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. Based on an extensive measurement campaign, we pinpoint several deficiencies of KAD in handling popular content and provide a series of improvements to address such shortcomings.
Our work reveals that keywords, which are associated to content, may become popular for two distinct reasons. First, we show that some keywords are intrinsically popular because they are common to many disparate contents: in such case we ameliorate KAD by introducing a simple mechanism that identifies *stopwords*.
Then, we focus on keyword popularity that directly relates to popular content. We design and evaluate an adaptive load balancing mechanism that is backward compatible with the original implementation of KAD. Our scheme features the following properties: *(i)* it drives the process that selects the location of peers responsible to store references to objects, based on object popularity; *(ii)* it solves problems related to saturated peers that would otherwise inflict a significant drop in the diversity of references to objects, and *(iii)* if coupled with a load-aware content search procedure, it allows for a more fair and efficient usage of peer resources.

1 Introduction

The traffic generated by peer-to-peer (P2P) applications represents a significant portion of the 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]. Simplified versions have been implemented in Mainline BitTorrent [8], Azureus [9] and Ares [10]. 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 deal with heterogeneity, primarily focusing on node churn. The dynamic nature of arrivals and departures of peers, and the resulting heterogeneous session times, represents one of the main, and better studied, characteristics of P2P networks.

- 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

Despite the vast amount of work on DHTs, little has been said about the heterogeneity in terms of *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 objects. While object popularity is determined by the number of replicas that exist, reference popularity arises for two reasons. Either the object (where the reference points to) is popular, or the reference contains a popular name or common keyword, such as “the,” “mp3,” or “dvd,” which can be found in different object names.

In this work we investigate how KAD copes with object and reference popularity. To this aim, we perform a set of measurements. 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 balance the amount of load each peer must support to store objects and references in a dynamic way that adapts to variations in content popularity. Current solutions use a *statically* preset number of peers to achieve load balancing. Instead, we propose an *adaptive* load balancing mechanism for KAD-based systems.

1.1 Main Contributions

Our work makes several contributions, ranging from the design of high performance measurement tools, to system design and evaluation. This paper builds upon a set of preliminary works [11], [12], extends their scope and offers a consolidated, holistic view of popularity management in KAD. Furthermore, we note that issues related to content popularity affects not only KAD, but many other P2P system: while the solutions presented in this paper are specific to KAD, the main ideas – e.g., exploiting content replication to enable an adaptive scheme – can be applied to other systems as well. Essentially, this work provides a reference toward the informed design of P2P systems that must support very popular content. In the following, we briefly summarize the main contributions of our work.

Measurement tools: We designed and implemented two measurement tools, a content spy called *Mistral* and an instrumented aMule client, which contribute to a better understanding of how KAD works. *Mistral* exploits for its operation the so called *Sybil attack* [13], [14], [15] and is, to the best of our knowledge, the first tool applying the *Sybil attack* to KAD.

Measurement campaign: With our 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. These results have never been observed before: in [16], for instance, the authors focus solely on query performance metrics, whereas the focus of this work is on a complete characterization of both, search and publishing traffic.

We have also studied how KAD manages popular 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. This is in contrast with previous work, e.g. [17], where the authors consider routing to be the main problem. Finally, our measurements show that many of the keywords that compose the reference names are meaningless stopwords, which constitute a substantial overhead. Stopwords have been used for decades in indexing and retrieval, but never for filtering the searches in P2P systems. As such, in this work we study means to reduce the publishing overhead without reducing the retrieval success rate of the KAD system.

Load balancing: We design a load balancing scheme that selects the peers used for storing references as a function of the object popularity. Our aim is to maintain backward compatibility: as such, our design does not alter the standard KAD protocol. The improvements we propose only modify the algorithms for search and publish executed by the peers. Despite the many papers on load balancing in P2P systems (see Sect. 2.3), our proposal is the first adaptive load balancing scheme for KAD. Moreover, we improve the content search procedure of KAD to better exploit reference replication. Our goal is to decrease the burden imposed on few peers by the current KAD implementation and spread the load more evenly.

Evaluation: We evaluate our load balancing and search schemes using a trace-driven simulator, which can reproduce realistic peer arrivals and departures; our results show that our load balancing scheme is effective in distributing the load among peers, and the search procedure is able to find objects referenced by many peers.

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 routing tables: in particular, the maintenance phase updates and rearranges each entry – which we refer to as **contacts** – of the routing table. A peer stores only a few contacts to 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 points to a peer that is offline, we refer to it as *stale*. The routing management is also responsible for replying to route requests issued by other nodes during the lookup phase (Sect. 2.2). In practice, a route request contains a target KAD ID and the desired number of contacts (usually two or four). A node that receives a route request for target T replies with contacts in its routing table that are the closest to T. The details on the process of filling the routing entries can be found in [7]. As reported in [18], routing tables contain many entries such that, on average, 2-3 hops are sufficient to reach a target.

Content management takes care of publishing the **references** to the objects a peer has, as well as retrieving the references to the objects a peer is looking for. KAD implements a two-level publishing scheme; a reference to an object comprises a **source** and W **keywords** (see Appendix A.1, Fig. 7 for an example):

- 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, when we refer to source, keyword, or target, we always mean the corresponding KAD ID. We call **publishing node** the node who owns an object and **host nodes** the nodes that have a reference to that object. When a node searches for a content, it first searches the keywords that describe the content, obtaining the pointers to different sources that contain such keywords. It then selects a source and contacts it 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 pushes *multiple* *copies* (the default value is 10) of each reference – source and keywords – on different host nodes. An **expiration time** is associated with each reference – 5 hours for a source and 24 hours for a keyword – after which the information on the host node is removed.

2.2 Content Management

Content management procedures, such as publishing and searching, use a common function called **Lookup**. Given a *target* KAD ID, the Lookup procedure builds 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 [19][20].

Initialization: The (publishing or searching) peer first retrieves from its routing table the 50 closest contacts to the target and stores them in the candidate list. The contacts are sorted by their distance, closest first. The peer sends a request to the first $\alpha = 3$ contacts, asking for β 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: 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 β contacts it has received into the candidate list in case they are not already present. Given the updated candidate list, a new route request is sent if, among the α closest to the target, there is a new contact that is closer to the target than the peer that provided that contact.

Stabilization: The Lookup procedure terminates when, for at least three seconds, no response arrives, or the responses contain contacts that are either already present in the candidate list or further away from the target than the top α candidates. At this point no new route request is sent and the list becomes *stable*.

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 accepts up to a maximum number (default set to 50,000) of references for a given keyword. The **load** is defined as the ratio between the current number of references published for a given keyword on a peer and the threshold value of 50,000 (times 100). It is important to note that, although host nodes with a load equal to 100 discard publishing message they receive, they nevertheless return an acknowledgment to the publishing nodes. As a consequence, when a node is located in a hot spot for a popular keyword, the threshold of 50,000 possible references can be reached easily and any further references (even to rare keywords) will thus be lost.

In case of search, peers send a '`search reference`' message to the first candidate in their list. If the response contains 300 references (sources), the process stops; otherwise, peers iterate through their candidates until the threshold of 300 sources is reached. Since host nodes may store up to 50,000 references for a given keyword, a reply message includes 300 references drawn randomly from the whole set stored by the peer.

2.3 Related Work

In Sect. 5 we propose essentially two solutions for dealing with popular content: the use of stopwords, and an adaptive 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 search results in P2P systems. The works in [21], [22] and [23] provide a general architecture that aims at building a full-text search engine, while in our approach we do not intend to support full-text search over the entire document; instead, our goal is to enhance the indexing process for file names in P2P systems.

The paper of Qiao and Bustamante [16] presents measurement results from a study of Gnutella and Overnet (a precursor of KAD). 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. In our work, we consider the publishing load (instead of the load due to the queries), which represents the majority of the overhead traffic.

Load balancing for DHT systems has been extensively studied in the past: here we consider the most representative works. Many solutions [24][25][26] 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* [27][28] have been devised to cope with heterogeneity of peer resources and content popularity: such schemes have a fixed number of possible peers for balancing 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 [29], which is focuses on KAD, also uses a fixed maximum number of peers for load balancing. Moreover, such a mechanism introduces new messages that require a modification to the original KAD protocol. Our solution, instead, does not change the KAD protocol.

Other works [30][31] consider the transfer of the content from overloaded peers to underloaded ones (content migration): load balancing is initiated by host nodes, and incurs in a high overhead. In our scheme, the load balancing is performed by the publishing peers, without any additional overhead with respect to the basic KAD scheme. The authors in [17] propose a load balancing scheme which is not adaptive, and does not avoid 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 to collect a vast amount of information on KAD.

Instrumented client: We have instrumented an aMule client to log the internal state of its KAD implementation. For instance, it detects the candidate list built for each published reference; or it records all the management messages (publishes, searches, replies).

Spying for Content with *Mistral*: *Mistral* exploits for its operation the well known *Sybil attack* [13], [14], [15]. It introduces a large number of "fake" peers, the *sybil*s, all controlled by us and executed on a the same machine. If positioned in a clever way in the KAD ID space, the Sybils can gain control over a fraction of the network or even over the entire network. In particular, the Sybils can intercept almost all the management traffic and provide forged replies. In our case, we have used the Sybils for recording passively the management messages (publish, search). The interested reader can find the details of Mistral design in Appendix A.2.

Other tools: During our measurements, we have used a KAD crawler, called *Blizzard*, which has been presented in [7]: we report here the main features of the crawler for the sake of clarity. *Blizzard* logs the IP address and the KAD ID of each peer it visits, and whether or not that peer responded to the crawler. The implementation of *Blizzard* is simple: the crawler starts by contacting a "seed" peer in our control to receive an initial set of peers to contact; subsequently, using breadth first search, it queries known peers to discover new peers to contact. Once the crawl terminates, results (*i.e.,* mainly log files) are written to disk.

4 Measurement Results

We now study the KAD publishing procedure by observing the traffic generated due to publish messages. First, we observe – for a period covering 24 hours – the traffic in a set of eight-bit zones with *Mistral* (Sect. 4.1). We then analyze in detail the outcome of the candidate list building process 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 show how the traffic varies over time.

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). Spying on the entire KAD ID space would be impractical. Nevertheless, the different tolerance zones are independent, and it is sufficient to focus on some sample zones to gain some insights into KAD.

Hence, our measurements focus on 20 different eight-bit zones of the KAD ID space, and cover a 24 hours period. 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 peer IDs are roughly uniformly distributed across 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 [32] 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 see *ten times* more publish messages than search messages. This result is confirmed by recent measurements using an instrumented aMule client (see Sect. 4.4). 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.

Next, we focus on two particular zones. Fig. 1 shows, the number of times a keyword publication is observed versus the ranking of the keyword for the two eight-bit zones `0xe3` and `0x8e`, where 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 $1/i^{\alpha}$). We estimate $\alpha \approx -1.63$, which is equal for all zones.

The zone `0xe3` contains the keyword "the" whereas the zone `0x8e` contains less 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 keywords are nearly equally popular. In Appendix A.3 we show similar results for the searching process.

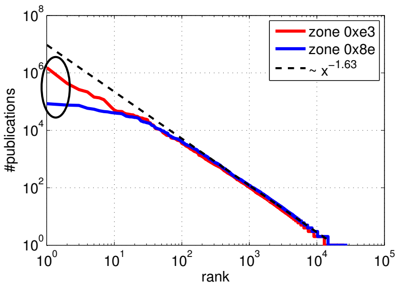

Fig. 1. The number of publications per keyword for two different zones.

4.2 Accuracy of the Candidate List

We have shown that popular keywords account for most of the load in KAD. For this reason, we need to further investigate how KAD handles popular objects: thus, we now consider the basic mechanism used for both content publishing and search, *i.e.,* how the candidate list is created.

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. 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 α ($\alpha = 3$ by default) candidates for three consecutive seconds. This means that peers focus 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 elements itself, $\mathcal{L}'$ contains the rank order of the elements as they appear in $\mathcal{L}$.

In order to evaluate the accuracy of $\mathcal{L}'$ with respect to $\mathcal{L}$, 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. The detailed results are shown in Appendix A.4. Here we report some examples of the outcome of the experiment in order to explain some interesting observations – see Table 1 (basic Lookup). In the first row, the peers returned by the basic lookup for ID A are in increasing distance from A: the closest, 2nd closest, 4-th closest, etc. peer.

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

ID	rank order for 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	rank order for β-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 peers that are very close to the target ID, the following peers are far from being among the ten closest ones.

Note that the authors in [17] impute this inaccuracy on 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 β closer contacts contained in the routing tables of other peers. By increasing the value of β, it should be possible to increase the accuracy of the candidate list. For instance, we set $\beta = 16$ and obtained the results labeled as "β-Lookup" in Table 1 (see also Fig. 9 in Appendix A.4), noticing an increased accuracy in the last positions[1].

4.3 Load Distribution

In Sect. 4.1 we have shown the measurements of the publishing and search traffic in an entire zone. It is interesting to understand how this traffic is distributed *within* a zone. Since the publishing traffic represents the majority of the traffic, we focus on it, and we consider some representative zones with popular keywords. For this experiment, we will not use the Sybils, since we study the impact of popular keywords on real 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. The collection of all the replies gives us a snapshot of the current load distribution. We consider two types of popular keywords: keywords whose popularity may change over time, such as "adele," and *static* popular keywords, *i.e.,* keywords that are usually present in the file names, such as "the" or "mp3."

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

Fig. 2 shows the results for two popular keywords ("dvdrip" and "adele"). We tested these and other keywords on different days and hours, obtaining similar results. The x-axis contains the number of bits that the different peers have in common with the target. We send a publish message and obtain a response from *all* peers that are alive in the 8-bit zone: since the KAD ID are approximately uniformly distributed over the KAD ID space, the peers with b bits in common with the target are twice the peers with $b+1$ bits in common with the target. Given a number of bits b in common with the target, we order the peers according to their XOR-distance to the target; for clarity of presentation, each bar in the figure represents the load of approximately 8-10 peers (the value is the mean load of the peers).

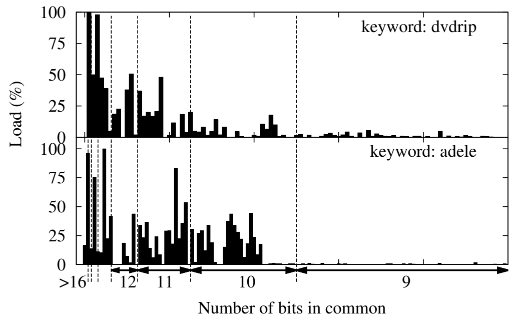

Fig. 2. Load distribution for two popular keywords.

For very popular keywords, not only the peers closest to the target are overloaded, but there are many peers far from the target that has a significant load. This is due to the inaccurate candidate list explained in Sect. 4.2.

As an example of a keyword with low popularity, in Fig. 3 we show the distribution of the load of the keyword "dexter," where the replicas are roughly concentrated around the target – number of bits in common with the target is greater than or equal to 16. Nevertheless, some replicas are published on peers with KAD IDs that share only 13 or 12 bits with the target.

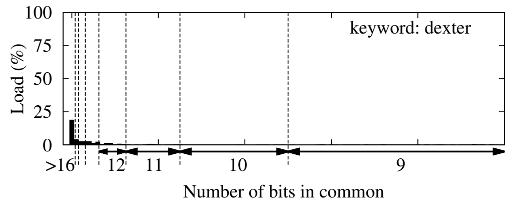

Fig. 3. Load distribution for the slightly popular keyword.

4.4 Diversity

As discussed above, for popular keywords (i) a considerable number of references are scattered over the 8-bit zone, and (ii) the closest peers to the target are overloaded. The first problem is related to the inaccuracy of the candidate list. We now focus on the second problem, i.e. the impact of overloaded peers. The references a peer can hold for a given keyword is limited to a maximum value (50,000): what happens when this limit is reached? A host peer that has reached its maximum number of references, replies positively but actually discards the reference. Therefore, all the following publishing traffic represents a waste of resources (download and upload bandwidth, processing power).

To understand how fast the limit can be reached, we show the traffic recorded by an instrumented aMule client placed close to two popular keywords. Fig. 4 provides the load (ratio between the 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.

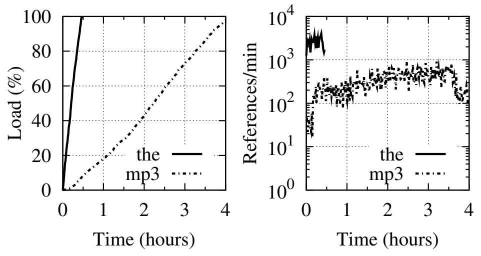

Fig. 4. Load (left) and publishing frequency (right) over time registered by our instrumented client.

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 average incoming traffic is approximately equal to 30 kbit/s.

In case of very popular keys, there is also a problem in the search phase, which we refer to as lack of *diversity*. The peer that searches will start querying the top-ranked peers in its candidate list. If it obtains in return at least 300 references, the search stops. For popular objects, the first peer in the candidate list will already have more that 300 references. Therefore, even if the references are replicated ten times, the search peers will contact only the top ranked candidate. This implies that all the peers that did not succeed to publish on the peer closest to the target will later not be contacted and therefore will not contribute their resources for the download of the these popular objects.

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 sustain a high publish and search traffic;
- references are lost when they are published on overloaded peers that discard them;
- the search phase considers mainly the peer closest to the target, without considering that references may be published on other peers.

KAD has been designed without considering skewed 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 publish process should consider its counterpart, the search process. Only a joint design will allow to address aspects such as diversity or dynamic load balancing.

5 Solutions

Keyword popularity occurs for two 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" or "mp3": most of the files contain these keywords, but such keywords do not characterize or describe the content of the files. If the system avoids publishing common keywords, the search procedure is marginally affected. In Sect. 5.1 we show the straightforward solution for the common keywords is to use stopwords. While stopwords have been known for decades in the indexing and retrieval community, they have never been used for filtering search in KAD.

In Sects. 5.2 and 5.3 we focus on file popularity, which may be variable over time. Despite the many works on load balancing in P2P systems, this is the first adaptive load balancing scheme targeted to KAD.

5.1 Common Keywords

The simplest solution for dealing with the common keywords is to ignore them. This approach requires the identification of the so called *stopwords*, i.e., words that can be filtered out with no impact on the usability of the system. Examples of such words are: "avi," "mp3," "the" or "video." A more exhaustive list can be found in Appendix A.5, Table 2.

We propose to treat all the keywords contained in Table 2 as stopwords. The eight-bit zones that contain the KAD ID of these stopwords will see a decreased traffic. For instance, zone 0xe3 contains the keyword "the" that account for 30% of the total traffic (cf. Sect. 4.1), therefore, with the introduction of stopwords, the zone will see a 30% decrease in the publishing traffic.

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 they can be included in the source code. Alternatively, the stopword list could be dynamically updated as it happens for the bootstrap nodes, where websites maintain 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 problems summarized in Sect. 4.5. We assume that stopwords are managed as discussed in the previous section, therefore the popularity of a reference is only due to the popularity of the object the reference points to.

In Sect. 4.2 we have analyzed the accuracy of the Lookup procedure. Even if the inaccuracy of the candidate list may seem a problem, this apparent drawback can be exploited 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. However, this is true only starting from the third or fourth candidate position onward, while usually the first three or four positions are mostly identical for the different publishing peers.

Considering the basic KAD scheme, we have shown that the accuracy of the last positions in the candidate list is extremely low (the tenth candidate corresponds to the thousandth peer from the target). Instead, with the *β-Lookup procedure* we have a candidate list that is still inaccurate, but with a better accuracy with respect to the basic KAD. Since it is simpler to manage the system with a candidate list that is inaccurate, but the inaccuracy is limited, we assume the use of the *β-Lookup procedure*.

We then focus on the publishing and the search procedures to perform an adaptive load balancing based on object popularity. We call our proposed schemes `LA-Publish` (load-aware publish) and `LA-Search` (load-aware search) respectively. 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. In any case, the `LA-Publish` procedure does not add any additional overhead with respect to the current KAD Publish procedure. Moreover, for non-popular objects, the `LA-Publish` procedure behaves exactly as the current KAD Publish procedure.

Given the candidate list produced by the Lookup procedure, the publishing procedure tries to publish ten replicas of the reference. The basic idea in the `LA-Publish` procedure 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 host peers closest to the target, the

Procedure `LA-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
Initialization:
    curr = 9;
    direction = backward;
    maxLoad = 80;

for $i \leftarrow 0$ to $9$ do
    contact ← candidates.get(curr);
    load ← publish(contact);
    if $curr < 10$ and $load > thresholds.get(curr)$ then
        direction = forward;
        curr = 9;
    if $curr \geq 10$ and $load > maxLoad$ then
        curr += 10;
    if direction == forward then
        curr++;
    else
        curr−−;
```

publishing peer should choose candidates progressively further from the best target.

In order to know the load of a host peer, the publishing peer needs to publish the content on that host: 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 candidate peer. If the load is below a certain threshold, the publishing peer publishes the next replica on the ninth candidate, and so on. Otherwise it considers the candidates with a rank higher than the tenth candidate peer.

As input of the `LA-Publish` procedure, 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 increase as we move towards 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 threshold increases linearly with the rank of the candidates, i.e., the threshold for the ith 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 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, the `LA-Publish` procedure considers 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 `LA-Publish` procedure represents a modification of an existing (and widely deployed) system: therefore 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 the is backward compatible since it used only the existing KAD messages.

5.3 Load Aware 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 target. 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 search 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. If it does not receive enough references, it falls back to the basic KAD Search scheme, *i.e.*, it starts from the first candidate.

For a detailed description of the load-aware search procedure (`LA-Search` procedure) please refer to Appendix A.6.

5.4 Discussion

For additional comments on different aspects related to the proposed `LA-Publish` and `LA-Search` procedures, including security considerations, parameter settings, and peer churn, please refer to Appendix A.7.

6 Numerical Results

In order to assess the effectiveness of the load-aware scheme, we take a simulation approach: an evaluation based on modified real peers is not possible since it does not allow to control some of the key parameters of our experiment: 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. Therefore we decided to implement a custom event-driven simulator [33].

Our simulator has two main characteristics: (i) the peer dynamics (arrival and departure) are realistic, since they follow the publicly available traces collected over six months from the KAD network [34]; (ii) the candidate list has the same accuracy as in the current KAD implementation, since we have used the results of the measurements presented in Sect. 4.2 to design this aspect of the simulator.

The interested reader will find the details of the simulator we developed and the settings in Appendix A.8.

We 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 with mean equal to 50, 5 and 0.5 publishing requests per second for objects with high, medium and low popularity, respectively[2]. Fig. 5 shows the results for the three cases. Thanks to the high number of peers in the used traces, 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. 2 and 3).

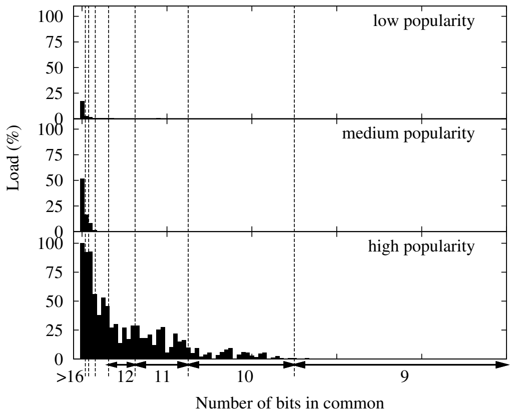

Fig. 5. Load distribution with the basic KAD scheme.

The simulator also tracks the amount of wasted messages for a given interval, set to 24 hours, i.e., equal to the period of validity of the references. The percentage of references lost with respect to the total number of published messages is equal to 35.8% in case of objects with high popularity (no publishing requests have been lost in the other cases). 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 adoption of the LA-Publish procedure.

2. The publishing rate is a random variable uniformly distributed between ±70% of the mean, as derived from measurements shown in Sect. 4.4.

With the same settings used in the basic KAD scheme, we have tested our LA-Publish procedure. Fig. 6 shows the results for the same keyword popularities used in Fig. 5. For objects with high and medium popularity, the LA-Publish scheme is able to spread the references on a higher number of peers with respect to the basic scheme. Moreover, *no publishing message* has been discarded, therefore, compared to the basic KAD scheme, the LA-Publish scheme is able to improve the diversity of the references.

For non-popular objects, the LA-Publish procedure behaves similarly to the basic KAD scheme: the LA-Publish procedure is able to adapt to the popularity conditions and spread the load accordingly.

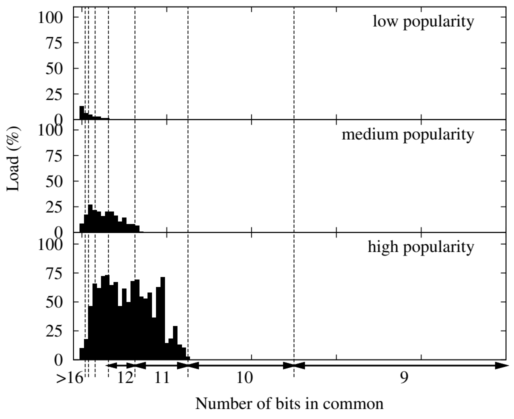

Fig. 6. Load distribution with our load balancing scheme.

Figs. 5 and 6 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 an object. The three different popularities may represent a snapshot of the evolution of the system. In this case, we can see that the LA-Publish 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 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.

We have also evaluated the performance of the searching phase: the results shows that the LA-Search scheme is able to obtain 300 references with slight increase in the average number of queried peers. The detailed description of the results can be found in Appendix A.9.

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, one of the largest DHT currently deployed. 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 among 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, 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 show that we can avoid the loss of object references due to node overload, which increases the diversity of the resources. Furthermore, we evaluated an enhanced search procedure, based on randomization, to exploit the increased diversity.

There are a number of possible future research directions stemming from our work. 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.

Acknowledgments

This work has been partially supported by the project ANR-09-VERS-014 VIPEER.

The authors would like to thank the anonymous reviewers for their invaluable comments that allowed improving the quality of our work.

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 information 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] DHT Spec., http://www.bittorrent.org/beps/bep_0005.html.

[9] Azureus/Vuze, www.azureus.sourceforge.net/.

[10] Ares, www.ares.net/.

[11] M. Steiner, W. Effelsberg, T. En-Najjary, and E. W. Biersack, "Load reduction in the KAD peer-to-peer system," in *Proc. of DBISP2P*, 2007.

[12] D. Carra, M. Steiner, and P. Michiardi, "Adaptive load balancing in KAD," in *Proc. of IEEE P2P*, 2011.

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

[14] 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.

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

[16] 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.

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

[18] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed DHT," in *Proc. of IEEE Infocom*, 2006.

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

[20] ——, "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.

[21] 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.

[22] 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.

[23] 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," Polytechnic University, Brooklyn, NY, Tech. Rep. TR-CIS-2003-01, Jun. 2003.

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

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

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

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

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

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

[30] 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.

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

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

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

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

[35] T. Cholez, I. Chrisment, and O. Festor, "Efficient DHT attack mitigation through peers' ID distribution," in *Proc. of HotP2P*, 2010.

[36] ——, "Evaluation of sybil attacks protection schemes in KAD," in *Proc. of AIMS*, 2009.

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

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

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

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

[41] 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.

Damiano Carra received his Laurea in Telecommunication Engineering from Politecnico di Milano, and his Ph.D. in Computer Science from University of Trento. He is currently an Assistant Professor in the Computer Science Department at University of Verona. His research interests include modeling and performance evaluation of peer-to-peer networks and distributed systems.

Moritz Steiner received the M.S. degree (Diplom) in computer science from the University of Mannheim, Germany in 2005, and the Ph.D. degree in computer networks from Telecom ParisTech, France and the University of Mannheim in 2008. Currently he is a Member of Technical Staff at Bell-Labs located in Murray Hill, NJ. His research interests and project activities are in the areas of analysis and design of peer-to-peer networks and cloud computing.

Pietro Michiardi holds a Ph.D. in Computer Science from Telecom ParisTech, and a M.S. in Electrical Engineering from Politecnico di Torino. Currently, Pietro is an Assistant Professor of Computer Science at Eurecom, in Sophia Antipolis, France. Pietro currently leads the Distributed System Group, which blends theory and system research focusing on large-scale distributed systems (including data processing and data storage), and scalable algorithm design to mine massive amounts of data.

Ernst W. Biersack studied Computer Science at the Technische Universität München and at the University of North Carolina at Chapel Hill. He received his Dipl. Inf. (M.S.) and Dr. rer. nat. (Ph.D.) degrees in Computer Science from the Technische Universität München, Munich, Germany, and his Habilitation à Diriger des Recherches from the University of Nice, France.

From March 1989 to February 1992 he was a Member of Technical Staff with the Computer Communications Research Group of Bell Communications Research, Morristown, US.

Since March 1992 he has been a Professor in Telecommunications at Eurecom, in Sophia Antipolis, France. His current research is on Peer-to-Peer Systems for Backup and Storage as well as Gaming Applications and Network Tomography.

Wolfgang Effelsberg received a Dr.-Ing. degree in Computer Science in 1981 from the Technische Universität Darmstadt, Germany. From 1981 to 1984 he was first an Assistant Professor at the University of Arizona in Tucson, then a Post-Doctoral Fellow at IBM Research in San Jose, California. From 1984 to 1989 he was a member of the IBM European Networking Center in Heidelberg. In 1989, he joined the University of Mannheim as a full professor of computer science.

He is a member of the editorial board of ACM Transactions on Multimedia Computing, Communications and Applications, and he serves on the program committee for many national and international conferences.

Taoufik En-Najjary received the B.S. degree in applied mathematics from Mohamed Ben Abdellah University, Fez, Morocco, in 2000, and the M.S. degree in statistics from University Pierre & Marie Curie, Paris VI, France, in 2001. In October 2001, he joined France Télécom R&D as a Research Engineer working on voice conversion for speech synthesis systems. He received the Ph.D. degree in signal processing and telecommunication in April 2005 from University Rennes I, France. From January 2005 to September 2010 he has been working as a postdoc at Eurecom, Sophia Antipolis, France.

Since October 2010 he is a research engineer at Orange Labs in Paris. His research interests include green networking, data mining and Internet traffic analysis.

APPENDIX A

A.1 Two-level publishing scheme

KAD implements a two-level publishing scheme: a reference to an object comprises a **source** and W **keywords**. In Fig. 7 we show an example where the file ("the matrix") has to keywords.

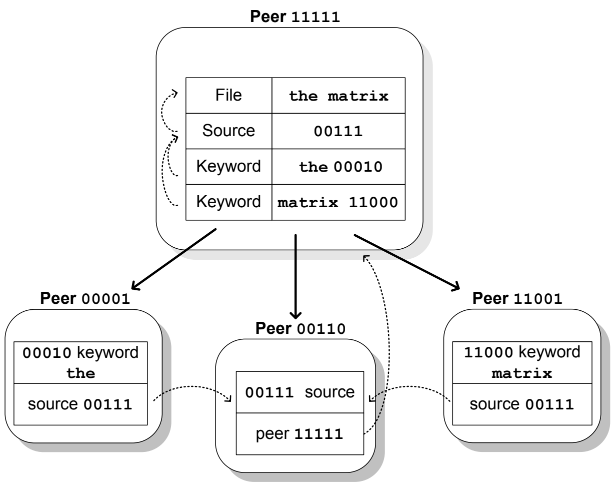

Fig. 7. Sketch of the 2-level publishing scheme

The source, whose KAD ID is obtained by hashing the content of the object (e.g., the file "the matrix" in the figure), contains information about the object and the pointer to the publishing node (e.g., peer 11111 in the figure).

The keywords, whose KAD IDs are obtained by hashing the individual keywords of the object name (e.g. the keywords "the" and "matrix" in the figure), contain (some) information about the object and the pointer to the source.

A.2 Details on Mistral

Mistral introduces many "fake" peers, all controlled by us and executed on a the same machine. The fact that all Sybils are executed on the same machine has the advantage that data collection is much easier.

Mistral creates a large number of Sybil peers in the KAD network and propagates information about them in order to "poison" the routing tables of the legitimate peers. To do so, we first crawl KAD using *Blizzard* to learn about the peers in the network. Next, Mistral sends `hello` messages to the peers we have learned about. A `hello` message is 120 bit long and includes the KAD ID of the sender which can be arbitrarily chosen. *Mistral* chooses at random the first 24 bit of the sender KAD ID put in a `hello` message, while the 96 remaining bits are always the same, thereby creating 2^{24} Sybils.

The routing queries that reach a Sybil node are always answered with the KAD IDs of other Sybils. These KAD IDs are closer to the target than the receiver of the query, giving the querying peer the impression of approaching the target. Once the requester gets close enough to the target, it will stop looking for closer peer and will query a Sybil for the content itself. Our Sybil will store the search request and return a fake source entry that points to our machine. As a consequence, when the real peer tries to start to download it will not succeed.

With our tool, we retrieve routing and search requests together with publish request messages. Publish requests are especially interesting since they are much more frequent than search requests. Whereas search requests can only be issued 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. 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 reply to a publish request contains the load of the host peer. The Sybils will always answer with a very low load to keep 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 by restricting the returned KAD IDs to a certain prefix. We insert 65,356 ($= 2^{16}$) distinct Sybils into a zone by varying the bits 9-24 of the KAD ID 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 with eMule version 0.49a and aMule version 2.2.1, launching a Sybil attack is more difficult: A peer will ignore multiple KAD IDs pointing to the same IP address. Moreover, other mechanisms have been proposed in [35] to limit the Sybil attack. Therefore, the only way to perform today a measurement with *Mistral* is to use a pool of different machines: for example, using a subset of the Planetlab machines can be effective in staging a Sybil attack [36]. In Sect. 4.4 we perform a set of simple tests, using the current version of eMule and aMule, which show that the main results obtained with *Mistral* are still valid.

A.3 Search Traffic

Fig. 8 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.

A.4 Additional Results on the Accuracy of the Candidate List

Fig. 9 shows the mean rank values of the peers in $\mathcal{L}'$, along with the 95% confidence interval (obtained with

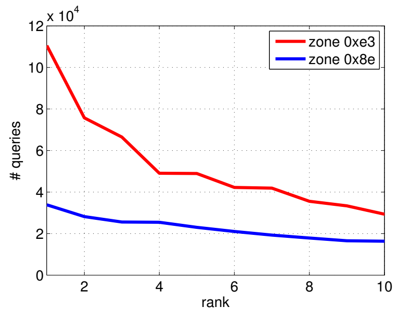

Fig. 8. The number of queries received by the Sybils for two different zones.

approximately 30 independent experiments). The figure shows also the ideal situation, where $\mathcal{L}'$ contains the ten closest peers. The graph shows that the Lookup procedure, except for the first few peers, returns peer far from being among the ten closest ones.

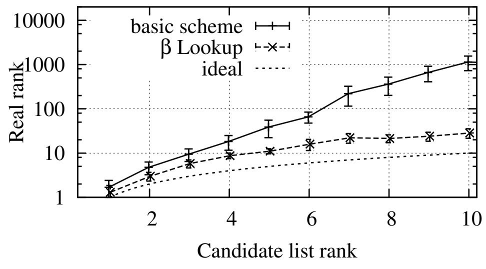

Fig. 9. Mean rank values of the elements in $\mathcal{L}'$ ($\beta = 16$).

Note the results labeled as "β-Lookup", where the accuracy of the list in the last positions is increased; in particular the last few elements are approximately within two or three times the best possible candidates.

A.5 List of stopwords

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) [37]. The number of peers on which a stopword is published (second and fifth column), as well as the number of files containing the stopword (third and sixth column), have been determined by first crawling the peers around the stopword with *Blizzard* and then by querying all those peers for the stopword.

A.6 Load Aware 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 target. Such peers are rarely used, with a consequent decrease in 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.

```
Procedure LA-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 LA-Search procedure introduces some randomness in the search process. Given the candidate list, instead of considering the first candidate, the searching node picks 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.

In the LA-Search 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 the basic KAD Search 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 [38] 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 `LA-Search` procedure works also in case of adoption of `LA-Publish` procedure: the references to popular objects will be scattered around the target, and a random search scheme will be able to easily find them.

A.7 Additional Discussion Related to the Proposed Solution

We now consider different aspects related to the proposed `LA-Publish` and `LA-Search` procedures, 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 evaluated the accuracy of the candidate list, we have shown the results up to the tenth position. The `LA-Publish` procedure considers the positions with a lower rank. In case of β-Lookup procedure 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 the `LA-Publish` procedure in a instrumented aMule client have shown that this assumption is reasonable.

Improving accuracy: The `LA-Publish` and `LA-Search` procedures rely on the fact that the candidate list is accurate in the first few positions and then becomes progressively inaccurate in the other positions. This is specific to the implementation of the Lookup procedure in KAD (both in the basic implementation and in the β-Lookup). One may ask what would happen in case of a completely new 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 β-Lookup procedure. By adopting this approach, The `LA-Publish` and `LA-Search` procedures remain sufficiently general, yet maintaining their 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 `LA-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 shown in Sect. 4.3 to set the thresholds. As for the `LA-Search` procedure, there are two parameters: the number of random attempts and the maximum rank in the candidate list. In Sect. 6 we study them in a synthetic environment. As 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 the `LA-Publish` procedure. 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 [39] is sufficient to weaken the attacks to the `LA-Publish` procedure. As for the eclipse attack, since the `LA-Publish` procedure 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 (but the candidate list does 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 peers is not a big problem since they have a low load.

A.8 Simulator Description and Settings

For the evaluation of the load balancing scheme (i.e., the `LA-Publish` procedure), 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 must have full control of these two aspects in a simulator: we have considered the few available KAD simulators [40][41], and none of them provides such control. For this reason we decided to implement a custom event-driven simulator [33].

The peer arrivals and departures follow the publicly available traces collected over six months from the KAD network [34]: 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 Sect. 5.4 (paragraph "Improving Accuracy") with the help of the measurements presented in Sect. 4.2. For the basic KAD scheme and the `LA-Publish` scheme, we have used the results shown in Fig. 9, labeled as "basic scheme" and "β-Lookup," respectively.

Besides the peer availability matrix, the inputs to 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 parameter "starting publishing instant" we can choose when the publishing process starts (the publishing process continues for an interval equal to "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. We have also recorded the number of the wasted messages due to saturation.

For the `LA-Publish` 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].

A.9 Load Aware Content Search Results

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 `LA-Search` scheme. 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 the `LA-Search` scheme. Note that if peers publish with the load balancing scheme, they will perform the `LA-Search` procedure, therefore the basic search is not shown in this case.

TABLE 3
Mean number of peers queried during a search.

	High popularity		Low popularity	
	basic search	`LA-Search`	basic search	`LA-Search`
basic KAD publ.	1.02	1.04	1.12	1.39
`LA-Publish`	n.a.	1.07	n.a.	1.23

We note that the `LA-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 `LA-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.