a value of p_{stale} approximately equal to 0.32. This value has a strong impact on the performance: one third of the contacts are stale, so a lookup process with low α may get stuck with high probability. With the default value $\alpha = 3$ this happens with probability $p_{\text{stale}}^{\alpha} = 0.03$. We will see that this value is partly responsible for the tail of the empirical CDF of the overall lookup latency (see Fig. 8).

Round Trip Latency (d). The other interesting metric that is independent from the client settings is the round trip delay of messages. Figure 5 shows the results of our measurements obtained using two different access networks.

Almost 80% of the responses are received within 700 msec after the request was sent. However, the distribution has a significant tail, which impacts, as we will see, the overall lookup latency.

Number of Hops (h). Table 1 shows the empirical probability mass function of the number of iterations necessary to reach the target. It is interesting to note that at maximum 4 hops are necessary, and in more than 90% of the cases 3 hops are sufficient. This means that, since the KAD network has more than one million concurrent users [14], the routing tables are very detailed (about 1,000 contacts).

[2]We provide the modified files of the aMule client as well as the list of keywords we used at `http://www.eurecom.fr/~btroup/kadlookup/`.

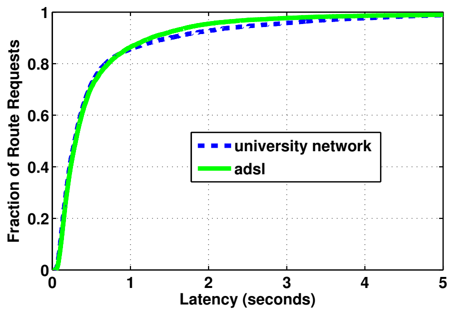

Figure 5: Empirical CDF of the round trip delay for route requests.

Table 1: Number of hops needed per lookup and the average number of bits gained per hop.

hop *i*	% of loookups terminating after *i* hops	bits gained at hop *i*
1	1	6.13
2	55	6.02
3	37	5.24
4	7	2.30

Once the content is found, we can evaluate the number of bits in common between the KAD ID of the keyword we searched for and the KAD ID of the contact that replied with the content. This helps in understanding how much the content is spread around the target. Figure 6 shows the empirical CDF of the number of bits in common between the replying peer and the content hash. The wide support of the empirical CDF indicates that many keywords can be far from the corresponding target. In Sect. 4.4 we will use this observation in order to study the impact of the timeout.

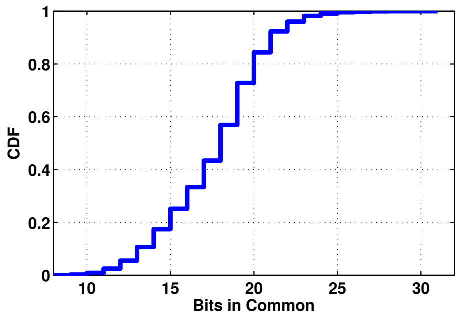

Figure 6: Empirical CDF of the bits in common between the peers replying to the search requests with the desired content and the hash of the content.

Looking deeper into the spread of the content, it is interesting to evaluate its correlation with the popularity of the keywords. In fact, if a keywork is popular (the files that contain that keywork are highly replicated), many peers will try to publish it; this means that many peers build a temporary candidate list and publish using such candidates, and the probability that these list contain peers that are not very close to the target (even if they are inside the tolerance zone) grows as more and more publishing attempts are done. To check the influence of the keyword popularity on the spread of its references, we perform an exhaustive search with our instrumented client: first the entire zone around the searched keyword in the hash space is crawled to learn about all peers; second all these peers are queried for the desired content [13]. As shown in Fig. 7, rare keywords (as “dreirad” or “fahrrad”) have more bits in common with the peers that host them, compared to popular keywords (as “the”, “french”, or “german”).

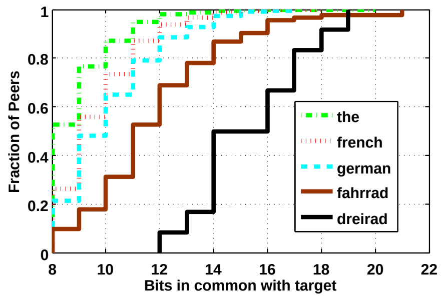

Figure 7: The CDF of the bits in common between the peers hosting a content and the hash of the content.

The final empirical CDF shown in Fig. 6 can be considered as the composition of the single keyword CDFs as shown in Fig. 7, where the contribution of each single keyword CDF depends on the popularity of the keyword.

4.3 Impact of Different Degrees of Parallelism α

In Figure 8 we show the empirical CDF of the overall lookup latency when the parameter α varies from 1 to 7 (its default value is 3). We note a significant difference between the case $\alpha = 1$ and the cases $\alpha \geq 2$, which is due to the high value of p_{stale}. In case of $\alpha = 1$, at each hop only one message is sent; if the contact is stale and the message is lost, the process has to wait for the timeout to expire. This has a strong impact on the overall lookup latency.

With $\alpha = 2$, the probability that the top 2 contacts are all stale decreases significantly. For instance, with $p_{\text{stale}} = 0.32$, the probability that at the first hop both contacts are stale is $p^2_{\text{stale}} = 0.1$. Therefore, the impact of the timeout due to stale contacts on the overall lookup latency reduces, and becomes negligible for $\alpha \geq 3$.

With $\alpha \geq 3$, the different empirical CDFs seems to overlap. If we look in detail at the median (Table 2, with $\beta = 2$ and $t = 3$), we see that, as α increases, the median of the overall lookup latency increases. This result was predicted by our qualitative analysis in Sect. 3.1 (c.f. Fig. 4). The higher α, the more messages the source peer sends ($\overline{\rho}_h$), the longer it takes for the candidate list to stabilize, since it is influenced by the delay of the last received response.

As also shown in the qualitative analysis in Sect. 3.1, the support of the empirical CDF starts at $d = 4$ seconds. In the best case, in fact, the candidate list stabilizes

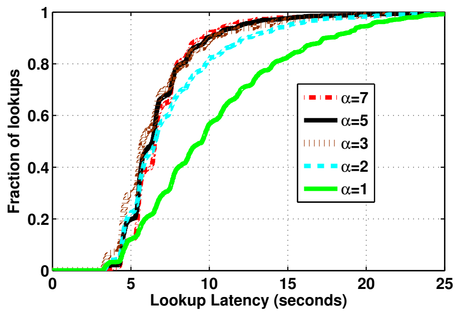

Figure 8: Empirical CDF of the overall lookup latencies as a function of the degree of parallelism α ($\beta = 2; t = 3$).

after approximately 100 milliseconds (each hop takes at least 40 msec, and the mean number of hops is 2.5). Once the list is stable, the source peer has to wait for the timeout ($t = 3$ seconds), and for the periodic execution of the Procedure `Content Search` (in average, 500 msec). If we consider the application level processing delay, we obtain almost 4 seconds.

As regards the overhead, Table 2 shows the average number of route requests sent for different values of α (left hand side of the table, with $\beta = 2$ and $t = 3$). The number of messages sent increases linearly with increasing α.

Table 2: The overall lookup latency and the number of route requests sent per lookup depending on α for different configurations.

α	$\beta = 2; t = 3$		$\beta = 2; t = 0.5$	
	average # of messages $\overline{\rho}_h$	median lookup latency	average # of messages $\overline{\rho}_h$	median lookup latency
1	8.5	9.5	10.4	5.6
2	11.5	6.6	12.8	2.4
3	**13.7**	**5.8**	**15.2**	**2.3**
4	16.9	6.1	18.0	2.3
5	20.0	6.4	20.6	2.2
6	22.9	6.5	24.0	2.3
7	26.5	6.5	27.7	1.8
8	30.0	6.6	30.4	1.6
9	32.9	6.6	34.0	1.5
10	36.7	6.6	36.8	2.2

4.4 Impact of the Timeout t

The default timeout t in aMule is set to three seconds. This implies that the candidate list must be stable for three seconds before the content can be requested. As we showed in Sect. 4.2 (Fig. 6), the contacts that hold the content may be spread around the target. This means that we could start asking for the content as soon as the lookup finds a candidate in the tolerance zone, without waiting for the candidate list stabilization.

One possible way to obtain the above result is to decrease the time the Procedure `Content Search` has to wait before starting iterating through the candidate list, i.e. we can decrease the timeout t.

As for α, also t can be changed locally at our instrumented client, without need to update all participants in the network. In Figure 9 we show the empirical CDFs of the overall lookup latency for different timeouts for the route request messages.

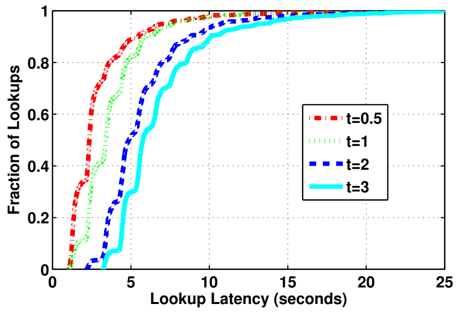

Figure 9: Empirical CDF of the overall lookup latencies as a function of the route request timeout t ($\alpha = 3; \beta = 2$).

As the timeout decreases, its influence on the overall latency becomes less significant: reducing the timeout from the default value of 3 seconds to 0.5 seconds decreases the median lookup latency by 60%, from 5.8 to 2.3 seconds. Note that further reducing the timeout would have no effect, since the periodic execution of the Procedure `Content Search` is set to 1 second.

Similar results are obtained using a different access network, a common ADSL line (see Fig. 10). This is as expected, since the CDFs of the round trip delay are almost the same for the ADSL network or the university access (Fig. 5).

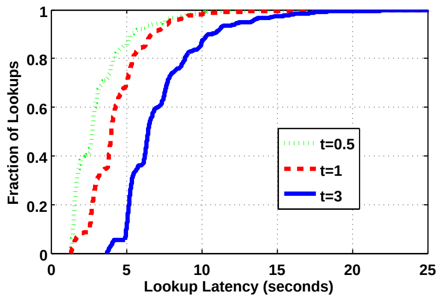

Figure 10: Empirical CDF of the overall lookup latencies depending on the route request timeouts for an ADSL client. ($\alpha = 3; \beta = 2; t = *$)

In Table 2 we show the overhead for a timeout t set to 0.5 seconds (right hand side of the table). If we compare the default case $\alpha = 3$, $t = 3$ with the case $\alpha = 3$, $t = 0.5$, we see that the overhead is slightly increased: this is mainly due to the fact that the timeout is also used to trigger new route requests, and, if the responses to the initial α requests arrive later than $t = 0.5$ seconds, new requests are sent out.

Table 3: The overall lookup latencies and the number of messages sent per lookup depending on t for different configurations.

t	$\alpha = 3; \beta = 2$		$\alpha = 1; \beta = 2$	
	average # of messages	median lookup latency	average # of messages	median lookup latency
0.5	16.2	2.3	11.4	5.5
1	15.6	3.4	10.9	5.6
2	15.2	4.9	10.2	7.8
3	14.7	5.8	9.5	9.5
4	14.7	7.4	9.1	10.9
5	14.4	8.2	8.7	12.7

In Table 3 we show the overhead and the lookup latency for different values of the timeout t. As already shown in Figs. 9 and 10, the lookup latency (in this case the median) decreases significantly for decreasing t. The interesting result is given by the average number of messages sent: in this case we have only a slight increase of the overheads. This means that a new design, where the role of the timeout is revisited, may improve the latency without affecting the overheads.

4.5 Impact of the Number of Contact Asked For

Once observed the gain that can be obtained by eliminating the effect of the timeout, we want to evaluate the impact of the parameters α and β on the overall lookup delay. Recall that β is the number of closer contacts that are asked for by a route request message. Unfortunately, this parameter cannot be chosen freely in the source code, but can be only set to 2, 4, or 11. We performed measurements for $\beta = 4$ and varying α.

Figure 11 shows the results we obtained with different settings. The more messages we send, the more the overall lookup latency is reduced. This comes at a cost of increased overhead. For instance, for $\alpha = 5$ and $\beta = 4$ the mean number of messages is equal to 29. By increasing further the values of the parameters, we are not able to notice a significant improvement in the empirical CDF, since the periodic execution of the Procedure `Content Search` determines the overall lookup delay.

5 Improving the Content Lookup

The evaluation of the impact of the timeout t on the overall lookup latency suggests that a different approach to the content management process would bring a substantial gain.

The idea is to differentiate the software architecture according to the two different aims – publishing or retrieval – as shown in Fig. 12. In the following, we describe the different approaches.

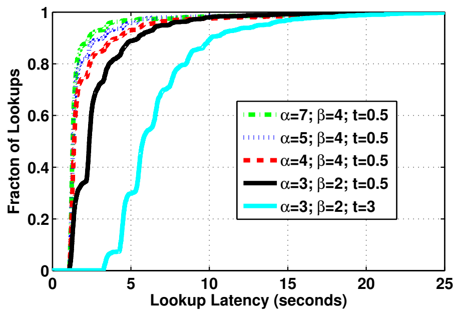

Figure 11: Empirical CDF of the overall lookup latencies varying α and β.

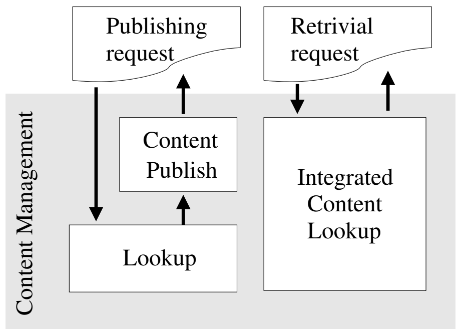

Figure 12: Improved software architecture of KAD.

5.1 Improved Content Publishing

The primary aim of the content publishing is to find the nodes as close as possible to the target: this process requires time because the peer needs to exhaustively search for the contacts. Only when the contact list is stable, the peer can publish the content. This is similar in spirit to the original design of the content lookup process. Nevertheless, in the original design the timeout has two different roles: a timeout occurs when (i) candidates do not reply or when (ii) candidates reply with contacts that are not closer than the other candidates. In the first case the lookup procedure may be stuck waiting for replies, with an increasing in the delay that is not useful for the publishing process. In the second case, contacts that are already known is a sign of a stable candidate list. These two meaning should be decoupled.

Each candidate should have a flag that indicate its status: NQ (not queried), Q (queried), R (reply received) and T (timeout). The candidate must be necessarily in one of this status. If it is a new contact, it is inserted as NQ; if a query is sent to that contact, its status is updated to Q and a timeout is associated to the contact. If a reply is received, the status is updated to R, otherwise, after the expiration of the timeout, the status is updated to T.

At this point the lookup phase checks periodically the status of the candidates. The list is considered stable if, starting from the the top of the list and not considering the contacts in status T, there are at least 10 consecutive contacts in status R. In order to avoid too long delays, a global timeout for the lookup process is maintained, after which the candidate list is sent to the content publishing module (see Fig. 12) in any case.

When the timeout expires, the lookup process can trigger new queries to contacts in the list that are in status NQ. The role of the timeout is then only related to stale contacts. The stabilization process results in a list that contains with high probability the best contacts (i.e. closer to the target) the node can reach.

5.2 Improved Content Retrieval

For the content retrieval, the lookup and the retrieval should be strictly coupled: as soon as the lookup finds a candidate in the tolerance zone, a content request should be sent. We call this approach *Integrated Content Lookup (ICL)*. In Sec. 4.4 we have shown how to obtain a similar objective with a simple *hack* of the code: by decreasing the timeout t we let the content search process iterate though the candidate list more frequently. The results we have obtained are pessimistic, since they include the delay due to the periodic execution of the Procedure `Content Search`. In this section we propose a model that shows the qualitative performance in terms of the overall lookup latency of ICL scheme.

We assume that the probability that all the initial α contacts are stale, $p_{\text{stale}}^{\alpha}$, is negligible. Among the initial α messages, only $\alpha(1 - p_{\text{stale}})$ replies are received. The process continues to the next hop using the contacts contained in the first reply. Thus, the delay of the first hop is the *minimum* delay among the replies. It is simple to show that the corresponding CDF for the first hop is equal to (see [3], Eq. 2.8)

$$F_{\text{1, ICL}}(d) = 1 - [1 - F_{\text{RTT}}(d)]^{\alpha(1-p_{\text{stale}})} . \tag{5}$$

At this point we neglect the contacts contained in the route responses that come after the first, and concentrate only on the β contacts we received. This simplification ignores possible better contacts contained in the responses of the first hop that are received later: in this sense the analysis is conservative. We assume that the contacts contained in the first response are placed in the top of the candidate list (they are closer to the target than the candidates already present). In the second hop the process sends $\gamma = \min(\alpha, \beta)$ new route requests. Among them, only $\gamma(1 - p_{\text{stale}})$ replies are received. The CDF of the delay for the second hop is

$$F_{\text{2, ICL}}(d) = 1 - [1 - F_{\text{RTT}}(d)]^{\gamma(1-p_{\text{stale}})} . \tag{6}$$

For the following hops, we have the same behavior as for the second one. When a contact replies, the Integrated Content Lookup process checks if it falls in the tolerance zone and immediately send a route request. Thus, the overall lookup latency is given by the sum of the delay of the single hops. Let $f_{\text{i, ICL}}(d)$ be the PDF of the delay for a single hop i, i.e., the derivative of $F_{\text{i, ICL}}(d)$ of Eqs.(5) and (6). The PDF of the overall lookup latency, $f_{\text{ICL}}(d)$, is then

$$f_{\text{ICL}}(d) = f_{\text{1, ICL}} * f_{\text{2, ICL}} * \ldots * f_{\text{h, ICL}}(d) \tag{7}$$

where the convolution is done for all the h hops. The CDF of the overall lookup delay can be found by integrating Eq. (7). Figure 13 shows the CDFs of the overall lookup latency for different values of the parameters α and β, with a number of hops

$h = 3$. We consider the input CDF of the round trip delay, $F_{\text{RTT}}(d)$, shown in Sect. 4. The design parameters α and β now have a significant impact on the overall lookup latency, at a cost of increased overhead. This qualitative analysis yields the same results as shown in the experimental evaluation, where we studied different settings for the parameters α, β and t (Fig. 11). It is interesting to note that the CDF has a similar tail as we found with measurements: this means that the tail of the input CDF $F_{\text{RTT}}(d)$ has a strong impact, even for large α and β.

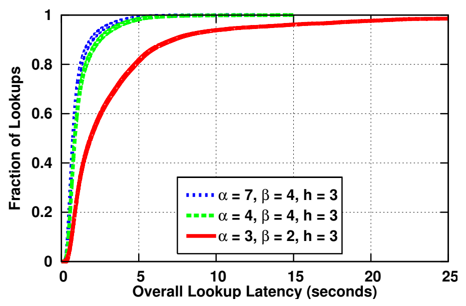

Figure 13: CDF of the Overall Lookup Latency, $F_{\text{ICL}}(d)$: qualitative analysis.

5.3 Additional Improvements

In the analysis so far we have considered a set of parameters – α, β and t – with fixed values. We have shown how these values can influence the overall performance considering the current adoption and characteristics of the user behavior. It is clear that fixed parameters may work in some scenarios, while they may give poor performances in others. For instance, in case of high churn (e.g. 50% of the contacts are stale), asking only for $\beta = 2$ contacts could result in high delay due to timeouts (many stale contacts are inserted in the candidate list and can be queried).

The publishing or retrieval scheme can be further improved if we let the design parameters to change, i.e. if we make them adaptive to the different scenarios. The input parameters would be the parameters that we cannot control: the probability that entries are stale (p_{stale}), the round trip delay of the messages (d), and the number of hops h necessary to reach the target. The client can continuously monitor these parameters and accordingly adjust the other design parameters – α, β and t. For instance, in contexts with a low churn, the probability that entries are stale (p_{stale}) reduces, and thus it is not necessary to have a large degree of parallelism in the sent requests. In this case we may choose α and β as functions of p_{stale}, rather than simply taking fixed values. The same applies for timeout t, i.e. we may choose t as a function of d, the (estimated) round trip delay.

These changes would make the design of the client more flexible to the future changes of its use. For instance, there are variants of the eMule client that works only for a specific ISP [6]: the majority of the clients have fiber connectivity and the ISP offer mainly flat rate, thus the user behavior for this ISP is different from a typical user behavior worldwide; in particular both the churn and the delay are reduced. In this context, the overheads can be reduced, yet maintaining the same performances.

6 Related Work

Stutzbach and Rejaie [16] did a detailed analysis of the routing tables and the lookup process in KAD and provide latency measurements for varying α. However the two remaining parameter β and t are not mentioned.

Falkner et al. [5] analyzed the implementation of KAD in *Azureus* and measured a median of the overall lookup latency of 127 seconds (more than 2 minutes!). The authors explain these huge values by the fact that the routing tables of the Azureus clients contain many stale contacts and thus timeouts occur frequently during the lookup. We think that such an high response time is highly incredible, if the system really suffered from such a high response time it would be unusable. The measurement we did on KAD in aMule using the default configuration showed a median overall lookup time of 5.8 seconds.

Li et al. [8] describe the lookup process in KAD identifying the parameters α and β. Using the one hop latency data obtained with the King method [7], a simulation of the overall lookup latency was performed with p2psim. This simulation found an average overall lookup latency of 250 milliseconds.

7 Conclusions

In this paper we study the content management process of KAD as implemented in aMule. We identified the key parameters, developed an analytical model for the lookup latency and the lookup overhead that was validated by measurements on a real client.

The measurements allowed us to:

- Characterize the external factors that influence the performance – such as the probability that entries in the routing tables are stale, or the round trip delay of messages
- Evaluate the influence of the design parameters – such as the number of requests sent initially or the timeout – on the overall performance.

We saw that the current implementation exhibits a poor lookup performance that can be significantly reduced by coupling the lookup procedure and the the content retrieval process, while keeping the overhead the same.

Moreover, we propose to dynamically adapt all the design parameters, the number of parallel requests, the number of closer contacts asked for, and the timeout, to the measured or estimated round trip delay.

Acknowledgments

The authors would like to thank Antonio Trifilo and Emanuele Leomanni for their work on the aMule code.

References

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

[2] Azureus. `http://azureus.sourceforge.net/`.

[3] E. Castillo. *Extreme Value Theory in Engineering*. Academic Press, Inc., 1988.

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

[5] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. Profiling a Million User DHT. In *Proc. of IMC*, 2007.

[6] Fastweb (Internet Provider). `http://www.fastweb.it/`.

[7] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proc. of Internet Measurement Workshop*, 2002.

[8] J. Li, J. Stribling, R. Morris, M. Kaashoek, and T. Gil. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. In *Proc. of INFOCOM*, 2005.

[9] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer informatiion system based on the XOR metric. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

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

[11] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of SIGCOMM*, 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale Peer-to-peer systems. In *Proc. of Middleware*, Heidelberg, Germany, 2001.

[13] M. Steiner, W. Effelsberg, T. En-Najjary, and E. W. Biersack. Load Reduction in the KAD Peer-to-Peer System. In *Fifth International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2007.

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

[15] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-to-peer lookup service for Internet applications. In *Proc. of SIGCOMM*, 2001.

[16] D. Stutzbach and R. Rejaie. Improving lookup performance over a widely-deployed DHT. In *Proc. of INFOCOM*, 2006.

[17] The Internet Movie Database. `http://www.imdb.com/`.