Understanding Lookup Performance Deficiencies in the KAD Network

Yunzhao Li, Don Gruenbacher, Caterina Scoglio
Kansas State University
{yzli,grue,caterina}@ksu.edu

***Abstract*—The KAD network is one of the largest distributed hash tables utilized by the popular P2P file-sharing systems: aMule and eMule. However, its lookup performance is recognized as inefficient. In this work, based on the fact that peers publish and retrieve information with the help of their routing tables and publishing tables, we evaluate this inefficiency problem via a series of real world tests. Our measurements show that even though the maintenance policy of routing tables is well designed, the current refresh scheme of the publishing table and the selfishness of publishing peers cause the poor lookup performance. To mitigate this problem, we propose there different modifications and analyze their advantages and weaknesses.**

***Index Terms*—P2P, KAD, DHT, PlanetLab, Measurement**

I. Introduction

Peer-to-Peer (P2P) file-sharing systems still generate a large amount of the traffic payload on the current Internet [1]. To publishing and retrieving content efficiently, structured methods based on a distributed hash table (DHT) technology are proposed [2], [3], [4], [5]. In reality, there is one DHT-based network being widely deployed in P2P file-sharing systems aMule/eMule [6], [7] – the KAD network. According to the work of Steiner et al. [8], the size of the KAD network is over one million online users. In the KAD network, content publishing and retrieving need the help of publishing peers and published peers. Publishing peers are the peers with shareable content who publish the content information onto other peers, and published peers are those peers in charge of maintaining the published information. Each Publishing peer uses its routing table to discover the corresponding published peers, and then inserts the content location information into the publishing tables of these published peers. Each user can also use its routing table to discover published peers, and then retrieve needed information from their publishing tables.

Our previous measurement study [9] showed that the lookup efficiency of the KAD network was low. In contrast to other searching mechanisms (e.g., the Source-Exchange method and the Passive-Exchange method) employed in eMule/aMule, little useful content location information can be retrieved from the KAD network. Kang et al. [10] also discovered the poor lookup performance in the KAD network. However, their thinking about high similarity among routing tables causing the poor lookup performance is inconvincible.

In this work, we determine the reasons for the low lookup performance by running multiple measurement tests. Previous measurement studies [11], [12], [13], [14] usually tested the KAD network from perspective of a single client. Due to the huge size of the KAD network, they usually measured a specific aspect of the whole KAD network. In contrast, we develop a distributed measurement framework employing multiple test nodes running on the PlanetLab testbed [15]. The entire ID space of the KAD network are uniformly separated into multiple parts, each of which are measured by an individual PlanetLab test node. Therefore, different from previous works, our measurements provide a picture of the entire KAD network. During the measurements, the routing tables of more than ten thousand peers have been crawled and analyzed. more than three million source-location information from the publishing tables of multiple peers have been retrieved and contacted . Based on these measurements, we have the following contributions.

1) We show that the maintenance policy of routing table is well designed. The availability of the routing table is high. That is, more than 80% of the entries in this table are connectable. Furthermore, the entries of routing tables among peers which are logically close with each other are different, which means these routing tables have low similarity or large diversity.
2) We discover that the maintenance policy for the resource location information publishing table is not well designed. The availability of the publishing table is low. On average, more than 75% entries in this table are stale and cannot be connected.
3) We also observe that more than 75% peers leave the system within one hour after publishing their downloaded content into the KAD network.
4) By exploring the implementation of the KAD network, we conclude that both the current maintenance schedule for the publishing tables and the selfishness of the publishing peer eventually results in the low availability of the publishing tables, and accordingly cause poor lookup performance of the KAD network.
5) To deal with these issues, we proposed three possible solutions to address all of these issues: the self-maintenance scheme with short period renewal interval, the chunk-based publishing/retrieval scheme, and the fairness scheme. The strengths and weaknesses of these solutions are also discussed.

To improve the performance of KAD network, previous works [13], [16] mainly focused on how to deal with published peers more efficiently. However, to our knowledge, the impact of the

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States
Copyright © 2012 ICST
DOI 10.4108/icst.collaboratecom.2012.250424

publishing peers on the lookup performance have not been analyzed until this work.

The remainder of the paper is organized as follows. In section 2, the background of KAD networks such as content publishing and retrieving mechanisms is reviewed. In section 3, a group of measurement studies in the performance of the KAD network are presented, including analyses of the results. The proposed solutions to mitigate the current issues are discussed in section 4. In section 5 and section 6, the related work and conclusion are presented.

II. Background

In the KAD network, each object (e.g., peers, published content information, keywords) has a unique 128-bit long identity called the KID. The KID of each peer is generated when it first joins the system. Using the Kademlia algorithm [4], the logical distances among peers can be calculated by bitwise XOR operation. For example, if peer A, B and C's KIDs are 1010, 0101, and 1100 respectively, the distances between A and B is 1111 and between A and C is 0110. Consequently, C is recognized closer to A than B. The distances also decide where the publishing items will be published and then be retrieved. In the KAD network, a publishing item with the KID k will be published onto other online peers. These peers together form a tolerance zone of k in the KAD network, where each of them have enough prefix-matching bits with k. Furthermore, this publishing scheme also simplifies the KAD lookup process. That is, to retrieve a needed information with the KID k, users just need to find the peers who belong to the tolerance zone of k.

In the KAD network, each peer individually maintains a routing table for its knowledge of other peers. The entries in the table are the connecting information of its known peers, such as peers' KIDs, the corresponding IP addresses, UDP ports, TCP ports, etc. A peer can obtain more peers' information either when it directly receives requests from new peers or it requires its known peers to send back more peers' information. On the other hand, a stale peer's information can also be removed from the routing table by periodically online verification.

The logical structure of a peer's routing table is represented by a binary tree as shown in Figure 1. Each level of the tree corresponds to one bit of the KID. Theoretically, the tree's height can be extended to 128 levels corresponding to the total length of the KID. Peers connecting information is collected into the tree's leaf nodes named buckets. The KIDs of peers belonging to a bucket at a specific level will have the same prefix bits until that level. A bucket may hold at most 10 entries, beyond which the bucket must be split into the next level for holding more information. In reality, each peer in the KAD network is only required to maintain information from more peers with a closer match to it and maintain fewer peers whose KIDs are farther away. There are no buckets in the levels 0 to 3. The left-hand branch of the whole tree shown in Figure 1(a) always stops at level 4, where a total of 8 buckets hold the information of at most 80 peers. These 8 buckets cannot be split into the next level. For the right-hand branch shown in Figure 1(b), there are 3 buckets in level 4. Beginning from level 5, the 5 left-most buckets cannot be split anymore, while each of the other 5 right-most buckets can be split into the next level if more than 10 peers in its range are known. Consequently, the structure of each peer's routing table becomes an unbalanced binary tree . Our measurements show on average each routing table holds the information of less than 600 peers. On the other hand, with the help of this special structure, A peer's connecting information can be easily located. For instance, if peer A wants to connect to another peer B for which B's KID is known but its connecting information are unknown, A can search for peers from its specific bucket that has the longest prefix-matching bits with B, and send the requests to them. These peers will then run the same process to find much closer peers to answer the request of peer A. Recursively, A can obtain the connecting information of B.

To publish and retrieve content from the KAD network, each peer needs to search information not only from the routing table but also from the publishing table. The publishing table holds the information of how to find the real content location. When a peer publishes its sharable content with KID k, as the dot lines shown in Figure 2, it looks up several published peers in its routing table or recursively from other peers' routing tables whose KIDs belong to the tolerance zone of k (step 1). After that, it inserts its location information (e.g., its own IP address, transmission ports, etc.) into the publishing tables of the published peers (step 2). Equivalently when a user wants to retrieve the same content, it also first looks up its own routing table or recursively requests other peers to find the corresponding peers within the same tolerance zone of k (step 3), and then send requests to these corresponding peers (step 4). These peers will search their local publishing tables, retrieve the content location information, and reply back to the requester (step 5). With the received location information, the user can finally connect to the publishing peers and conduct a downloading process (step 6).

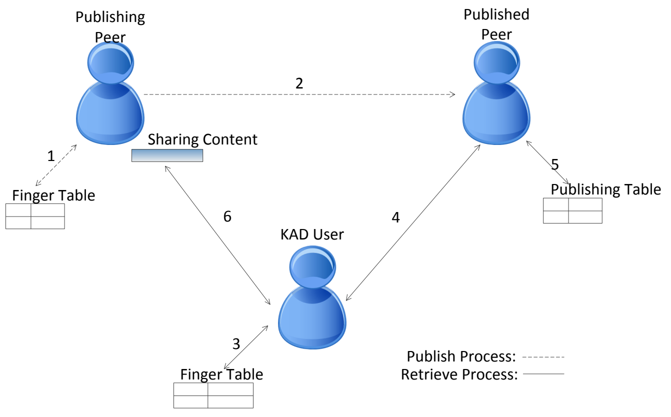
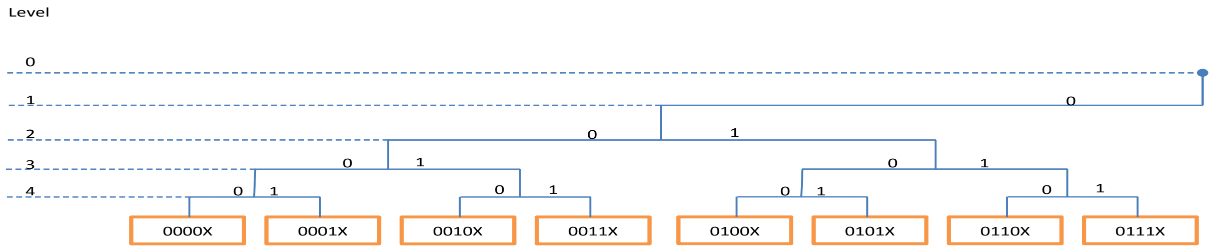
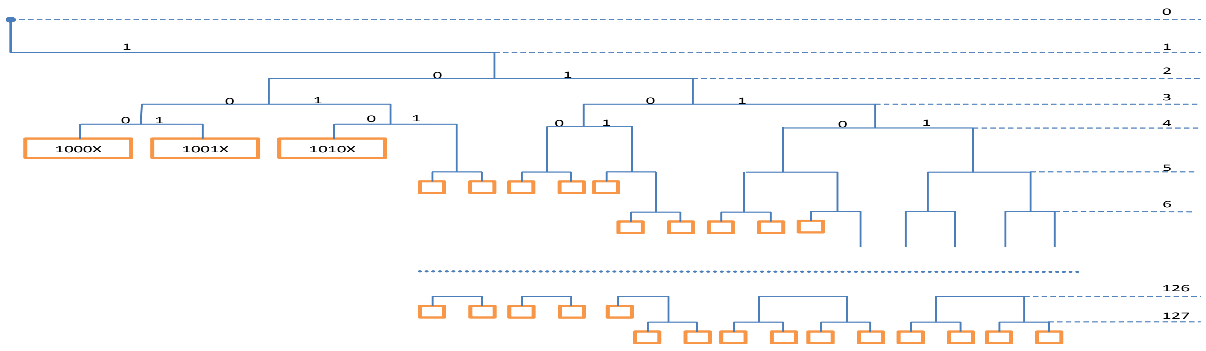


Fig. 2: Publishing and Retrieval Processes.



(a) left binary tree



(b) right binary tree

Fig. 1: logical structure of the routing table

III. Measurement-based analysis

In this section, we first review our previous work that measured the lookup performance of the KAD network from user's perspective. Then, we show our measurements for the maintenance of the routing table and the publishing table by using multiple nodes on the PlanetLab testbed.

A. Lookup performance of the KAD network

To explore the lookup performance of the KAD network, in our previous work we first conducted a client-based measurement in the real world. We chose a popular client application–eMule v0.49c for our client-based measurement. This application has been installed on a typical PC with a 100Mb Internet connection. The downloaded content is selected from a large scope of popular but non-copywrite types like audio, video, Linux ISO-distributions, etc., and the size of the content also varies from several MB to thousands of MB. This measurement took place over more than two months. The client application downloaded over 100GB of content. During the test, we recorded the number of useful sources from different source-searching methods (For more details of the measurement, please reference our previous work [9]).

As shown in Figure 3, the fully distributed DHT-based KAD network provides much less sources information in contrast with other two methods: the Source-Exchange method and the Passive-Exchange method used in aMule/eMule. Thus, it is natural to ask what the reasons are for the lower lookup performance of the KAD network. Considering the key roles of the routing table and the publishing table on the lookup processes of the KAD network, we are motivated to explore their performance.

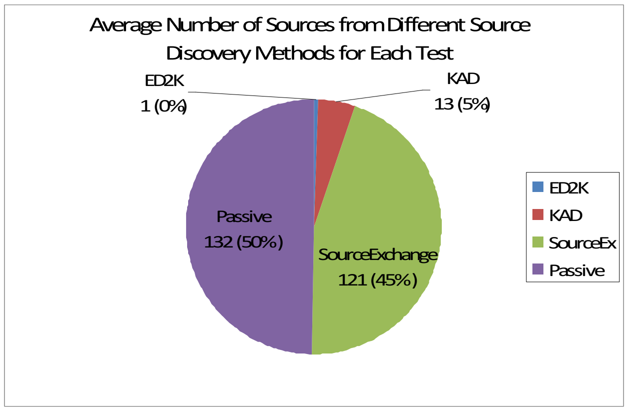


Fig. 3: Received sources information from different source-lookup methods.

B. The routing table measurement

One of the possible reasons for the poor lookup performance may come from a poorly maintained routing table. If the routing table is full of stable entries, which means its availability is low, peers may not get enough available information to locate other useful peers for either publishing or retrieving. Or, like Kang's work [10] mentioned, another reason is that peers within a tolerance zone may have a large amount of the same entries among their routing tables, in here we call the routing tables of these peers have high similarity. Thus, these peers can only publish information on the same peers that they know. However, when users try to retrieve their published information from other peers who belong to the same zone but are not the published peers (this can be possible, since peers may not know all the members belonging to the same zone), they cannot obtain this published information and consequently the lookup performance becomes inefficient.

1) Measurement Metrics: To verify these potential causes, we conduct two experiments by measuring the availability $F_a(t)$ and the similarity $F_s(t)$ of the routing table separately. If n_f and n_l represent the total number of entries and the total number of the living nodes in a routing table respectively, the availability F_a of a routing table at a specific measurement time equals

$$F_a = \frac{n_l}{n_f}. \tag{1}$$

As a result, a high value of F_a usually represents a well maintained routing table.

To measure the similarity F_s among m peers belonging to a specific tolerance zone at a particular measurement time, we denote $n_f(i)$ as the total number of entries in peer i's routing table and n_s as the total number of the same items among all routing tables in all m peers. Therefore, the similarity F_s of a group of m peers can be defined as the total number of the same entries in their routing tables divided by the average number of all entries in their routing tables.

$$F_s = \frac{n_s}{\sum_{i=1}^{m} n_f(i)/m}. \tag{2}$$

Consequently, the smaller the value of F_s, the better the routing table has been maintained.

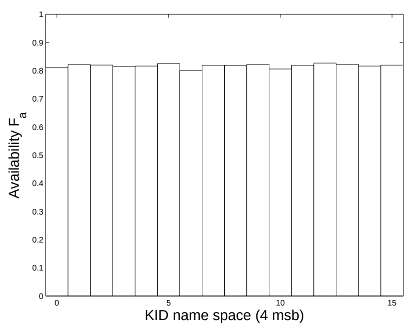
2) Measurement Methodology: We build a distributed measurement framework and deploy it onto multiple nodes using the PlanetLab testbed. Two applications are developed for measurement: KADmon and RoutingTCrawl. KADmon is a customized KAD client by modifying a popular aMule client application – aMule v2.26. It as our monitors is installed on 16 different PlanetLab nodes. We uniformly separate the entire KID name space into 16 parts. To assign each PlanetLab node to measure each part, the most significant four bits among the total 128-bit long KID of each node are uniquely assigned from 0, 1, 2...... E, F, while the 124 remaining bits are randomly generated. During the measurement, each PlanetLab node joins the KAD network and maintains its routing table accordingly. It also records its routing table into a log file. To reduce the impact of the test on the real system, these PlanetLab monitors neither download content from any other peers nor share content to them. On the other hand, another measurement application RoutingTCrawl is also run on each PlanetLab node to measure F_a and F_s. RoutingTCrawl verifies the online status of each entry from the routing table of the monitor via Ping-Pong requests. If the corresponding peer is online, its whole routing table is crawled and the online status of each entry in its routing table is tested.

Algorithm 1 Crawling routing Table

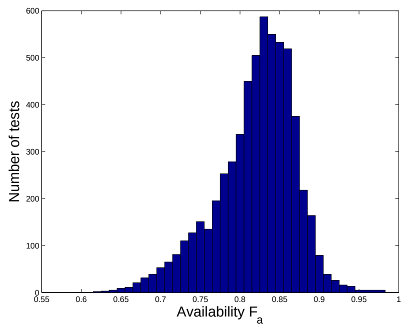
```
Data:
    list: requestList=generated requests packets
    list: targetList=peers retrieved from a routing table
    list: livingList=verified online peers
    hash(128): kid=peer's kid
    timestamp: t=timeout
    int: retrycount=# of retry when timeout
    int: parallelcount=# of requests in parallel
Initialization:
    /*according to the crawled peer's kid*/
    targetList=generateTargetKids( )
    /*according to the targetkidList*/
    requestList=generateRequests( )
Test:
    for each entry in requestList, do in parallel
        send request to the target peer
        if received response, then
            add to the livingList
        else if no response and timeout then
            if retrycount>0 then
                retrycount-1
                send request again
                wait for response
            else
                mark the peer offline
    /*livingList/targetList*/
    calculate availability $F_a$
```

Algorithm 1 presents our routing table crawling method in detail, which is minimally discussed in previous works [10], [8], [17]. In the KAD network, by carefully choosing KID k, one single Kademlia-Request packet can retrieve a whole bucket of any peer. Furthermore, according to the maintenance algorithm of routing table and the current population of the KAD network (there are about one million peers (2^{20}) uniformly distributed in the KAD network within the 128-bit long space [8]), each peer's routing table includes the information of less than 1000 peers ($(11+5\times15+10)\times10 = 960$, here 11 represents the total buckets in the first to 4th levels. There are 5 buckets in each of the following 5th to 19th levels, and the last 20th level has 10 buckets). Therefore, by sending less than 100 carefully created Kadmelia-Request packets in parallel, we can retrieve the whole routing table of any peer within one minute. Since KAD messages use UDP for transmission, we introduce the retransmission mechanism for preventing packet lost situation. Moreover, the multi-threading method is also adopted for testing peers' online status quickly.

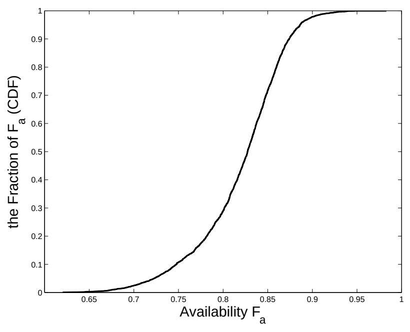
3) Routing table availability measurement: We run our PlanetLab monitors KADmon to collect peers information. To obtain a unified perspective of the KAD network, this initial process was both performed and terminated at the same time. After that, the RoutingTCrawl application at each monitor node was called immediately, and around six thousand peers distributed among the entire KID name space have been crawled, which means that every entry of the routing table of each crawled peer has been retrieved and tested. Table I shows



(a)



(b)



(c)

Fig. 4: (a) average value of F_a measured in different aspects of the KAD name space; (b) Histogram of the peers availability F_a of the routing table; (c) CDF of the peers availability F_a of the routing table

the measured data. Figure 4(a) shows the average values of availability F_a for each of these 16 different aspects of the KID name space. As the measurement results shown in Figure 4(b) and 4(c), there are around 80% of living peers in the routing table. This high availability of the routing table benefits from the current maintaining schedule of the routing table, which refreshes each bucket every minute. With this high availability, users can find their needed online peers from the routing tables for their publication/retrieval. Therefore, the availability of the routing table is not an issue for the lookup performance.

TABLE I: Measurement Data for Routing Table Availability

Data	Value
Total number of Crawled Peers	5985
Average Number of Entries	549
Average Availability F_a	0.819
Standard Deviation of F_a	0.050

4) Routing Table Similarity Measurement: Kang et al. [10] measured the routing tables between two peers within a specific tolerance zone and indicated that they on average has a high similarity of 70%. However, only testing the similarity between two peers is not enough. This is because to publish or retrieve information in the KAD network, users each time have to conduct at least three peers in the same zone, and obtain information from their routing tables. Steiner et al. Our work also measured the similarity among more than two peers. [16] also indicated that increasing this value can improve the KAD performance.

Our work measured the similarity among more than two peers within the same tolerance zone. Like Kang's work did, the 16-bit tolerance zone was selected for measurement. Within this zone, the KIDs of all peers have at least the 16-bit prefix-matching. By using our measurement components KADmon and RoutingTCrawl, we retrieved peers from the routing tables of the Planetlab monitors and crawled their routing tables. From the obtained routing tables, the peers belonging to the same 16-bit tolerance zone were selected and their routing tables were crawled. To obtain more peers for comparison, we continuously retrieved new appropriate peers from the routing tables of previously crawled peers, and crawled the routing tables of them. During the test, the routing tables of a total number of more than 13 thousand peers belonging to five thousand different tolerant zones have been crawled and compared. their results is in contrast to our observations.

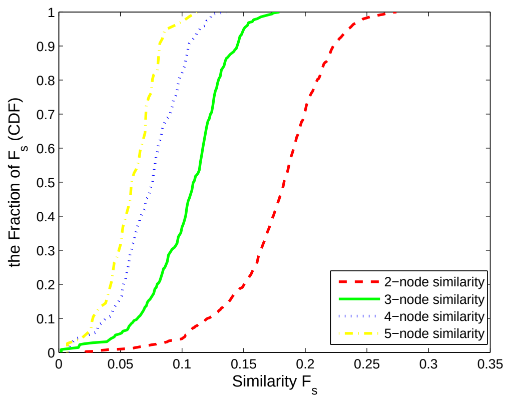


Fig. 5: CDFs of similarity for different number of compared nodes

The measurement results are shown in Table II and Figure 5, where the average value of the similarity F_s, the standard deviation of F_s, and the CDF of F_s among 2, 3, 4 and 5 peers within a specific tolerance zone have been presented. Our measurement reveals that the similarity F_s within two individual peers is only less than 20%, which is in contrast to Kang's work. Further than that, the similarity significantly reduces to less than 10% when comparing to more than two individual peers within a same tolerance zone. Therefore, The routing tables of the peers within a 16-bit tolerance zone have a small similarity or large diversity. The reason can be explained as follows. Under an open KAD environment, individual peers have its own individual behaviors: they may both arrive and leave the system at different times; they may meet different peers when sharing or requesting different content. Consequently, their perspective to the KAD network become differentiated. Moreover, the design of the KAD network may also deliberately exacerbate the high diversified viewpoint among each peer. That is, to fill out its routing table, each peer are designed to randomly search for more

TABLE II: Measurement Data for Routing Table Similarity

# of compared routing tables	# of tested tolerance zones	# of tested peers	average value of F_s	Standard Deviation of F_s
2	4120	8240	0.171	0.050
3	1018	3054	0.093	0.043
4	329	1316	0.069	0.029
5	102	510	0.060	0.022
Summary	5569	13120		

new peers instead of explicitly synchronizing the routing tables among its close peers. On the other hand, we believe the low similarity of routing tables does not degrade but improve the lookup performance in the KAD network. This is because a low similarity can keep publishing peers having a much broader perspective to the whole KAD network. As a result, more peers belonging to the same tolerance zone can be found and chosen for the publication, and this also causes users to retire the published information more easily. In contrast, if the routing tables of the peers within a zone have a high similarity, the information will be published onto a limited number of peers. When these peers leave the system, no information can be retrieved by users. In summary, high similarity is not the critical issue for the poor lookup performance, since it is not detected. Since Kang et al. [10] didn't present their measurement setup in detail, the accuracy of their test cannot be evaluated.

C. The publishing table measurement

Poorly maintaining publishing tables will also affect the lookup performance. This is because after the appropriate published peers have been located from routing tables, resource information have to be eventually published into/retrieved from publishing tables of these peers. In the KAD network, two kinds of publishing tables are used to maintain publishing information: the resource-location-information (RLI) publishing tables and the keyword-information (KI) publishing tables. The content location information will be inserted into RLI publishing tables of published peers. The related keywords for the publishing content will be inserted into the KI publishing tables. The importance of the RLI publishing table is due to it containing the actual locations of content, while the KI table conveniently helps users look for desirable content by Meta data, like file name or wild cards characters. In the following, we conduct real world measurements to verify the availability of the RLI publishing table. Then, we use the obtained results to evaluate the maintenance of the KI table, because the KI table are also maintained by the same schedule of the RLI table.

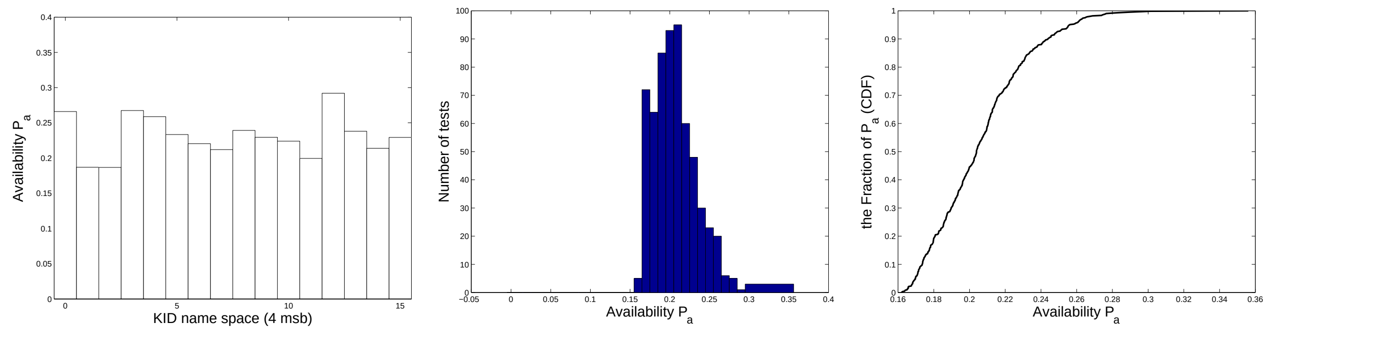
Using the same measurement method in section 3.2, we test the publishing tables of multiple nodes that are uniformly distributed on the entire KID name space. This time all 16 monitors from the PlanetLab testbed are acting as published peers. They accept the relevant publishing information, and build their own publishing tables accordingly. Every half hour, the entire RLI publishing table of each monitor is recorded into a log file. At the same time, a developed crawler component called PublishTCrawl is triggered. PublishTCrawl employs the same technique used by the RoutingTCrawl component in section 3.2. It reads this log file and sends the KAD ping request to test the online status of each entry in the RLI publishing table. Similar to the routing table measurement metric F_a, a measurement metric P_a are also defined to measure the availability of the RLI publishing table. For a single RLI publishing table, P_a is equal to the total number of living peers' information divided by the total number of the recorded peers' information.

The test was run on 16 PlanetLab-based monitors for 25 hours. In each trial, on average the online status of four thousand peers were tested. We totally conducted 800 measurement trials, and a total number of more than three million peers have been connected.

The measurement results shown in Figure 6 explicitly reveal that the RLI publishing table is not well maintained, which is the main reason for the poor lookup performance in the KAD network. On average, only less than 25% of peers recorded in the publishing table are actually online. Consequently, when users request information from these RLI tables, more than 75% of the entries from the corresponding reply are useless. Moreover, Figure 6(a) indicates that this trend is consistent within the entire KAD name space.

By exploring the KAD maintenance algorithm for the RLI publishing table, we have discovered some serious design issues. The KAD protocol requires each publishing peer is responsible for maintaining its publishing information. In detail, when resource location information has been inserted into publishing tables of published peers for the first time, a living-time period of five hours has also been attached. After five hours, If the publishing peers are still online, they will renew those information; if the publishing peers are offline, their publishing information will be automatically removed from the publishing tables after 30 minutes. This method has no problem when the publishing peers keep staying online. However, if publishing peers leave the system within 5 hours after its publication, both the published location information and the published keywords information become useless for the users subsequently searching for the rest time period. The KI publishing table is also maintained by the same scheme, except that the living-time period for each publishing entries has been extended to 24 hours. Thus, it also suffers from the weakness of this maintenance scheme.

If peers really leave the system within 5 hours of publication, This will be the main issue that causes the low availability of publishing tables and consequently the critical reason for the inefficient lookup performance. To verify this, we examine publishing peers behavior by checking their online status every 30 minutes. Our measurement shown in Figure 7 verifies that the majority of the publishing peers (more than 75% of



(a) (b) (c)

Fig. 6: (a) average value of P_a measured in different aspects of the KAD space; (b) histogram of the source-location-information publishing table; (c) CDF of the availability P_a of the source-location-information publishing table

them) leave the system within one hour after they publish their content. Since the publishing mechanism in aMule/eMule forces completed downloading content be immediately published onto the KAD network. our measurement discovers that the majority of peers are selfish and they will leave the system quickly after they finish their downloading tasks.

In summary, the reasons for the low availability of the publishing tables and the poor lookup performance of the KAD network can be summarized as follows:

- the selfishness of the publishing peers;
- the publishing tables of published peers are maintained by the publishing peers;
- the poorly designed 5/24 hours maintaining schedule for both the RLI publish tables and the KI publishing table.

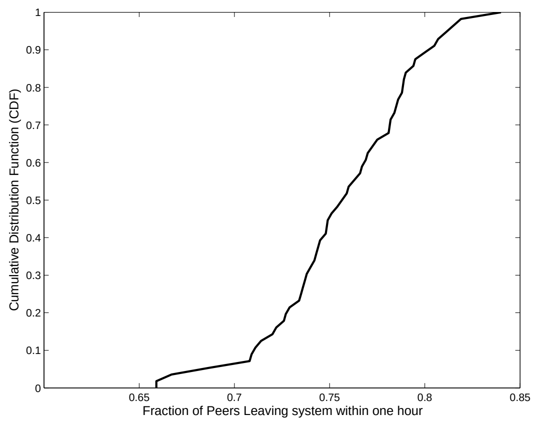


Fig. 7: CDF of the percentage of selfish peers leaving the system within one hour after publishing content to the KAD network

We noticed that previous works[16], [13], [10] mainly focus on improving the performance at the published peers side. However, with the selfish departure of the publishing peers, the publishing information become meaningless. Therefore, we argue that strictly controlling peers behavior should become a necessary consideration for the KAD design. Actually, protocols should be designed to deal with the high level of the selfishness.

IV. Discussion of Possible Solutions

To solve the low lookup performance issue of the KAD network , we discuss three possible schemes in this section and show their advantages and weaknesses respectively.

A. Self-maintenance Scheme

One possible scheme is to assign the task of maintaining publishing tables back to the local peers (published peers) themselves. The published peers can verify the online status of entries in their publishing tables within a short time period (e.g., 10 minutes), or they may directly synchronize peers' information from their well-maintained routing tables. With this modification, the stale records in the publishing tables can be significantly reduced, and the retrieved information can be more useful. Furthermore, this method does not change any fundamental structure of the current system, and it unifies the maintenance pattern of the routing table and the publishing tables. This scheme may also be simply implemented. However, because of the peers' selfish behavior, this scheme improves the availability of the publishing tables just based on reducing the number of useless records. It cannot fundamentally improve the performance of the KAD network, because the amount of more useful information to users was not increased. Finally, since it needs to renew publishing tables frequently, it may also introduce much more table maintaining traffic into the system.

B. Chunk-based Publish/Retrieval Scheme

We can use a chunk-based publishing/retrieval scheme to replace the current file-based publishing/retrieval scheme. In aMule/eMule, the chunk-based downloading and uploading scheme has already been employed to speed up the content sharing process. Under the chunk-based scheme, a file is usually separated into several chunks and each chunk can be downloaded from different peers and uploaded to others simultaneously. However, instead of publishing the information of obtained chunk immediately, a peer in aMule/eMule publishes the information of an entire file into the KAD network. That is, under the current publish/retrieval scheme, the file will not be published into the KAD network until it has been fully obtained.

TABLE III: Comparison of Different Schemes

Methods	improve P_a	improve the amount of useful information	reduce selfish behavior	change current structure	introduce more traffic
current scheme	No	No	No	No	No
self-maintenance	Yes	No	No	No	Yes
chunk-based	Yes	Yes	No	Yes	Yes
fairness policy	Yes	Yes	Yes	Yes	Yes

In a typical P2P file-sharing environment, a peer usually runs both the uploading and the downloading processes concurrently, and it will not leave the system before it completes its download tasks. Thus according to the chunk-based publish/retrieval scheme, if the information of peers possessing some chunks of the whole file can also be published, the availability of the publishing table will be significantly improved. This modification can increase the amount of useful information to users. However, it requires a redesign of the publishing/retrieving scheme. Moreover, it will also complicate the maintenance of the publishing tables and introduce more traffic into the system.

C. Strict Fairness Scheme

The third solution is to adapt some fairness policy to mitigate the selfish behavior of peers. Currently, aMule/eMule has already deployed a local credit system to reward the sharing behavior. However, this policy is neither efficient nor fair [9]. It also has not been used in the KAD network. Since peers' selfish behavior will lead to the low lookup performance of the KAD network, some strict fairness modification such as both rewarding generous peers and publishing selfish ones have to be introduced to the KAD network. Another benefit for adopting the fairness-based modification is due to its ability to keep the publishing/retrieving scheme unchanged. Consequently, it may make the implementation relatively easy. On one side, the modification based on fairness design will fundamentally reduce the peers' selfish behavior and accordingly improve the lookup performance of the KAD network. On the other side, the introduction of a strict fairness policy will still increase the complexity of the P2P file-sharing system .

We have summarized the comparison of these modifications in Table III. It looks that no modification is perfect. We believe that the combination of these possible modifications will be the most valid scheme to deal with the lookup performance issue of the KAD network.

V. Related work

The KAD network is one of the largest deployed DHT networks integrated into the aMule/eMule P2P file-sharing system [6], [7] with millions of users [8]. Its implementation is based on the Kademlia algorithm [4], which uses the binary tree as its logic structure of the routing table and uses the XOR operation to calculate the logic distance among peers. Due to its large deployment, the research community has shown great interest in the KAD network. For instance, Brunner [18] analyzed the implementation of the KAD network in detail. Memon et al. [19] developed a measurement tool that can accurately monitor the KAD traffic. Varvello et al. [20] created a social virtual network on top of the KAD network. Steiner et al. [21] mentioned that misusing the KAD network will easily conduct a DDOS attack. Steiner et al. [22] also pointed out several characteristics of peers in the KAD network such as geographical distribution of peers, daily usage, and peers' lifetimes by their measurement study.

In aMule/eMule, the KAD network is used to implement the content information publication/lookup function. Stutzbach et al. [13] measured the accuracy of the routing table, and proposed to improve the performance by using a parallel lookup and increasing the number of published peers. Steiner et al. [16] modeled the lookup process, and propose a strategy to reduce the lookup latency. Kang et al. [10] discovered the lookup issue that users can only find few published peers. They believed the reason is due to the high similarity of the routing tables. All the studies measured the KAD network through a modified client viewpoint or only measured a specific aspect of the entire KAD name space. On the other hand, to improve the performance of KAD network, previous work focused mainly on how to deal with published peers more efficiently, such as by increasing the number of parallel requests to different published peers [13], or by publishing content on more published peers [16]. However, the impact of the publishing peers on system performance has not been addressed until this work.

Comparing to previous measurement studies, we believe that our work measure the whole KAD network by adapting a distributed measurement framework via the PlanetLab testbed in the first time. We are also the first to analyze the publishing tables in the KAD network, and to reveal the key factor that affects the KAD lookup performance: the publishing peers' selfish behavior.

VI. Conclusion

This work focuses on investigating the reasons for the poor lookup performance of the KAD network. We conduct several large-scale measurements to analyze the maintenance of both routing and publishing tables which are the key components in the publishing/retrieving process. By deploying our measurement framework on the PlanetLab testbed, we first test the availability and the similarity of peers' routing tables. Our results show that on average more than 80% of nodes in the routing table are online and less than 25% of records are the same among different routing tables of peers belonging to a tolerance zone. This means that the routing table is well maintained and a peer can use it to find the desirable peers easily. After that, we measure the RLI publishing table and found that on average only around 25% of items in this table are online. Furthermore, our measurement reveals that over 75% of peers leave the system within one hour after publishing content. By analyzing the KAD protocol, we discover that both the current maintenance method for the publishing tables and the selfishness of the publishing peer are the reasons for the low availability of the publishing tables and accordingly cause the poor lookup performance of the KAD network.

Future work can consider the implementation of these modifications and the comparison of their impact on the KAD performance through simulation and real world experiments. It should also consider the design of a brand-new publishing mechanism by combining the publishing-control scheme and the published-control scheme.

VII. Acknoledgements

We would like to thank PlanetLab, Sapan Bhatia, and Thierry Parmentelat for their supports during our measurement.

References

[1] "Internet study," http://www.ipoque.com/resources/internet-studies/internet-study-2008-2009, 2009.

[2] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the ACM SIGCOMM Conference*, San Diego, California, August 2001.

[3] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, ser. Middleware '01. London, UK: Springer-Verlag, 2001, pp. 329–350. [Online]. Available: http://dl.acm.org/citation.cfm?id=646591.697650

[4] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[5] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, Jan. 2004.

[6] aMule Officical Website, http://www.amule.org.

[7] eMule Officical Website, http://www.emule-project.net/.

[8] M. Steiner, E. W. Biersack, and T. Ennajjary, "Actively monitoring peers in kad," in *In Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[9] Y. Li, D. Gruenbacher, and C. Scoglio, "Reward only is not enough: Evaluating and improving the fairness policy of the p2p file sharing network emule/edonkey," *Peer-to-Peer Networking and Applications*, vol. 5, pp. 40–57, 2011.

[10] H. J. Kang, E. Chan-Tin, N. J. Hopper, and Y. Kim, "Why kad lookup fails," in *in IEEE Peer-to-Peer Computing (IEEE P2P)*, 2009.

[11] T. Locher, D. Mysicka, S. Schmid, and R. Wattenhofer, "A peer activity study in edonkey and kad," in *International Workshop on Dynamic Networks: Algorithms and Security (DYNAS)*, 2009.

[12] M. Steiner, T. En-Najjary, and E. W. Biersack, "A global view of kad," in *In ACM Internet Measurement Conference (IMC)*, 2007.

[13] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed dht," in *In 25th IEEE International Conference on Computer Communications (INFOCOM)*, 2006.

[14] J. Yu, C. Fang, J. Xu, E.-C. Chang, and Z. Li, "Id repetition in kad," in *IEEE International Conference on Peer-to-Peer Computing*, 2009.

[15] PlanetLab, "http://www.planet-lab.org/."

[16] M. Steiner, D. Carra, and E. W. Biersack, "Faster content access in kad," in *in IEEE Peer-to-Peer Computing (IEEE P2P)*, 2008.

[17] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *In Internet Measurement Conference(IMC)*, 2006.

[18] R. Brunner, "A performance evaluation of the kad-protocol," Master's thesis, Institute Eurecom, 2006.

[19] G. Memon, R. Rejaie, Y. Guo, and D. Stutzbach, "Large-scale monitoring of dht traffic," in *In Proceedings of 8th International Workshop Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[20] M. Varvello, C. Diot, and E. Biersack, "P2p second life: experimental validation using kad," in *In Infocom*, 2009.

[21] M. Steiner, T. En-najjary, and E. W. Biersack, "Exploiting kad: Possible uses and misuses," *ACM SIGCOMM CCR*, vol. 37, pp. 65–69, 2007.

[22] 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, pp. 1371–1384, OCTOBER 2009.