Adaptive Load Balancing in KAD

Damiano Carra
University of Verona
Verona, Italy
damiano.carra@univr.it

Moritz Steiner
Bell Labs
Alcatel-Lucent, USA
moritz@bell-labs.com

Pietro Michiardi
Eurecom
Sophia Antipolis, France
pietro.michiardi@eurecom.fr

***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, which pinpoints several deficiencies of the present KAD design in handling popular objects, we set off to 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 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.**

I. Introduction

The design of large scale distributed systems, such as Peer-to-Peer (P2P) networks, poses many challenges due to the heterogeneity of its components. In particular, many systems based on distributed hash tables (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.

In addition to churn, the research community has also considered other types of heterogeneity, such as resource availability (e.g., bandwidth capacity): this led to the design of many load balancing schemes that target a fair contribution of all peers, with their resources, to the P2P network.

In this work, we focus on the heterogeneity in object (*i.e.*, content) *popularity*. The crux in handling heterogeneous popularity is to design a load balancing mechanism that tailors the amount of load each peer must support in an adaptive way. However, current proposed solutions usually consider a *statically* pre-set number of peers to use for load balancing. Instead, to support popularity dynamics, the system should determine the number of peers dedicated to a particular object according to its current popularity.

The endeavor of this work is to realize an *adaptive* load balancing mechanism for a widely deployed DHT system, namely KAD. While many DHT systems have been proposed in the literature, only few of them have been practically implemented and are being used by millions of users. As such, focusing on KAD offers a unique opportunity to understand in detail the effects of object popularity on an operational network, and measure the inadequacy and the weaknesses of its current design.

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

- We establish an extensive measurement campaign to evaluate how KAD manages popular objects. Our results indicate that a large fraction of references to popular objects (pointing to the peers storing such objects) are lost due to peer saturation. Moreover, our measurements identify the KAD lookup procedure as one of the main culprit of the load imbalance that occurs when objects are placed and retrieved;
- We design a load balancing scheme for KAD which adapts the number of peers used for storing object references based on 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 object 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 object 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 under the adaptivity and the exploitation of the available replicas can be generalized and used in other systems. The remainder of the paper in organized as follows. In Sec. II we provide some background on KAD, on the content management and we discuss the related work. In Sec. III we provide a set of measurement results that give us insights on the current implementation and performance of KAD in case of popular keywords. The weaknesses highlighted in this section will help us in design an adaptive load balancing scheme which we present in Sec. IV. We evaluate the proposed scheme in Sec. V, and we conclude in Sec. VI.

II. Background and Related Work

A. The Kademlia DHT System

KAD is a DHT protocol based on the Kademlia framework [9]. 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. II-B). 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 [17].

Content management takes care of publishing the **references** to the objects the 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.

Hereinafter, 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.

B. 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 [14][15].

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 ordered by their distance, the 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 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 β returned contacts in 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 α 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 α candidates. At this point no new route request is sent and the list becomes *stable*.

Note that, in every step of the Lookup procedure, only the peers whose KAD ID 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 objects with the requested reference, the process stops; otherwise, the peer iterates through the candidates until it has reached 300 objects. Note that a host node may have up to 50,000 references for a given keyword or source: in the reply, the host node will select randomly 300 references.

C. Related Work

Load balancing for DHT systems has been extensively studied in the past: in this section we focus on few representative works. Many solutions [2][3][6] 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.

In order to cope with heterogeneity (of the peer resources, of the object popularity), there exist many works based on the concept of *virtual servers* [5][11]: such schemes have a fixed number of possible peers to be used to balance the load, while our solution varies the number of storing peers based on the popularity of the objects. The work in [18], which is focused on KAD, is also based on a maximum number of peers that can be used for load balancing. Moreover, they introduce a set of new messages that changes the protocol. Our solution, instead, is based solely on the currently available messages on KAD, without changing the protocol.

Other works [20][21] 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 the basic KAD scheme. The authors in [8] – another work specific to KAD – propose a load balancing scheme which is not adaptive, and does not avoid the loss of information.

III. Motivations

The aim of a DHT is to uniformly map objects to the key space. Nevertheless, objects are not all equal. Depending on the popularity of the objects, the publishing or searching traffic may vary significantly. In this section we investigate the impact of the heterogeneity of the object popularity on KAD (in the implementation of the eMule / aMule software) through an extensive measurement campaign. Armed with the insights we gain on the inner design choices of KAD, we motivate the need for an adaptive load balancing mechanism and improved content searching.

A. Load Distribution

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? To understand this aspect, we have collected the values of the load for different 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).

For the measurement campaign, we used a modified version of the KAD crawler Blizzard [17]. Given a keyword, Blizzard extracts the eight most significant bits and iteratively crawls the corresponding zone. In other words, the crawler considers only the tolerance zone that contains all the possible host peers on which a keyword can be published. The output of Blizzard is the list of all the peers in the eight-bit zone that are alive (stale contacts are removed): at this point we send a publish message to all peers, obtaining as a response the load from each of them. We collect the replies and we order them according to the XOR-distance to the KAD ID of the keyword, obtaining a snapshot of the current load distribution.

Figure 1 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 $d = 2^{116} - 1$, which in percentage becomes $d/d_{max} = 6.25\%$.

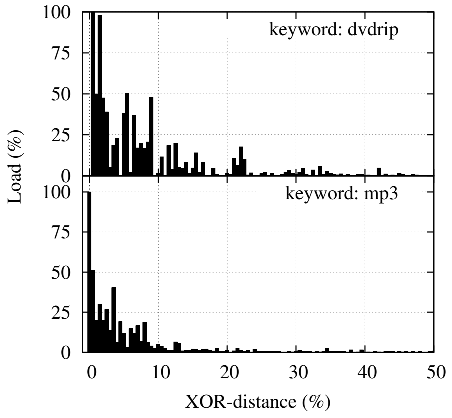

Fig. 1. Load distribution for two popular keywords.

For clarity of presentation, we have divided the x-axis into bins, therefore 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 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 object with low popularity, in Fig. 2 we show the distribution of the load of the keyword "dexter," where the replicas are roughly concentrated around the target.

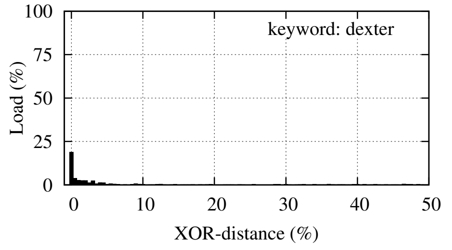

Fig. 2. Load distribution for the slightly popular keyword.

At a first glance (cf. Figs. 1 and 2), it appears that KAD inherently distributes the load among increasingly distant peers when objects are popular. Unfortunately, as we will see in Sec. III-C, this effect has not been included *intentionally* in the design of KAD, but derives from an imperfect Lookup procedure. More to the point, despite the spread of the references in case of popular objects, the peers closest to the target are overloaded and need to handle more management traffic (publishing and searching messages) than other peers. In order to quantify the amount of management traffic hitting a hot spot, we perform the following additional measurement study. We assign an instrumented aMule client a KAD ID close to the KAD ID of a popular keyword, and we register all the incoming traffic. The management traffic amounts to 3.5 publishing messages per second, and 0.3 searching messages per second. The corresponding amount of incoming traffic is approximately equal to 30 kbit/s. Even if this value seems affordable by most 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. Besides the absolute value of such management traffic, as we will show next, the publishing traffic may result in wasted resources.

B. Reliability and Diversity

The publishing peer, for reliability reasons, publishes ten replicas of each object. In case of popular objects, it may happen that the host peer has reached its maximum number of references: in this case, the host peer replies positively to the publishing request, but actually discards the reference. From the publishing node 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 [4] for a detailed evaluation of the impact of the number of replicas on the reliability.

From the host peer point of view, once the maximum number of objects is reached, all the following publishing traffic represents a waste of resources, such as the download's bandwidth for receiving the messages, the processing power for processing them, and the upload's bandwidth for replying.

With the instrumented client used to record the traffic (cf. Sect. III-A) we have monitored the publishing messages over time for two popular keywords. Fig. 3 shows the load 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 to many files.

Our measurements show that after only few tens of minutes after joining the system the host peer is saturated; upon saturation, all the publishing messages represent wasted traffic: due to the limit in the number of stored objects, a peer in a hot spot is not able to record all the published objects, decreasing the reliability. However, reliability is not the unique concern that affects KAD: 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 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.

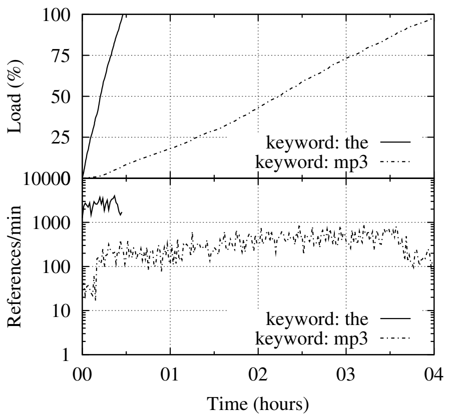

Fig. 3. Load and publishing frequency over time registered by our instrumented client.

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 and it does not work using its full service capacity.

C. Accuracy of the Candidate List

In case of popular objects, we have shown that the references may be spread over a wide portion of the KAD ID space. For instance, in Fig. 1, 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.

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 does not receive any contact closer than the top α ($\alpha = 3$ 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 shares the first 8 bits with the KAD ID of a given target; $\mathcal{L}$ is ordered according to the XOR-distance to the target, closer 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 $\mathcal{L}$. For instance, given $\mathcal{L} = \{p_1, p_2, p_3, p_4, p_5\}$, $\mathcal{L}'$ can be $\{2, 3, 5\}$, which means that $\mathcal{L}'$ contains the elements p_2, p_3 and p_5.

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 2 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 I (basic Lookup): for a given row, we show the index of $\mathcal{L}'$.

TABLE I
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. 4 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 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.

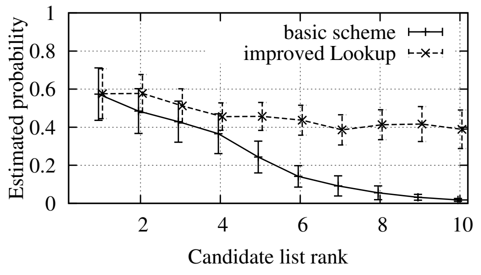

Fig. 4. Estimated probability for building $\mathcal{L}'$.

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 [8] impute the inaccuracy to the routing management, while our experience indicates 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 probabilities p_i labeled as "improved Lookup" in Fig. 4 – examples of the candidate lists can be found in Table I (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.

[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; as we said, the Lookup procedure would need a complete redesign, which is out of the scope of this paper.

D. Summary of the Motivation

The set of measurements campaigns have highlighted different issues on the current KAD procedures related to content publishing and content searching. In case of popular objects,

- some peers must support a management traffic that decreases the available bandwidth, which represents a mistreatment of peers residing in a hot spot;
- many references are lost, since they are published on overloaded peers which discard them: as such, diversity decreases;
- the search phase considers only 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, as compared to the whole set of peers hosting the content, which imply increased delays.

In practice, KAD has been designed without considering 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 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.

IV. Adaptive Publishing Scheme

In the previous section we have highlighted the main characteristics and weaknesses of the current KAD procedures: the Lookup procedure – responsible for the candidate list creation – and the publishing and searching procedures. Even if the inaccuracy of the candidate list may seem a problem, it actually represents 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 - fourth position onward, while usually the first three - 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. 4). This can be obtained by simply modifying one of the parameters. We then focus on the publishing and the search procedures to perform an 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.

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
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 - curr%10);
    if direction == forward then
        curr++;
    else
        curr−−;
```

A. Content Publishing

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 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 procedure `Publish` shows the details of our solution. As input, we provide a vector of thresholds used to identify if 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 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 and onward. Note that, if the load is above 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 procedure `Publish` has an extremely simple form thanks to the specific way in which the candidate list is built. In Sec. IV-C 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.

B. 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 such 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 among the first ten candidates. If the answer contains 300 references, the process stops. Otherwise, the searching node needs to pick another candidate. The procedure `Search` 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 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 the candidate is just arrived and it had little time to record the references. In case of non-popular object, this process results in a overhead. We believe that, thanks to the gain in terms of diversity and load balancing in case of popular objects, such overhead is a fair price that can be paid: measurement studies [10] have shown that few popular files

Procedure `Search`

```
Data: list: candidates /* peers ordered by their
    distance to target                          */
Data: list: references /* obtained refs         */
Data: int: maxRandomTentatives
Data: int: maxIndex
Initialization:
    maxRandomTentatives = 2;
    maxIndex = 10;
    references = {∅};

while references.size() < 300 and candidates not empty do
    if maxRandomTentatives > 0 then
        contact ← candidates.getRandom(maxIndex);
        references.add(search(contact));
        maxRandomTentatives−−;
    else
        contact ← candidates.getFirst();
        references.add(search(contact));
    candidates.remove(contact);
```

(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.

C. 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 parameter β 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 object, there is an expiration time associated to it, after which the object 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, so that to drive the peer candidate choice. We will evaluate this enhancement as a future work.

Parameter setting: The procedure `Publish` 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 issues; 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 should 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. III to set the thresholds. As for the procedure `Search`, there are two parameters: the number of random tentatives and the maximum rank in the candidate list. As a future work we plan to perform a measurement campaign to evaluate the impact of such parameters in real environments. In Sec. V-B we study them in a synthetic environment.

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 object KAD ID, in both cases the effect would be minimal. If the malicious peer is far from the object KAD ID (i.e., it tries to be in the ninth or tenth position in 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 to a Sybil attack: therefore, any solution that prevents a Sybil attack [13] 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 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 problem, since they have a low load.

V. 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 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.

A. Simulator Description and Settings

For the evaluation of the load balancing scheme, we need essentially two key ingredients: (i) the peer dynamics (arrival and departure) should be realistic, and (ii) the candidate list should have the same accuracy of the current KAD implementation. We should have full control on these two aspects in a simulator: we have considered the few available KAD simulators [12][19] and none of them provides such control. For this reason we decided to implement a custom event driven simulator [7].

The peer arrivals and departures follow the publicly available traces collected over six months from the KAD network [1]: 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. IV-C, with the help of the measurements presented in Sec. III-C. For the basic KAD scheme and our load balancing scheme, we have used the results shown in Fig. 4 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 a time equal to 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 hereinafter we will not explicitly state the values of such parameters.

Once published, an object 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 ordered 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 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 a 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 [16].

B. 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 5 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 measurements (cf. Figs. 1 and 2).

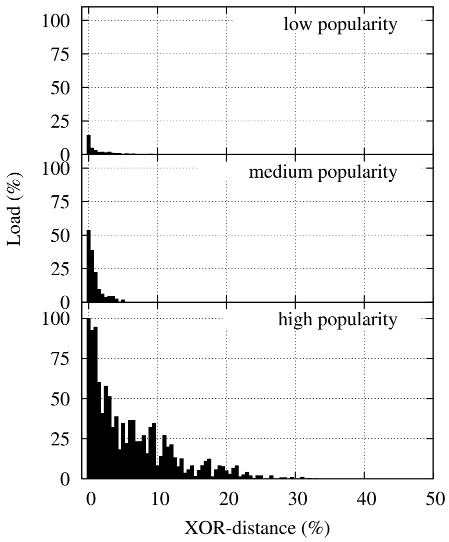

Fig. 5. Load distribution with the basic KAD scheme.

The simulator has also registered 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 registering the messages), they can be used in comparison to the wasted messages in case of load balancing.

We note that, if we sum 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 the total number of references, all replicas included, currently stored in the system.

With the same settings used in basic KAD scheme, we have tested our load balancing scheme. Fig. 6 shows the results for the same keyword popularities used in Fig. 5. In case of 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 the basic KAD scheme (the area under the "skyline" is bigger than the corresponding ones in Fig. 5): this is due to the fact that *no publishing messages* have been discarded. Therefore, compared to the basic KAD scheme, our load balancing mechanism 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.

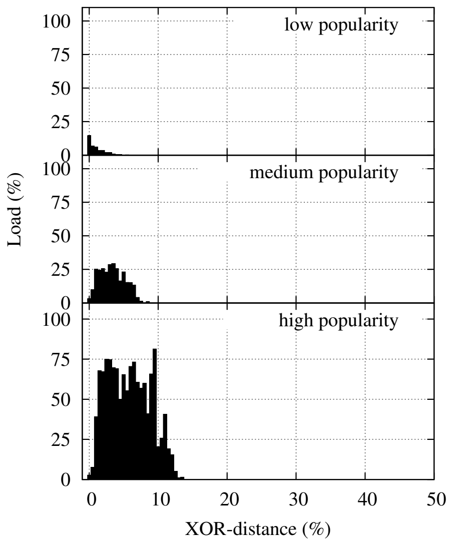

Fig. 6. Load distribution with our load balancing scheme.

Figures 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 object. The three different popularities may represent a snapshot of the evolution of the system. In this case, we can see 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. IV-B). For each search, we record the number of peers that has been queried in order to obtain at least 300 references. Tab. II shows the average of such performance index (over 300 searches, with 95% confidence intervals not reported since they are all smaller than 1% of the measured value), in case of basic KAD publishing scheme and our load balancing scheme[2].

TABLE II
AVERAGE NUMBER OF QUERIED PEERS DURING THE SEARCH PHASE.

	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 slight increase in the average number of queried peers seems a reasonable price to pay.

VI. Conclusion

The popularity distribution of objects in a P2P network is highly skewed. Therefore, load balancing is necessary to ensure a fair use of the available resources in the network. In this work, we have proposed a solution that dynamically adjusts the criteria used to select the number and the location of peers responsible for storing objects, based on their popularity. We have focused on a production system – namely Kademlia, as implemented in the aMule / eMule clients – which introduces a number of constraints to the design of an adaptive load balancing scheme in order to maintain backward compatibility. In particular, we have modified the algorithms used by the clients, without modifying the protocol and the messages.

The design of the proposed solution has been driven by a detailed measurement campaign whose aim was to highlight how the current implementation of the system deals with popular objects. Once identified the weaknesses and the available mechanisms that can be exploited in the current design of KAD, we have designed and evaluated a load balancing mechanism that copes with heterogeneous object popularity and reference diversity. Our results, based on a trace-driven simulation approach, showed that our scheme avoids the loss of object references due to saturation, thus increasing the reliability and the diversity of the resources. Furthermore, we also evaluated an enhanced searching procedure, based on randomization, to exploit such increased diversity: our results indicated 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 research directions that can be followed as future work. The fact that KAD is used by millions of users, makes available a set of data that can be used to redesign part of the system. For instance, the Lookup procedure can be re-implemented considering the approach taken by the crawler used for our experiments, so that to increase the accuracy of the candidate list. Another example is the introduction of a new set of messages that allow querying peers' load, without the necessity to force a publish procedure.

[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.

Acknowledgments

The authors would like to thank Ernst Biersack for the valuable discussions in the early stage of this work. This research has been partially supported by the French ANR-VERSO projects PROSE and VIPEER.

References

[1] http://www.eurecom.fr/~btroup/kadtraces/.
[2] M. Bienkowski, M. Korzeniowski, and F. Meyer auf der Heide. Dynamic load balancing in distributed hash tables. In *Proc. of IPTPS*, 2005.
[3] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *Proc. of IPTPS*, 2003.
[4] 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.
[5] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. In *Proc. of IEEE INFOCOM*, 2004.
[6] B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *Proc. of IEEE INFOCOM*, 2005.
[7] KAD Load Balancing Simulator. http://profs.sci.univr.it/~carra/downloads/kadsim.tgz.
[8] H.-J. Kang, E. Chan-Tin, Y. Kim, and N. Hopper. Why Kad lookup fails. In *Proc. of IEEE P2P*, 2009.
[9] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-peer information system based on the XOR metric. In *Proc. of IPTPS*, 2002.
[10] S. Petrovic, P. Brown, and J.-L. Costeux. Unfairness in the e-mule file sharing system. In *Proc. of ITC*, 2007.
[11] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in strucured p2p systems. In *Proc. of IPTPS*, 2003.
[12] L. Sheng, J. Song, X. Dong, and L. Zhou. Emule simulator: A practical way to study the emule system. In *Proc. of ICN*, 2010.
[13] M. Steiner, E. W. Biersack, and T. En-Najjary. Exploiting kad: Possible uses and misuses. *Computer Communication Review*, 37(5), 2007.
[14] M. Steiner, D. Carra, and E. W. Biersack. Faster content access in KAD. In *Proc. of IEEE P2P*, 2008.
[15] M. Steiner, D. Carra, and E. W. Biersack. Evaluating and improving the content access in KAD. *Journal of Peer-to-Peer Networks and Applications*, 3(2):115–128, June 2010.
[16] 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.
[17] M. Steiner, T. En-Najjary, and E. W. Biersack. Long term study of peer behavior in the KAD DHT. *IEEE/ACM Transactions on Networking*, 17(5):1371–1384, October 2009.
[18] K. Wang T.-T. Wu. An efficient load balancing scheme for resilient search in kad peer to peer networks. In *Proc. of IEEE MICC*, 2009.
[19] P. Wang, J. Tyra, E. Chan-Tin, T. Malchow, D.F. Kune, N. Hopper, and Y. Kim. Attacking the kad network: real world evaluation and high fidelity simulation using dvn. *Security and Comm. Networks*, 2010.
[20] D. Wu, Y. Tian, and N. Kam-wing. Resilient and efficient load balancing in distributed hash tables. *Journal of Network and Computer Applications*, 32(1):45–60, 2009.
[21] Z. Xu and L. Bhuyan. Effective load balancing in p2p systems. In *Proc. of IEEE CCGRID*, 2006.