# Characterization and Management of Popular Content in KAD

Damiano Carra, Moritz Steiner, Pietro Michiardi, Ernst Biersack,
Wolfgang Effelsberg and Taoufik En-Najjary

**Abstract**—The endeavor of this work is to study the impact of content popularity in a large-scale Peer-to-Peer network, namely KAD. Armed with the insights gained from an extensive measurement campaign, we pinpoint several deficiencies of the present KAD design in handling popular content, and provide a series of solutions to address such shortcomings. Among them, we design and evaluate an adaptive load balancing mechanism. Our mechanism is backward compatible with KAD, as it only modifies its inner algorithms, and presents several desirable properties: *(i)* it drives the process that selects the number and location of peers responsible to store references to objects, based on their popularity; *(ii)* it solves problems related to saturated peers, that would otherwise entail a significant drop in the diversity of references to objects, and *(iii)* if coupled with an enhanced content search procedure, it allows a more fair and efficient usage of peer resources, at a reasonable cost. Our evaluation uses a trace-driven simulator that features realistic peer churn and a precise implementation of the inner components of KAD.

## 1 Introduction

As demonstrated by a wide range of measurement campains, the amount of traffic generated by peer-to-peer (P2P) applications represents a significant portion of all Internet traffic [1], [2]. KAD-based P2P systems have become very popular: KAD is a Kademlia-based P2P routing system. Kademlia [3] is a distributed hash table (DHT) that is implemented in several popular P2P applications, such as Overnet [4], eMule [5] and aMule [6], which involve several millions of users worldwide [7]. Thus it is important to understand whether the inner components of KAD, namely the mechanisms used to publish and search for content, are well designed and whether they can be improved to obtain performance and efficiency gains.

The design of large scale distributed systems poses many challenges due to the heterogeneity of its components. Many systems based on DHTs have been proposed to manage such heterogeneity, primarily focusing on node churn. The dynamic nature of arrivals and departures of peers, and the consequent heterogeneous session times, represents one of the main, and better studied, characteristics of P2P networks.

Despite the vast amount of work on DHTs, little has been said about the heterogeneity in *content popularity*. When dealing with content popularity, we need to consider both objects and references – in a DHT network, to simplify the search based on keywords, nodes store not only objects, but also the references to them. While object popularity derives from how often an object is replicated, reference popularity arises mainly from two reasons. Either the object (where the reference points to) is popular, or the reference is formed by common keywords, such as "the," "mp3", or "dvd," that can be found in different object titles.

In this work we study how KAD copes with object and reference popularity. To this aim, we perform a set of measurements campaigns. While the solution to reference popularity due to common keywords is straightforward, handling heterogeneous object popularity represents a major challenge. The main problem is to tailor the amount of load each peer must support in an adaptive way. Currently proposed solutions usually consider a *statically* pre-set number of peers to use for load balancing. Instead, we realize an *adaptive* load balancing mechanism for KAD-based systems.

The main contributions of our work can be summarized as follows:

- We designed and implemented two measurement tools, an instrumented aMule client and a content spy called *Mistral*, which are able to provide foundamental insights that lead to a better understanding of how KAD works;
- With these tools, we establish an extensive measurement campaign to characterize content popularity and the traffic associated to content publishing and searching. The results show that content publishing generates ten times more messages than content searching; in addition, publish messages are, on average, ten times larger than search messages. We have also studied how KAD manages popular

- *D. Carra is with the Computer Science Dep., University of Verona, Italy. E-mail: damiano.carra@univr.it*
- *M. Steiner is with Bell Labs, Alcatel-Lucent, USA. E-mail: moritz@bell-labs.com*
- *E. Biersack and P. Michiardi are with EURECOM, Sophia Antipolis, France. E-mail: erbi@eurecom.fr, pietro.michiardi@eurecom.fr*
- *W. Effelsberg is with the Computer Science Dep., University of Mannheim, Germany. E-mail: effelsberg@informatik.uni-mannheim.de*
- *T. En-Najjary is with Orange Labs, France Telecom, France. E-mail: taoufik.ennajjary@francetelecom.com*

content. Our results indicate that a large fraction of references to popular objects are lost due to peer saturation. Moreover, our measurements identify the KAD lookup procedure as one of the main culprits of the load imbalance that occurs when references are placed and retrieved;
- Our measurements show that most of the keywords that compose the reference names are meaningless stopwords, which constitute a substantial overhead. As such, in this work we study means to effectively reduce the publishing overhead without reducing the retrieval success rate of the KAD system;
- For popular objects, we design a load balancing scheme which adapts the number of peers used for storing object references to their popularity. The main constraint we consider in our design is to work exclusively on algorithmic changes to KAD, without modifying the underlying protocol by introducing new messages;
- We improve the content searching procedure of KAD to better exploit reference replication. Our goal here is to decrease the burden imposed on few peers by the current KAD implementation and spread the load due to content search on reference replicas;
- We evaluate our proposed schemes (load balancing and search) with a trace-driven simulator which is able to reproduce realistic peer arrivals and departures; our results show that our load balancing scheme is effective in distributing the load among peers in the key space, and the searching procedure is able to find objects referenced by a large number of peers, with low penalty in terms of content search overhead.

We note that, while the schemes presented in this paper are specific to KAD, the main ideas underlying the adaptivity and the exploitation of the available replicas can be generalized and used in other systems as well.

The remainder of the paper in organized as follows. In Sec. 2 we provide some background on KAD, on the content management, and we discuss the related work. In Sec. 3 we introduce our measurement tools, including *Mistral*, our KAD content spy. In Sec. 4 we provide a set of measurement results that give us insights on the current implementation and performance of KAD in case of popular content. The weaknesses highlighted in this section will drive us in the design of the solutions for helping KAD to manage popular content which we present in Sec. 5. We evaluate the proposed load balancing scheme in Sec. 6, and we conclude our paper in Sec. 7.

## 2 Background and Related Work

### 2.1 The Kademlia DHT System

KAD is a DHT protocol based on the Kademlia framework [3]. Peers and objects in KAD have an unique identifier, referred to as KAD ID, which is 128 bit long. The KAD IDs are randomly assigned to peers using a cryptographic hash function. The distance between two entities – peers, objects – is defined through the bitwise XOR of their KAD IDs.

The basic operations performed by each node can be grouped into two sets: routing management and content management. Routing management takes care of populating and maintaining the routing table. The maintenance requires to update the entries – called **contacts** – and to rearrange the contacts accordingly. 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 it as **stale**. The routing management is responsible also for replying to route requests sent by other nodes during the lookup phase (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 [7].

Content management takes care of publishing the **references** to the objects a peer has, as well as retrieving the references to the objects the peer is looking for. KAD implements a two-level publishing scheme; a reference to an object comprises a **source** and $W$ **keywords**:
- The source, whose KAD ID is obtained by hashing the content of the object, contains information about the object and the pointer to the publishing node;
- Keywords, whose KAD IDs are obtained by hashing the individual keywords of the object name, contain (some) information about the object and the pointer to the source.

In the following, we will refer to source and keywords considering the corresponding KAD IDs. We call **publishing node** the node that owns an object and **host nodes** the nodes that have a reference to that object. When a node wants to look for an object, it first searches for the keywords and does a lookup to obtain all the pointers to different sources that contain these keywords. It then selects the source it is interested in, looks up that source to obtain the information necessary to reach the publishing node.

Since references are stored on nodes that can disappear at any point in time, the publishing node publishes *multiple copies* (the default value is set to 10) of each reference – source and keywords. An **expiration time** is associated to each reference, after which the information on the host node is removed: for a source and for a keyword the expiration times are set to 5 and 24 hours, respectively.

### 2.2 Content Management

Content management procedures take care of publishing and searching processes, which leverage on a common function called **Lookup**. Given a *target* KAD ID, the Lookup procedure is responsible for building a temporary contact list, called **candidate list**, which contains the contacts that are closer to the target. KAD creates a thread for each keyword and source, so that the lookup is done in parallel for the different target KAD IDs. The

list building process is done iteratively with the help of different peers. Here we summarize the main steps of the Lookup procedure: for a detailed explanation, we refer the interested reader to [8][9].

**Initialization:** The (publishing or searching) peer first retrieves from its routing table the 50 closest contacts to the destination, and stores them in the candidate list. The contacts are sorted by their distance, the closest one first. The peer sends a request to the first $\alpha = 3$ contacts, asking for $\beta$ closer contacts contained in the routing tables of the queried peers (in case of publishing $\beta = 4$, while in case of searching $\beta = 2$). Such a request is called *route request*. A timeout is associated to the Lookup process, so that, if the peer does not receive any reply, it can remove the stale contacts from the candidates, and it can send out new route requests.

**Processing Replies:** When a response arrives, the peer inserts the $\beta$ returned contacts to the candidate list, after having checked that they are not already present. Considering the modified candidate list, a new route request is sent if *(i)* a new contact is closer to the target than the peer that provided that contact, and *(ii)* it is among the $\alpha$ closest to the target.

**Stabilization:** The Lookup procedure terminates when the responses contain contacts that are either already present in the candidate list or further away from the target than the other top $\alpha$ candidates. At this point no new route requests are sent and the list becomes *stable*.

Note that, in every step of the Lookup procedure, only the peers whose KAD IDs share at least the first eight bits with the destination are considered: this is referred to as the **tolerance zone**. When the candidate list becomes stable, the peer can start the publishing or searching process. In case of publishing, the peer sends a '`store reference`' message to the top ten candidates in the candidate list. As a response to each publishing message, the peer receives a value called **load**. Each host peer can accept up to a maximum number of references for a given keyword or source, by default set to 50,000. The load is the ratio between the current number of references published on a peer and 50,000 (times 100). If the host node has a load equal to 100, even if it replies positively to the publishing node, it actually discards the publishing message; therefore, popular references may not be all recorded.

In case of searching, the peers sends a '`search reference`' message to the first candidate. If the response contains 300 references (sources), the process stops; otherwise, the peer iterates through the candidates until it has reached 300 sources. Note that a host node may have up to 50,000 references for a given keyword: in the reply, the host node will select randomly 300 references out of those.

### 2.3 Related Work

In Sect. 5 we propose essentially two solutions for dealing with popular contents: the use of stopwords, and a load balancing scheme. Here we discuss the related work on these two topics.

Stopwords have been used for decades in indexing and retrieval, but never for filtering the searches in P2P systems. The works in [10], [11], [12] and [13] provide general architectures that aim at building a full-text search engine, while in our approach we do not intend to support full-text searches over the entire document; we are just trying to enhance the indexing for file names in P2P systems.

Detailed measurement results from a study of Gnutella and Overnet (a precursor of KAD) are presented in the paper of Qiao and Bustamante [14]. Among other things, the authors evaluate the performance of queries in Overnet. Of particular interest to our work are their results on queries to popular keywords. They conclude that these are handled well by Overnet because it distributes the query load to multiple peers whose hash IDs are "close enough" to the hash of the keyword: the more popular the keyword, the broader the hash range. Our measurements contradict this conclusion: first, considerable overhead is generated by initially querying the peer with the closest hash to the popular keyword with load equal to 100; this goes on with an iteratively less precise hash value until a peer is found who is able to answer. Thus, a considerable additional load is imposed on the peers next to popular keywords. Second, we not only consider the querying but also the publishing load, which is much higher.

Load balancing for DHT systems has been extensively studied in the past: here we consider the most representative works. Many solutions [15][16][17] focus on the balancing of the responsibility zone, assuming a load uniformly distributed in the identifier space, while we consider the problem due to skewness in the popularity of the objects.

Many works based on the concept of *virtual servers* [18][19] have been devised to cope with heterogeneity of peer resources and content popularity: such schemes have a fixed number of possible peers to be used to balance the load. Instead, in our solution the content popularity itself drives the number of peers selected to store objects, and as such this number is not fixed a-priori. The work in [20], which is focuses on KAD, also uses a static, maximum number of peers that support load balancing. Moreover, such mechanism introduces a set of new messages that requires a modification to the original KAD protocol. Our solution, instead, is based solely on the currently available messages on KAD, and does not change the protocol, which is a clear advantage because it is backward compatible with existing KAD deployments.

Other works [21][22] consider the transfer of the content from overloaded peers to underloaded ones (content migration): the load balancing is initiated by the storing peers (host nodes) and incurs in a high overhead. In our scheme, the load balancing is performed by the publishing peers, without any additional overhead w.r.t.

the basic KAD scheme. The authors in [23] – another work specific to KAD – propose a load balancing scheme which is not adaptive, and does not avoid the loss of information.

## 3 Measurement Tools

In what follows, we provide an extensive set of measurement results that explain how content management in KAD works.

To this aim, we designed a series of tools which derive a vast amount of information on KAD behavior. The first measurment tool is an instrumented aMule client that logs the internal state of a KAD client. For instance, it detects the candidate list which is built for each published reference. Another tool, designed and implemented for this work, is a content spy called *Mistral*, described in Sect. 3.1. Finally, we designed and implemented a KAD crawler, called *Blizzard*, which has been presented in [7]: even this tool does not represent a contribution of this paper, we describe it in Sect. 3.2 for clarity of exposition.

### 3.1 Spying for Content with *Mistral*

*Mistral* is based on the same principle as the *Sybil attack* [24], [25], [26]. We introduce a large number of our own peers, the *sybil*s, into the network, all controlled by one machine. Positioned in a strategic way in the KAD space but physically all running on the same machine, the Sybils can gain control over a fraction of the network or even over the entire network. The fact that all Sybils run on the same machine has the advantage that data collection is much easier.

We insert a large number of Sybil peers into the network and propagate information about them in the routing tables of the legitimate peers. To do so, we first crawl KAD using *Blizzard* (see Sect. 3.2) to learn about the peers in the network. Next, we send `hello` messages to the peers we have learned about. A `hello` message is 120 bit long and includes the KAD ID of the sender, and this can be arbitrarily chosen. In *Mistral*, the first 24 bit of a `hello` message are chosen at random while the 96 remaining bits are fixed.

The routing queries reaching the Sybils are always answered with other Sybils. The returned KAD ID is closer to the target included in the query than the receiver of the query, thus the querying peer has always the impression of approaching the target. Once the requester gets close enough to the target, it queries a Sybil for the content itself and not for any closer peers. Our Sybil stores the search request and returns a fake source entry. This source entry points to our machine. As a consequence, the real peer tries to start to download which is not successful.

With our tool, we retrieve routing and search requests toghether with publish request messages. As stated above, these requests are especially interesting since they are much more frequent than search requests. Whereas search requests are always launched by a human, publish requests are automatically and regularly launched by the KAD clients. Also, the publish information is richer than the search requests: it includes the full file name, the KAD ID of the source and a significant amount of metadata on the file. As explained above, the filename is tokenized and published on the part of the DHT corresponding to the hash of *each* of its tokens (that is, its keywords). The answer to a publish request is the load of the peer addressed. The Sybils always answer with a very low load, thus attracting more and more publish requests.

An eight-bit zone contains the peers with KAD IDs that agree in the first eight bits, thus each zone can theoretically contain $2^{120}$ hash values. We actually observe between 12,000 and 25,000 peers per zone. The entire KAD network contains 256 eight-bit zones and between 3 and 5 million peers. It is possible to spy on one zone of the KAD network only by restricting the returned KAD IDs to a certain prefix. We insert 65,356 distinct Sybils into a zone to make sure to catch at least one of the ten publish messages for a keyword or a source and at least one of the three search messages that are sent per user-initiated search.

The approach adopted by *Mistral* was possible until May 2008: starting from eMule version 0.49a and aMule version 2.2.1, in fact, the developers have inserted a set of rules that limit the Sybil attack. In practice, each peer will ignore multiple KAD IDs pointing to one IP address: the only way to perform today a measurement with *Mistral* would be to have a distributed set of coordinated nodes [27] (e.g. PlanetLab). As a consequence, the results presented in Sect. 4.1 can not be simply replicated nowadays. Nevertheless, the changes in the code of the applications that use KAD have mainly focused on security issues, but the basic components of KAD have not been modified. In Sect. 4.2 we perform a set of simple tests, using the current versions of eMule and aMule, which show that the main results obtained with *Mistral* are still valid.

### 3.2 Crawling KAD with *Blizzard*

*Blizzard* logs, for each peer $P$, the IP address of $P$, the KAD ID of $P$, and whether or not has responded to the crawler. *Blizzard* has been designed to be extremely fast: since peers constantly arrive and leave, the crawling speed represents an important aspect to get a consistent view of the system.

The implementation of *Blizzard* is simple: it starts by contacting a seed peer run by us. Then it asks the seed peer for a set of peers to start with, and it uses a simple breadth first search and iterative queries. It queries the peers it already knows to discover new peers. For every peer returned, the crawler checks if this peer has already been discovered during this crawl. After the crawl is completed, the results are written to disk.

Since in this work we will use *Blizzard* as a supporting

tool, we refer the interested reader for implementation details to [7].

## 4 Measurement Results

In this section we study the KAD publishing procedure looking at the corresponding generated traffic. We first record for 24 hours a set of eight-bit zones with *Mistral* (Sect. 4.1). We then analyze in detail the traffic over time, focusing on a couple of popular keywords in Sect. 4.2. Since KAD publishes ten copies for each reference, we analyze in Sect. 4.3 where these replicas are placed, and in Sect. 4.4 we study the reasons for the results obtained in Sect. 4.3.

### 4.1 Analysis of the Traffic

Given a reference, KAD publishes it on peers whose KAD ID shares at least the first eight bits with the KAD ID of the reference (tolerance zone). For this reason, the traffic on a single eight-bit zone can be studied independently from the other zones. The analysis of the entire KAD ID space (i.e., 256 zones) would be impractical due to the high traffic generated. Assuming a network with three millions online peers – this is the average number of peers, as shown in [7] – we need to crawl the network with *Blizzard* at least every two hours to cope with the churn in the system. Each crawl accounts for 4 GByte of traffic. Afterwards, the Sybils must be announced to those peers. Suppose that we only announce to each peer the 256 closest Sybils: one announcement costs 50 bytes plus another 50 bytes for the ack; that accounts for $3,000,000 * 256 * 2 * 50$ bytes = 72 GByte. Announcements must be done periodically: on average, announcements of Sybils generates about 40 MBytes/s of traffic. Moreover, the Sybils will also attract search and publish messages.

Besides the traffic generated for spying on the entire KAD ID space, since the different tolerance zones are independent, it is sufficient to focus on some sample zones to obtain interesting information about KAD. For this reason, we spied on 20 different eight-bit zones of the KAD ID space for 24 hours. During this time, on average, 4.3 million publish messages, 350,000 search messages and 8.7 million route messages were recorded. The publish messages contained 26,500 different keywords per zone, most of them in Latin letters, and 315,000 distinct sources, i.e., 315,000 distinct files. Among the 65,356 Sybils we introduced, on the average 62,000 were hit by search or publish requests.

The hash values of the sources and of the keywords are uniformly distributed over the KAD ID space. Similarly, we know from our earlier measurements with *Blizzard* [7] that the peers are roughly uniformly distributed on the KAD ID space.

This property allows us to estimate the total number $\mathcal{S}$ of sources (files) in the system by simply counting the number of sources in a zone. Let $\mathcal{S}_{part}$ be the number of sources counted in an eight-bit zone, and $\hat{\mathcal{S}} := 256 * \mathcal{S}_{part}$ the estimate for the total number of sources in the KAD system. Using Chernoff bounds (see [28] Chapter 4) we tightly bound the estimation error. Indeed, $Prob(|\mathcal{S} - \hat{\mathcal{S}}| < 45000) \geq 0.99$, which means that our estimate $\hat{\mathcal{S}}$ has most likely an error of less than 3% for a total number of at least 80 million sources.

The most important result that we observed is that, independently from the zone, our measurements show that there are *ten times* more publish messages than search messages. This result is confirmed by recent measurements, as shown in Sect. 4.2. Moreover a publish message is ten times bigger than a search message since it contains not only a keyword but also metadata describing the published content. This is true also with the current version of KAD, since the message format has not been changed.

The number of times a keyword publication is observed versus the ranking of the keyword for the eight-bit zones `0xe3` and `0x8e` are shown in Figure 1 in a log-log scale. Rank 1 is the most popular keyword. If each curve were a straight line, the popularity of keywords would follow a Zipf-like distribution (i.e., the probability of seeing a publication message for the $i$'th most popular keyword is proportional to $1/i^{\alpha}$ [29]). We used Matlab's curve-fitting tools to estimate the value of $\alpha$, for the curve. The value of $\alpha$ is the same for all zones: $\alpha \approx -1.63$.

We picked two zones as examples. The zone `0xe3` contains the keyword "the" whereas the zone `0x8e` does not contain any popular keywords. The keyword "the" in zone `0xe3` accounts for 30% of the total load in the zone. In total 1,518,717 publish requests with the keyword "the" hit our Sybils in 24 hours. In contrast, in zone `0x8e`, the most popular keyword accounts only for 5% of the load. In this zone the most popular keywords are nearly equally popular.

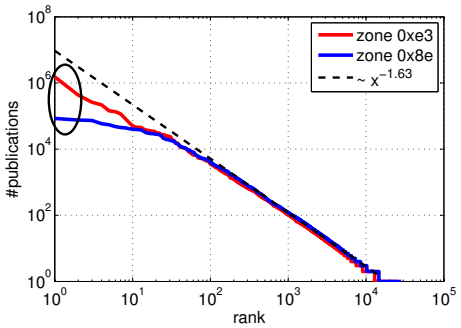



Fig. 1. The number of publications per keyword for two different zones.

Figure 2 shows the number of queries that hit our ten most loaded Sybils in the two zones `0xe3` and `0x8e`. The popular keyword "the" in zone `0xe3` is mainly responsible for the high load on these Sybils. The Sybils with a lower rank have the same load in both zones.

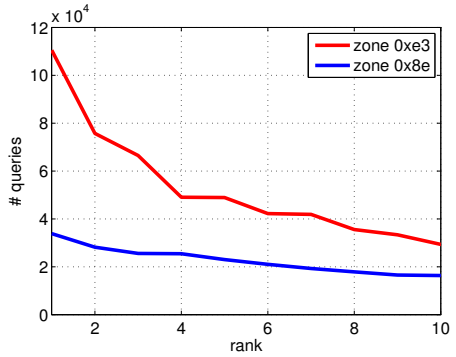
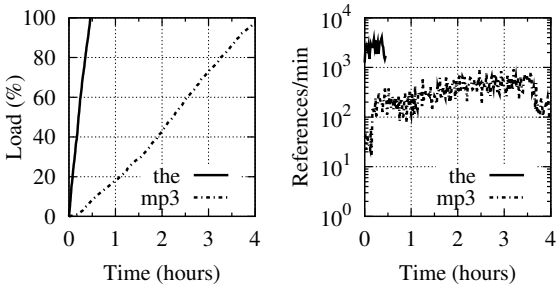



Fig. 2. The number of queries received by the Sybils for two different zones.




(a) Load over time (b) Publishing frequency over time

Fig. 3. Load and publishing frequency over time registered by our instrumented client.

### 4.2 Reliability and Diversity

In the previous section, we have discussed how we recorded the traffic with our Sybils for 24 hours, and we have shown that popular keywords account for most of the load. For this reason, we need to investigate further how KAD handles popular objects. This is important since the number of references a peer can hold for a given object is limited to a maximum value (50,000): what happens when this limit is reached?

When the host peer reaches its maximum number of references, it replies positively to any publishing request, but actually discards the reference. Therefore, all the following publishing traffic represents a waste of resources, such as the download bandwidth for receiving the messages, the processing power for processing them, and the upload bandwidth for replying.

From the publishing node's point of view, this translates into a decreased reliability: the probability over time to find a reference to the publishing peer will be significantly lower in case of popular objects w.r.t. non-popular objects – the interested reader is referred to [30] for a detailed evaluation of the impact of the number of replicas on the reliability.

In order to understand how fast the maximum limit can be reached, we focus on the traffic recorded by an instrumented aMule client placed closed to two popular keywords. For the sake of experimental reproducibility, the keywords we consider are *static* popular keywords, *i.e.*, keywords that are usually present in the file names, such as "the" or "mp3." It is reasonable to assume that the results we present here can be considered equivalent to those that can be obtained during transient peaks of popularity for other keywords (such as "ubuntu" immediately after a new release).

Figure 3 shows the load (ratio between the current number of references and 50,000, times 100) and the frequency of the publishing requests over time. Note that a single publishing message may contain multiple publishing requests, since a keyword may be associated with many files.

Our measurements show that the host peers located close to a popular keyword saturate in only a few minutes after joining the system; upon saturation, all the publishing messages are wasted traffic. In particular, the traffic amounts to 3.5 publishing messages per second, and 0.3 searching messages per second. Note that this result confirms the main finding of Sect. 4.1, *i.e.*, there are ten times more publish messages than search messages. The corresponding total amount of incoming traffic is approximately equal to 30 kbit/s. Even if this value seems affordable by most of today's Internet connections, we note that the actual available bandwidth of ADSL users may be less than 500 kbit/s, therefore such traffic decreases by 6% the available bandwidth to peers laying in a hot spot.

In addition to the above mentioned problems, there is another issue, which we call *diversity*. The publishing phase is complemented by the searching phase: the searching peer starts querying the top-ranked peers in its candidate list, and, if it obtains at least 300 references, the searching phase stops. For popular objects, the first peer in the candidate list will have most probably more that 300 references (even if it has just arrived, it takes only a few minutes to receive more than 300 publishing messages). Therefore, even if the references are replicated ten times, if all the replicas are on saturated nodes, the publishing peer may be never be contacted by other peers, and it will not contribute with its resources to the P2P system. Let $\mathcal{S}(t)$ be the set of peers that owns a specific object at time $t$. Due to churn and, in case of popular keywords, due to the limited number of references held by a host peer, the peers close to the target will have a subset $\mathcal{S}'(t)$. Since the searching peers focus on a limited set of peers close to the target, they will obtain references from $\mathcal{S}'(t)$, instead of from $\mathcal{S}(t)$. We call diversity the ratio between $|\mathcal{S}(t)'|$ and $|\mathcal{S}(t)|$: the system should ensure a diversity close to one, despite churn and keyword popularity.

If we look at a popular object, and we observe a short period during which the churn can be considered negligible, a diversity smaller than one has a direct impact on the performance of the system, precisely on the actual content transfer phase. The searching peers, in fact, retrieve references belonging to $\mathcal{S}'(t)$, *i.e.*, they will download the content from the peers in $\mathcal{S}'(t)$. If such

peers in $\mathcal{S}'(t)$ have limited resources, they will put the searching peers in a waiting queue, increasing the overall download time, while other peers in $\mathcal{S}(t) \setminus \mathcal{S}'(t)$ will stay idle instead of serving the content. In other words, the system is not able to exploit all the available resources; it is not running at its full service capacity.

### 4.3 Load Distribution

In the previous section we have shown the load over time of a Sybil placed close to a popular keyword. For a given reference, a publishing node places ten replicas. Therefore, it is interesting to understand the load on the entire eight-bit zone where there is a popular keyword. For this experiment, we will not use the Sybils, since we study the impact of popular keywords on real, legitimate peers.

The experimental methodology is as follows. Given an eight-bit zone, with the help of *Blizzard* we obtain the list of all the peers that are alive (stale contacts are removed). We send a publish message to all peers, obtaining as a response the *load* from each of them. We collect the replies and we sort them according to the XOR-distance to the KAD ID of the keyword, obtaining a snapshot of the current load distribution.

Figure 4 shows the results for two popular keywords ("dvdrip" and "mp3"). We tested such keywords (and others, not shown here for space constraints) in different days and hours within a day, obtaining similar results. The x-axis contains the distance from the target KAD ID as a percentage of the maximum distance: since the KAD ID is composed by 128 bits, a peer with all the bits of the KAD ID different from the bits of the KAD ID of the keyword (except for the first eight, since we focused on an eight-bit zone) would have distance $d_{max} = 2^{120}-1$. A peer with all the bits of the KAD ID different from the bits of the KAD ID of the keyword, except for the first twelve, would have distance $d = 2^{116} - 1$, i.e., $d/d_{max} = 6.25\%$.

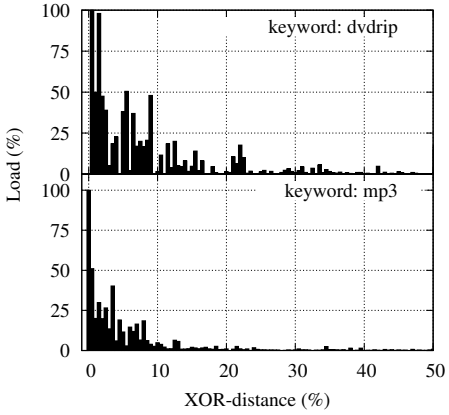



Fig. 4. Load distribution for two popular keywords.

For clarity of presentation, we have divided the x-axis into bins; each bar in the figure represents the load of approximately 8-10 peers (the value is the mean load of such peers). For very popular keywords, not only the closest peers to the target are overloaded, but there is a high fraction of peers away from the target that has a significant load. The snapshot clearly can not capture the dynamics of the zone, *i.e.*, peer arrivals and departures: the effect of node dynamics determines the irregularity in the shape of the load distribution, but it can not justify the high load in peers far from the target. As an example of an keyword with low popularity, in Fig. 5 we show the distribution of the load of the keyword "dexter," where the replicas are roughly concentrated around the target.

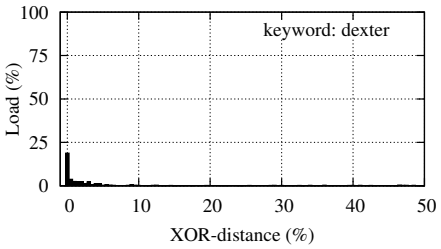



Fig. 5. Load distribution for the slightly popular keyword.

At first glance (cf. Figs. 4 and 5), it appears that KAD inherently distributes the load among increasingly distant peers when objects are popular. Unfortunately, as we will see in Sec. 4.4, this effect has not been included *intentionally* in the design of KAD, but results from an imperfect Lookup procedure. Actually, a closer look at Fig. 4 reveals some issues. For the keyword "dvdrip," we can see that there are peers at 42% XOR distance (which corresponds to a peer sharing the first *ten* significant bits with the target) with a load equal to five, which means approximately 2500 references. At the time of the snapshot, the number of peers between such peers and the target is approximately 800. Since churn alone may not justify such a spread, this result requires a deeper analysis of the publishing procedure.

### 4.4 Accuracy of the Candidate List

In this section we investigate the effectiveness of the candidate list building process as implemented in the Lookup procedure. The candidate list represents a snapshot of the current peers around a target that the publishing (or the searching) peer builds with the help of other nodes. This process is similar to the process of crawling KAD: the designers need to face different trade-offs, such as the accuracy of the results versus the traffic generated, or versus the time it takes to build the list. In KAD, the building process stops (i.e., the candidate list is considered stable) when the peers do not receive any contact closer than the top $\alpha$ ($\alpha = 3$ by default) already present in its candidate list for three seconds. This means that the focus is on the top positions of the candidate list, while the other positions may not be accurate.

Let $\mathcal{L}$ be the list of peers whose KAD IDs share the first 8 bits with the KAD ID of a given target; $\mathcal{L}$ is sorted according to the XOR distance to the target, closest first. Let $\mathcal{L}'$ be the candidate list built by the Lookup

procedure. The list $\mathcal{L}'$ is a (ordered) subset of $\mathcal{L}$. For simplicity, instead of the element itself, $\mathcal{L}'$ contains the order of the elements in $\mathcal{L}$.

In order to evaluate the accuracy of $\mathcal{L}'$ w.r.t. $\mathcal{L}$, we set up a measurement campaign using Blizzard. We place a content in the shared folder of an instrumented aMule client: this triggers the publishing process, whose related messages (requests and replies) we register. In the meantime, we crawl with Blizzard the KAD ID zone corresponding to the keywords and source of the content. The publishing process and the crawling process last for two minutes, making the effect of churn negligible. With the output of the crawl we build $\mathcal{L}$, while with the logs of our instrumented client we build $\mathcal{L}'$. We repeat this process several times, for different keywords and sources, in order to gain statistical confidence. An example of the outcome of the experiment is given in Table 1 (basic Lookup): for a given row, we show the index of $\mathcal{L}'$.

TABLE 1
Examples of $\mathcal{L}'$.

| ID | order in $\mathcal{L}$ (basic Lookup) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 2 | 4 | 5 | 6 | 21 | 35 | 95 | 187 | 310 |
| B | 1 | 3 | 10 | 12 | 15 | 58 | 84 | 134 | 456 | 1232 |
| C | 2 | 6 | 13 | 14 | 39 | 40 | 43 | 77 | 89 | 716 |
| ID | order in $\mathcal{L}$ (improved Lookup) | | | | | | | | | |
| D | 2 | 3 | 6 | 9 | 10 | 11 | 12 | 15 | 20 | 27 |
| E | 1 | 2 | 4 | 6 | 7 | 8 | 9 | 10 | 11 | 13 |
| F | 1 | 4 | 6 | 7 | 8 | 10 | 13 | 14 | 17 | 19 |

While the first few positions contain almost the same elements of $\mathcal{L}$, the other elements of $\mathcal{L}'$ are scattered on a wider KAD ID space. In order to quantify the accuracy of $\mathcal{L}'$ w.r.t. $\mathcal{L}$, we estimate the probability that an element of $\mathcal{L}$ is chosen during the candidate list building process. For ease of representation, we assume that the candidate list building process can be modeled as a Bernoulli trial process, with success probability $p_i$ that depends on the position in $\mathcal{L}'$. For instance, for the first element of $\mathcal{L}'$ we pick the elements from $\mathcal{L}$ with probability $p_1 = 0.55$; once the first element is selected, we consider the elements of $\mathcal{L}$ with probability $p_2 = 0.5$, and so forth. For the estimation of the probabilities $p_i$, we consider the results of the measurements, and we take the difference between the positions in $\mathcal{L}$ for two consecutive elements of $\mathcal{L}'$. Fig. 6 shows $p_i$, the probability to pick an element from $\mathcal{L}$ to be put in the position $i$ of the list $\mathcal{L}'$, along with the 95% confidence interval (obtained with approximately 30 independent experiments). The graph shows that the Lookup procedure is largely accurate in selecting the first 2-3 positions, but the elements in the lower positions of $\mathcal{L}'$ are far from the target. The candidate list building process, therefore, revealed to be imperfect and inaccurate, especially for the lower positions: this explains the spread of the reference for popular keywords.

Such an inaccurate candidate list has several problems, e.g., the ninth and tenth replicas are published so far from the target that they may be never considered during the search phase. A redesign of the Lookup procedure is out of the scope of this paper. Note that the authors in [23] impute the inaccuracy to the routing management, while our experiments indicate that the main issue lies in the Lookup procedure. To support this claim, we report here our experience during the tests, in which we have modified the values of some constants in the Lookup procedure to understand their role in the lookup process. During the Lookup procedure the peer asks for $\beta$ closer contacts contained in the routing tables of other peers. By increasing the value of $\beta$, it should be possible to increase the accuracy of the candidate list. For instance, we set $\beta = 16$ and obtained the probabilities $p_i$ labeled as "improved Lookup" in Fig. 6 – examples of the candidate lists can be found in Table 1 (improved Lookup). We notice that the accuracy of the list in the last positions is increased[1]. Rather than trying to increase further the accuracy, we will exploit such inaccuracy in the design of our load balancing scheme.

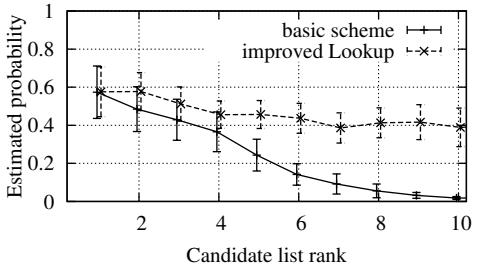



Fig. 6. Estimated probability for building $\mathcal{L}'$.

### 4.5 Summary of the Issues

Our measurements show that there are *ten times* more publish messages than search messages: since most of the traffic is due to popular content, we should focus on the management of the references to popular objects. For such references, our analysis has highlighted different issues on the current KAD procedures related to content publishing. In particular, in case of popular objects,

- some peers residing in a hot spot must support a management traffic that decreases the available bandwidth, which may cause unfairness among peers;
- many references are lost since they are published on overloaded peers that discard them: as such, diversity decreases;
- the search phase considers mainly the peer closest to the target, without considering that some references may be found on other peers. This problem has a broader impact than it first appears: content transfers are limited to a small number of peers,

1. The modification of the parameter $\beta$ has been done to test if it is possible to increase the accuracy, therefore we have not evaluated the impact of $\beta$ on the traffic generated by the application; as we said, the Lookup procedure would need a complete redesign, which is out of the scope of this paper.

as compared to the whole set of peers hosting the content, which implies increased delays.

In practice, KAD has been designed without considering in depth the heterogeneous content popularity. Note that a naive solution in which we increase the maximum number of stored references on peers would not solve the above mentioned issues. As a general guideline, the design of the publishing process should consider its counterpart, the searching process. With a joint design, it is possible to take into account aspects, such as diversity, or dynamic load balancing, and provide an efficient solution that a separated design approach may not obtain.

In the next sections we will consider different solutions for more effective treatment of popular objects. For instance, we will show how to exploit the imperfect design of the current KAD Lookup procedure to provide a dynamic load balancing scheme, that not only decreases the burden on hot spots, but also increases diversity.

| The stopwords for KAD | | | The Google stopwords | | |
|---|---|---|---|---|---|
| stopword | # peers | # files | stopword | # peers | # files |
| avi | 491 | 8101 | about | 513 | 7608 |
| xvid | 479 | 13683 | are | 330 | 7282 |
| 192kbps | 437 | 8005 | com | 463 | 11550 |
| dvdscreener | 413 | 12343 | for | 549 | 12303 |
| screener | 433 | 7377 | from | 399 | 8345 |
| jpg | 456 | 10529 | how | 542 | 8282 |
| pro | 303 | 8378 | that | 423 | 9148 |
| mp3 | 482 | 12019 | the | 487 | 14502 |
| ac3 | 424 | 8045 | this | 452 | 8510 |
| video | 468 | 10478 | what | 394 | 7710 |
| music | 335 | 8558 | when | 294 | 7241 |
| rmvb | 454 | 13643 | where | 431 | 9445 |
| dvd | 450 | 10194 | who | 302 | 7742 |
| dvdrip | 560 | 13235 | will | 458 | 7976 |
| english | 388 | 7849 | with | 338 | 8543 |
| french | 377 | 9468 | www | 391 | 11203 |
| *dreirad* | 28 | 30 | and | 577 | 13706 |

TABLE 2
The KAD and Google stopwords with more than two letters, the number of peers storing them and the number of files containing them. For comparison the rare keyword "dreirad" is shown.

## 5 Solutions

Keyword popularity derives from two main reasons. Either the object (the name of which contains the keyword) is popular, or the keyword itself is a common keyword that can be found in different objects. Examples of the latter case are the keywords "the," "mp3" and "dvd": most of the files contain these keywords, but such keywords do not characterize or describe the content of the files. It is clear that, if the system avoids publishing common keywords, the search procedure is marginally affected: indeed, users seldomly specify a lookup requests using such common keywords. In Sect. 5.1 we show the straightforward solution for the common keywords. In Sects. 5.2 and 5.3, instead, we focus on file popularity, which may be variable over time.

### 5.1 Common Keywords

The simplest solution for dealing with the common keywords is to ignore them. This approach require the identification of the so called *stopwords*, i.e., words that can be filtered out with no impact on the usability of the system. Table 2 (first column) shows specific stopwords for KAD file names which complement the set used in by popular Internet search engines (Table 2 fourth column) [31]. The number of peers on which a stopword is published (second and fifth columns), as well as the number of files containing the stopword (third and sixth columns), have been determined by first crawling the peers around the stopword with *Blizzard* and then by querying all those peers for the stopword.

We propose to treat all the keywords contained in Table 2 as stopwords. From the implementation point of view, the solution is simple: as described in Sect. 2.2, when a Publish or Search is performed, KAD creates a thread for each keyword. KAD should check if the keyword is a stopword before launching the thread. The list of stopwords should be sufficiently stable, so that including it in the source code represents the simplest solution. Alternatively, the list could be dynamically updated as it happens for the bootstrap nodes, where websites maintains the list that can be used by new clients to find nodes from which to bootstrap the connectivity.

### 5.2 Adaptive Content Publishing

In this section, we consider objects with a time-varying popularity, and show how the system should manage the references in order to avoid the issues summarized in Sect. 4.5. We assume that stopwords are managed as discussed in the previous section, therefore the popularity of a reference comes from the popularity of the object the reference points to.

In Sect. 4.4 we have analyzed the accuracy of the Lookup procedure. Even if the inaccuracy of the candidate list may seem a problem, this appearent drawmback can be turned in a way to perform load balancing: the probability that two publishing peers have the same candidate list at the same time is low, thus they publish their replicas on different peers. This is true only starting from the third or fourth position onward, while usually the first three or four positions are accurate, i.e., they are the almost same for the different publishing peers.

In our design, we exploit the inaccuracy of the candidate list: since we have shown that with the basic KAD scheme the accuracy is extremely low for the last positions in the candidate list, we assume an improved Lookup procedure (as shown in Fig. 6). We then focus on the publishing and the search procedures to perform an

```
Procedure Publish
  Data: list: candidates /* peers ordered by their
        distance to target                          */
  Data: int: curr /* current candidate               */
  Data: bool: direction /* used to decide how to
        iterate through the candidates              */
  Data: list: thresholds /* for deciding if an
        object is popular or not                    */
  Data: int: maxLoad
1 Initialization:
2   curr = 9;
3   direction = backward;
4   maxLoad = 80;
5
6 for i ← 0 to 9 do
7   contact ← candidates.get(curr);
8   load ← publish(contact);
9   if curr < 10 and load > thresholds.get(curr) then
10    direction = forward;
11    curr = 9;
12  if curr ≥ 10 and load > maxLoad then
13    curr += (10 - curr%10);
14  if direction == forward then
15    curr++;
16  else
17    curr−−;
```

adaptive load balancing based on object popularity. We modify only the algorithms, without introducing new messages or modifying the existing ones, so that our solution is completely backward compatible with the current KAD protocol. Moreover, for non-popular objects, the proposed solution behaves exactly as the current KAD scheme.

Given the candidate list produced by the Lookup procedure, the publishing procedure tries to publish ten replicas of the reference. The basic idea of our solution is as follows: we use the value of the *load* (which is returned by a peer as a response to a publish) as an indication of popularity, and we drive the selection of the candidates according to it. In case of popular objects, instead of trying to publish the references on the best host peers, the publishing peer should choose candidates progressively far from the best target.

In order to obtain the load, the publishing peer needs to publish the content: since we want to avoid the risk to overload the closest host node, instead of publishing starting from the first peer in the candidate list, the publishing process should start from the tenth peer. If the load is below a certain threshold, the publishing peer publishes the next replica on the ninth candidate, otherwise it considers the candidates with a rank worst than the tenth.

The `Publish` procedure shows the details of our solution. As input, we provide a vector of thresholds used to identify if the object is popular. Such thresholds are set only for the first ten positions, and they are higher and higher as we get close to the top ranked candidates. In particular, let $D_{\max}$ and $D_{\min}$ the thresholds for the first and the tenth candidates, respectively. For simplicity we assume that the growth of the threshold is linear with the rank of the candidates, i.e., the threshold for the $i$th candidate, $D_i$, $i = 0, 1, \ldots, 9$, is given by $D_i = D_{\max} - \big(D_{\max} - D_{\min}\big)i/9$.

If the publishing peer finds a candidate with a load greater than the threshold, then it publishes the remaining replicas starting from the eleventh node and onward. Note that if the load is above the threshold at the beginning of the publishing process, the object is considered very popular, and all the remaining replicas will be more scattered (since the publishing node will consider up to the 19th candidate). If the threshold is never exceeded, the publishing node publishes on the top ten ranked peers, as in the current KAD implementation.

If the object is extremely popular, then the candidates that usually occupy the 11th position up to the 19th position may become overloaded, too. In this case, we have introduced a maximum value of the load, equal to 80: if this value is reached, we start considering the candidates from the 20th position up to the 29th, and so forth. In this way, as the number of publishers increases, we add more and more peers for storing their references.

We would like to stress the fact that the `Publish` procedure has a very simple form, thanks to the specific way in which the candidate list is built. In Sec. 5.4 we will discuss how to modify the approach in case of an extremely accurate candidate list. Moreover, our solution represents a modification of an existing (and widely deployed) system: for this reason we cannot introduce a set of mechanisms or messages that would facilitate the load balancing process – for instance, we may introduce a message for knowing the load of a host peer without the need to publish on it. Our contribution lies in the design of a load balancing scheme based solely on the available KAD messages.

### 5.3 Content Search

In the current KAD implementation, when a peer is looking for references to an object, it stops the search process as soon as it receives at least 300 references. A single reply may contain 300 references, therefore a single query may be sufficient. In case of popular objects, it is possible to find peers that hold more than 300 references even if they are not close to the KAD ID of the object. Such peers are rarely used, with a consequent decrease in diversity.

The simplest solution to overcome this limitation is to introduce some randomness in the searching process. Given the candidate list, instead of considering the first candidate, the searching node should pick randomly one of the first ten candidates. If the answer contains 300 references, the process stops. Otherwise, the searching node needs to pick another candidate. The `Search` procedure shows the details of our proposed solution.

In the procedure, we use the following heuristic: the searching node tries twice with a random candidate; if it does not receive enough references, it falls back to

```
Procedure Search
  Data: list: candidates /* peers ordered by their
        distance to target                        */
  Data: list: references /* obtained refs          */
  Data: int: maxRandomTentatives
  Data: int: maxIndex
1 Initialization:
2   maxRandomTentatives = 2;
3   maxIndex = 10;
4   references = {∅};
5
6 while references.size() < 300 and candidates not empty do
7   if maxRandomTentatives > 0 then
8     contact ← candidates.getRandom(maxIndex);
9     references.add(search(contact));
10    maxRandomTentatives−−;
11  else
12    contact ← candidates.getFirst();
13    references.add(search(contact));
14  candidates.remove(contact);
```

the basic scheme, i.e., it starts from the first candidate. This heuristic derives from the fact that, if a candidate has less than 300 references, there could be two reasons: either the object is not popular, or the candidate has just arrived, and it had little time to record the references. In case of a non-popular object, this process results in an overhead. We believe that, thanks to the gain in terms of diversity and load balancing in case of popular objects, such an overhead is a fair price that can be paid: measurement studies [32] have shown that a few popular files (approximately 200) account for 80% of the requests, therefore the impact on non-popular objects should be acceptable.

The proposed solution for the search procedure works also in case of adoption of our proposed publishing procedure: the references to popular objects will be scattered around the target, and a random search scheme will be able to easily find them.

### 5.4 Discussion

In this section we comment on different aspects related to the proposed scheme, including security considerations, parameter settings, and peer churn. We do not discuss the introduction of new messages, which would simplify the load balancing, since, as we stated before, we aim at proposing a solution that does not modify the KAD protocol.

**Accuracy of the candidate list:** In our measurement campaign, when we have derived the accuracy of the candidate list, we have shown the results up to the tenth position. Our proposed load balancing scheme considers the positions with a lower rank. In case of our improved Lookup procedure (where we have set the $\beta$ parameter to 16) we have assumed that the accuracy remains the same up to the 20th position, thanks to the high number of peers in the candidate list. Preliminary tests with a prototype implementation of our load balancing scheme in a instrumented aMule client have shown that this assumption is reasonable.

**Improving accuracy:** The proposed scheme (in both publish and search procedures) relies on the fact that the candidate list is accurate in the first few positions, and progressively inaccurate in the other positions. This is specific to the implementation of the Lookup procedure in KAD (both in the basic implementation and with our modification). One may ask what would happen in case of an improvement of the Lookup procedure, such that it provides an extremely accurate candidate list. The solution would be straightforward: it is sufficient to reproduce the inaccuracy of the current Lookup procedure. By adopting this approach, our proposed scheme remains sufficiently general, yet maintaining its simplicity.

**Keeping the history:** For each published reference, there is an expiration time associated to it, after which the reference is republished. A publishing peer can maintain information about the popularity of an object. It may be a simple flag that indicates that in the previous publishing process the object was popular, in order to influence the peer candidate choice. We will evaluate this enhancement in future work.

**Parameter setting:** The `Publish` procedure has a set of parameters, namely the thresholds used to discriminate between popular and non-popular object. Changing such thresholds has an impact of the effectiveness of the proposed solution: low thresholds may spread too much the references, while high thresholds may detect a popular object too late. Unfortunately there is no a simple distributed solution to this problem: a centralized solution – e.g., a server that keeps track of object popularity – is impractical and subject to security problems; a solution based on gossiping increases the overhead and may not assure that the information is available when it is needed. In both cases, the designer would introduce new messages, changing the KAD protocol. The use of thresholds is the simplest solution that does not require significant modifications to KAD. In our case, we have used the measurements showed in Sec. 4.3 to set the thresholds. As for the `Search` procedure, there are two parameters: the number of random tentatives and the maximum rank in the candidate list. In Sec. 6.2 we study them in a synthetic environment. As a future work we plan to perform a measurement campaign to evaluate their impact in real environments.

**Security considerations:** Here we consider attacks specifically related to our scheme. A malicious peer could return a load of 100 even if the object is not popular, or a load of 0 even if the object is popular. If the peer is very close to the reference KAD ID, in both cases the effect would be minimal. If the malicious peer is far from the reference KAD ID (i.e., it tries to be in the ninth or tenth position of the candidate list), the inaccuracy of the candidate list would limit the impact of such malicious behavior. In order to be effective, a malicious

peer should perform these types of attacks in conjunction with a Sybil attack: therefore, any solution that prevents a Sybil attack [33] is sufficient to weaken the attacks to our scheme. As for the eclipse attack, since our scheme tends to scatter in a wider zone the references of popular objects, we have as a by-product a countermeasure to such a malicious behavior.

**Churn:** Considering a specific target KAD ID, the peers around such target change over time. The candidate list of a publishing peer may contain newly arrived peers (they do not contain stale contacts, since the Lookup procedure eliminates them): during the publishing process, a newly arrived peer has a low load, thus the publishing peer may consider the object not popular. The impact of this aspect is minimal, since eventually the candidate list should contain a peer with the load above the threshold. In any case, publishing on newly arrived peer is not a big problem since they have a low load.

## 6 Numerical Results

In order to assess the effectiveness of our solution, we take a simulation approach: an evaluation based on real modified peers, in fact, would be impractical for many reasons. For instance, the generation of the publishing traffic for a popular keyword requires a high peer arrival rate, each of them with a different KAD ID and a differentiated candidate list building process; such a process needs different initial neighbor set, since starting from the same set of neighbors may result in correlated candidate lists, which in turn affects the publishing and the searching process.

### 6.1 Simulator Description and Settings

For the evaluation of the load balancing scheme, we need two key ingredients: (i) the peer dynamics (arrival and departure) should be realistic, and (ii) the candidate list should have the same accuracy as in the current KAD implementation. We should have full control of these two aspects in a simulator: we have considered the few available KAD simulators [34][35], and none of them provides such control. For this reason we decided to implement a custom event-driven simulator [36].

The peer arrivals and departures follow the publicly available traces collected over six months from the KAD network [37]: the simulator takes as input the availability matrix of all the peers seen in a specific zone and generates the corresponding arrival and departure events, reproducing the dynamics of real peers measured over a six month period.

Given the set of peers that are online at a given instant, and given a target KAD ID, we are able to build an accurate list $\mathcal{L}$. Starting from $\mathcal{L}$, we build the candidate list $\mathcal{L}'$ following the procedure explained in Sec. 5.4, with the help of the measurements presented in Sec. 4.4. For the basic KAD scheme and our load balancing scheme, we have used the results shown in Fig. 6, labeled as "basic scheme" and "improved Lookup," respectively.

Besides the peer availability matrix, the inputs of the simulator are (i) the target KAD ID, (ii) the starting publishing instant, (iii) the observation time, and (iv) the publishing rate. The target KAD ID can be set to check if there is a bias in the KAD ID space – which we actually never observed, so any KAD ID can be used. With the starting publishing instant, we can set the point in time, within the six months period, when the peers can start publishing the content. Once started, we observe the evolution of the publishing process for the observation time. The publishing rate defines the number of publishing attempts per second, and can be tuned to reproduce the desired keyword popularity.

We tested different input parameters – target KAD ID, the starting publishing instant, and the observation time – obtaining similar results, therefore in the following we will not explicitly state the values of such parameters.

Once published, a reference has a validity of 24 hours, after which it is removed from the host peer. The output of the tool is represented by the peer load, with peers sorted according to the XOR distance to the target KAD ID. We have also recorded the number of the wasted messages due to saturation.

For our load balancing scheme, we need to set the thresholds used to identify popular keywords. Looking at the load measurements, we see that the tenth replica is usually published on peers with a limited load (10%-20%). For this reason, we set $D_{\min}$ and $D_{\max}$ to 15 and 60, respectively. We performed tests with limited variations on such thresholds ($\pm 20\%$ on both $D_{\min}$ and $D_{\max}$, results not shown for space constraints), obtaining similar results.

Note that we consider an eight-bit zone with a single popular object: thanks to the KAD hash function, it is very unlikely that the KAD IDs of two popular objects are close enough to influence each other [7].

### 6.2 Results

We first validate our simulator by reproducing the basic KAD scheme, and taking snapshots of the system at different times, for different popularity of the keywords. In particular, we consider a publishing rate equal to 50, 5 and 0.5 publishing requests per second for objects with high, medium and low popularity, respectively. Figure 7 shows the results for the three cases. Thanks to the high number of peers, all the simulations have always shown the same qualitative behavior. The high and low popularity results match the corresponding ones obtained with our measurements (cf. Figs. 4 and 5).

The simulator also trackes the rate of wasted messages: for the peers close to the target, this is equal to the probability to be chosen times the publish rate (once the peer is saturated). The output of the simulator confirmed this computation: even if these results cannot be compared with the real measurements (where we had a single, always online, peer tracking the messages), they can be used in comparison to the wasted messages in case of load balancing.

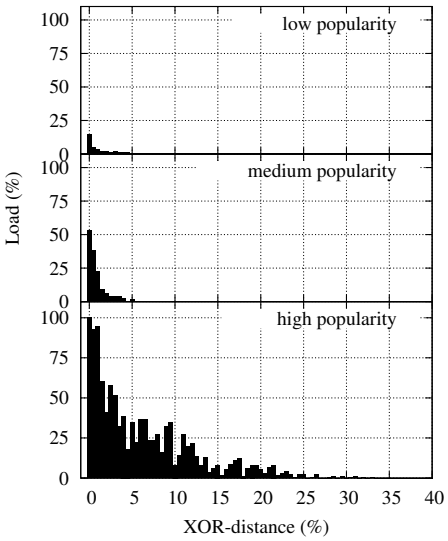



Fig. 7. Load distribution with the basic KAD scheme.

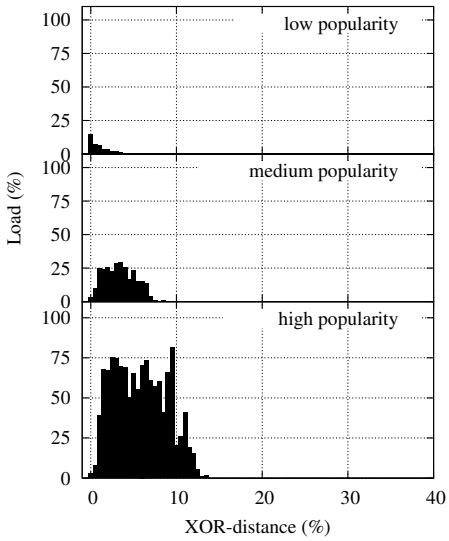



Fig. 8. Load distribution with our load balancing scheme.

We note that if we sum up the load of all the peers in the snapshot (which corresponds visually to the area under the "skyline" of the load distribution), we obtain the total number of references, all replicas included, currently stored in the system.

With the same settings used in the basic KAD scheme, we have tested our load balancing scheme. Fig. 8 shows the results for the same keyword popularities used in Fig. 7. For objects with high and medium popularity, the load balancing scheme is able to spread the references on a higher number of peers w.r.t. the basic scheme. Moreover, the total number of stored references is larger than with the basic KAD scheme (the area under the "skyline" is bigger than the corresponding ones in Fig. 7): this is due to the fact that *no publishing messages* have been discarded. Compared to the basic KAD scheme, our load balancing scheme is able to improve the reliability and the diversity of the references, since no publishing messages are lost due to overload of the host peers.

For objects with low popularity, the behavior of our mechanism remains similar to the basic KAD scheme: our load balancing solution is able to adapt to the popularity conditions and spread the load accordingly.

Figures 7 and 8 can be analyzed also under a different perspective: consider an object whose popularity varies over time, from low to high, due to a sudden increase of interest in such object. The three different popularities may represent a snapshot of the evolution of the system. In this case, we can see that our scheme is able to involve increasingly more host nodes, balancing at the same time the load among them, without losing any reference. Instead, the basic KAD scheme, even if it actually uses more host peers, shows a strong imbalance among them, which results in some lost references. If the popularity variation goes from high to low, the fact that references have an expiration time (after which they are removed from the host peers) ensures that the load on host peers far from the target will decrease.

The evaluation of the load balancing scheme needs to consider the performance of the searching phase as well. Every 30 minutes we simulate a search, i.e., we use the same candidate list building process and we send a search request following the basic KAD scheme (i.e., starting from the first candidate) and our proposed scheme (cf. procedure `Search` in Sec. 5.3). For each search, we record the number of peers that has been queried in order to obtain at least 300 references. Tab. 3 shows the performance (over 300 searches, with 95% confidence intervals not reported since they are all smaller than 1% of the measured value), in case of the basic KAD publishing scheme and our load balancing scheme[2].

TABLE 3
Mean number of queried peers during the search.

| | High popularity | | Low popularity | |
|---|---|---|---|---|
| | basic search | improved search | basic search | improved search |
| basic KAD publ. | 1.02 | 1.04 | 1.12 | 1.39 |
| load balancing | n.a. | 1.07 | n.a. | 1.23 |

We note that our improved search scheme is able to provide 300 references with a small penalty in the number of queried peers: in practice, in the worst case, 27% of the time the searching peers need to query two candidates, which are randomly chosen among the first ten. As the improved search scheme is able to improve diversity, since it may retrieve references that have not been published on the top ranked peers (due to overload), such a slight increase in the average number of queried peers seems to be a reasonable price to pay.

2. If peers publish with the load balancing scheme, they will perform the improved search, therefore the basic search is not shown in this case.

## 7 Conclusion

The popularity distribution of objects in a P2P network is highly skewed. We developed *Mistral*, a content spy to gain an overview of the content published and searched in KAD. We have reported our findings from an extensive measurement campaign on KAD, the largest currently deployed DHT. Our observations show that the publication process in KAD accounts for more than 90% of the total network control traffic. Moreover we note that the load is highly unbalanced between the peers. The peaks of load are due to very popular keywords: among them, meaningless stopwords can simply be excluded to improve the overall system performance. For keywords with popularity tied to the popularity of files, which may vary over time, load balancing is necessary to ensure a fair use of the available resources in the network. We have proposed a solution that dynamically adjusts the criteria used to select the number and the location of peers responsible for storing the references, based on their popularity. Working with KAD introduces a number of constraints to maintain, for example, backward compatibility. As such, our mechanism operates at the algorithm-level and does modify the KAD protocol and the messages.

Our simulation results showed that we can avoid the loss of object references due to saturation, thus increasing the reliability and the diversity of the resources. Furthermore, we evaluated an enhanced searching procedure, based on randomization, to exploit such increased diversity: our results indicate that the price to pay for a more efficient use of peer resources in the network (which implicitly include the content delivery phase) is arguably small.

There are a number of possible future research directions stemming form our work. For instance, the Lookup procedure can be re-implemented to increase the accuracy of the candidate list produced by KAD clients. Additionally, our load balancing mechanism can be improved if we allow protocol modifications, so as to eliminate "redundant" publish messages to infer load information.

## References


[1] F. L. Fessant, S. B. Handurukande, A.-M. Kermarrec, and L. Massoulié, "Clustering in peer-to-peer file sharing workloads," in *Proc. of IPTPS*, 2004.

[2] J. S. Otto, M. A. Sanchez, D. R. Choffnes, F. E. Bustamante, and G. Siganos, "On blind mice and the elephant – understanding the network impact of a large distributed system," in *Proc. of SIGCOMM*, 2011.

[3] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-peer informatiion system based on the XOR metric," in *Proc. of IPTPS*, 2002.

[4] Overnet, http://www.overnet.org/.

[5] E-Mule, http://www.emule-project.net/.

[6] A-Mule, http://www.amule.org/.

[7] M. Steiner, T. En-Najjary, and E. W. Biersack, "Long term study of peer behavior in the KAD DHT," *IEEE/ACM Transactions on Networking*, vol. 17, no. 5, pp. 1371–1384, October 2009.

[8] M. Steiner, D. Carra, and E. W. Biersack, "Faster content access in KAD," in *Proc. of IEEE P2P*, 2008.

[9] ——, "Evaluating and improving the content access in KAD," *Journal of Peer-to-Peer Networks and Applications*, vol. 3, no. 2, pp. 115–128, June 2010.

[10] I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer, "Beyond term indexing: A p2p framework forweb information retrieval," *Informatica*, vol. 30, pp. 153–161, 2006.

[11] F. Klemm, A. Datta, and K. Aberer, "A query-adaptive partial distributed hash table for peer-to-peer systems," in *Current Trends in Database Technology - EDBT 2004 Workshops*, 2004, pp. 506–515.

[12] K. Aberer, F. Klemm, M. Rajman, and J. Wu, "An architecture for peer-to-peer information retrieval," in *Workshop on Peer-to-Peer Information Retrieval*, 2004.

[13] T. Suel, C. Mathur, J.-W. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram, "Odissea: A peer-to-peer architecture for scalable web search and information retrieval," Department of Computer and Information Science, Polytechnic University, Brooklyn, NY, Tech. Rep. TR-CIS-2003-01, Jun. 2003.

[14] Y. Qiao and F. E. Bustamante, "Structured and unstructured overlays under the microscope - a measurement-based view of two p2p systems that people use." in *Proc. of USENIX ATC*, 2006.

[15] M. Bienkowski, M. Korzeniowski, and F. M. auf der Heide, "Dynamic load balancing in distributed hash tables," in *Proc. of IPTPS*, 2005.

[16] J. Byers, J. Considine, and M. Mitzenmacher, "Simple load balancing for distributed hash tables," in *Proc. of IPTPS*, 2003.

[17] B. Godfrey and I. Stoica, "Heterogeneity and load balance in distributed hash tables," in *Proc. of IEEE INFOCOM*, 2005.

[18] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured p2p systems," in *Proc. of IEEE INFOCOM*, 2004.

[19] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in strucured p2p systems," in *Proc. of IPTPS*, 2003.

[20] K. W. T.-T. Wu, "An efficient load balancing scheme for resilient search in kad peer to peer networks," in *Proc. of IEEE MICC*, 2009.

[21] D. Wu, Y. Tian, and N. Kam-wing, "Resilient and efficient load balancing in distributed hash tables," *Journal of Network and Computer Applications*, vol. 32, no. 1, pp. 45–60, 2009.

[22] Z. Xu and L. Bhuyan, "Effective load balancing in p2p systems," in *Proc. of IEEE CCGRID*, 2006.

[23] H.-J. Kang, E. Chan-Tin, Y. Kim, and N. Hopper, "Why Kad lookup fails," in *Proc. of IEEE P2P*, 2009.

[24] J. R. Douceur, "The Sybil attack," in *Proc. of IPTPS*, 2002.

[25] J. Dinger and H. Hartenstein, "Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration," in *Proc. of ARES*, 2006.

[26] G. Danezis, C. Lesniewski-Laas, M. Kaashoek, and R. Anderson, "Sybil-resistant DHT routing," in *European Symposium On Research In Computer Security*. Springer, 2005.

[27] T. Cholez, I. Chrisment, and O. Festor, "Evaluation of sybil attacks protection schemes in KAD," in *Proc. of AIMS*, 2009.

[28] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge Press, 2005.

[29] K. Sripanidkulchai, "The popularity of gnutella queries and its implications on scalability," in *Proc. of O'Reilly's OpenP2P*, 2001.

[30] D. Carra and E. W. Biersack, "Building a reliable P2P system out of unreliable P2P clients: The case of KAD," in *Proc. of CoNEXT*, 2007.

[31] Stop words used in Internet search engines, "http://www.link-assistant.com/seo-stop-words.html."

[32] S. Petrovic, P. Brown, and J.-L. Costeux, "Unfairness in the e-mule file sharing system," in *Proc. of ITC*, 2007.

[33] M. Steiner, E. W. Biersack, and T. En-Najjary, "Exploiting kad: Possible uses and misuses," *Computer Communication Review*, vol. 37, no. 5, 2007.

[34] L. Sheng, J. Song, X. Dong, and L. Zhou, "Emule simulator: A practical way to study the emule system," in *Proc. of ICN*, 2010.

[35] P. Wang, J. Tyra, E. Chan-Tin, T. Malchow, D. Kune, N. Hopper, and Y. Kim, "Attacking the kad network: real world evaluation and high fidelity simulation using dvn," *Security and Comm. Networks*, 2010.

[36] KAD Load Balancing Simulator, "http://profs.sci.univr.it/~carra/downloads/kadsim.tgz."

[37] KAD traces, "http://www.eurecom.fr/~btroup/kadtraces/."