

Scalable discovery of private resources

Spyros Kotoulas and Ronny Siebes
Vrije Universiteit Amsterdam
Department of Computer Science
De Boelelaan 1081, 1081HV, Amsterdam
{kot,ronny}@few.vu.nl

Abstract—Resource discovery is fundamental to a multitude of distributed systems, including grids, web-based applications and multi-agent systems. To achieve scalability at a low cost, many researchers have turned to a peer-to-peer paradigm, leading to the development of a multitude of protocols and algorithms being developed, with implementations still lagging behind. In this paper we consider the privacy implications of peer-to-peer discovery systems and propose a framework for discovery of private resources. Furthermore, we propose and evaluate an architecture and a series of methods using distributed hash tables. Finally, we provide an implementation in the context of the OpenKnowledge project.

I. INTRODUCTION

Resource discovery is about localizing computational resources. It is of paramount importance for a large proportion of computer systems. Practically all dynamic distributed systems require mechanisms to enable users and applications to locate resources scattered across the network. Notable examples are Grids, file and knowledge sharing systems, peer-to-peer systems, multi-agent systems and distributed knowledge bases. Despite their incongruence in architecture and assumptions, all such systems require at least this basic functionality: retrieving the location of resources given a set of keywords or attribute-value pairs.

With the notable exception of peer-to-peer search engines[1], most currently deployed systems are based on centralized architectures[2], [3], [4]. Nevertheless, such architectures are typically associated with a number of flaws: there is a single point of failure or a single point of administration, in turn leading to lack of scalability and selective or preferential disclosure of information, censorship, preferential ranking of results and centralization of information respectively.

Peer-to-peer systems are often seen as a vehicle for the democratization of distributed computing, since a collection of community-volunteered peers standing on an equal footing, collaborate to achieve a common goal. This architecture is perceived as capable of alleviating most of the problems mentioned above. Nevertheless, as we will show, current peer-to-peer discovery systems not only fail to solve privacy infringement issues, but they are actually more vulnerable than centralized approaches. The main reason behind this is that instead of the existence of one central point where privacy can be infringed, any peer in the system is a potential point of attack.

Furthermore, we argue that the techniques that are traditionally employed to provide security over unreliable overlays

are inadequate as far as protection of privacy is concerned. For instance, although a quorum-based approach relying on a number of peers voting to perform an action can be used to bestow robustness against malicious peers[5], it is detrimental in the event that this action has to remain private.

The focus of this paper is on scalable discovery of private resources using private queries. The term *private* signifies that peers, other than the one posing the query and the one answering, should not be able to view neither the resources published nor the queries posed (or a meaningful description of them). The term *scalable* signifies that it should retain good performance as the number of peers, resources and concurrent queries increases.

Initially, we examine how current solutions fare with privacy and define a framework for discovery of private resources. Based on this framework, we develop an obfuscated index on top of a Distributed Hash Table(DHT), where resources are published without revealing either their location or their description. In addition, we implement this method as part of the OpenKnowledge discovery[6].

II. PRIVACY IN PEER-TO-PEER DISCOVERY SYSTEMS

The purpose of discovery systems is to provide location services for large pools of resources. To this end, users or applications *publish* resources by sending a *descriptor*, an *identifier* and the *location* of a resource to the discovery system, and *query*, which means searching for the location of resources given a set of criteria. Once the location is established (i.e. when the discovery system has returned the locations of the resources which have descriptors matching the query), the *querier* may proceed to negotiate with the *publisher* of the resource.

Using the above terminology, we can identify a series of privacy challenges concerning the disclosure of the following information:

- **Identity of the publisher** Information that can identify which user or through which host a particular piece of information was published. For example, the IP address of the host, along with an identifier to the resource.
- **Identity of the querier** Information that can identify the issuer of a query.
- **Resource descriptor** A meaningful descriptor associated with an identifier of the resource.
- **Resource location** The location where a resource resides, or an association of a resource identifier with its location.

- **Content of the query** The content of a query, regardless of whether it is associated with a querier.

We advocate that the peer-to-peer paradigm can be used to preserve privacy as well as provide scalability and robustness in discovery systems.

We urge the reader not to confuse anonymizing networks with peer-to-peer discovery. The former refers to providing communication using anonymous endpoints which are known in advance while the latter refers to providing discovery functionality through a peer-to-peer network. Although it should be possible to provide an integrated solution, in this paper we consider the two problems orthogonal.

The benefits of a peer-to-peer architectures for discovery have been extensively studied[7]. Nevertheless, current peer-to-peer discovery systems do not fare well with privacy. In the following paragraphs, we will provide an overview of the privacy implications of current peer-to-peer systems in combination with their scalability properties organized according to overlay type.

A. Centralized index

The simplest scalable solution is to have a central register of keys that maps to the peers that store the resources. Napster[8] was such an example where resource descriptors and their locations are stored in a central register. For a query the register returns the location of the peer(s) that hold the matching resources. Although it was scalable in storage due to content distribution, there is a single point of failure which makes this architecture not robust against failure of the index node.

B. Unstructured overlays

In unstructured overlays, peers maintain a set of ad-hoc connections to other peers, also termed as *neighbors*. According to the criteria used to establish and preserve these connections, a further distinction can be made between network-based and semantic-based unstructured overlays. The former use information concerning the network to self-organize on, for example, hop count and latency. The latter use descriptions of the content of each peer, for instance, summaries of filenames or sets of descriptive keywords.

a) *Network information-based overlays*: The first fully-distributed peer-to-peer networks[1] use information concerning the underlying IP network to self-organize. Typical criteria to establish neighbor relations include number of IP hops and link quality.

Searches are typically flooded through the network, i.e. sent to all neighbors and forwarded further, similar to a breadth-first search. Although a series of improvements over the original model have been proposed, like focusing on reducing duplicate messages[9], each query still has to be propagated to a large percentage of all peers, making this approach non-scalable for resource discovery.

As far as privacy is concerned, the main advantage of this category of systems is that peers do not need to share either their resource descriptions or their location. Nevertheless, the content of queries and the querier are public.

b) *Semantic-based overlays*: Semantic-based overlays are similar to network information-based unstructured overlays but for the fact that they use semantic information about their resources instead of information about the network. In most cases, peers propagate summaries or descriptions of their data, and cluster according to content similarity or positive past experience.

Queries are propagated either in a similar fashion as with network-based overlays, or using similarity measures. In some cases, semantic-based overlays have shown performance improvement of one order of magnitude compared to network-based approaches, measured in terms of recall[10].

Unfortunately, this comes at a cost of reduced privacy as content descriptions need to be shared in order to perform peer clustering effectively. On the bright side, peers can choose to whom their descriptions will be sent (i.e. they may send descriptions of their content only to peers that they trust).

C. Structured overlays

Structured overlays impose a global structure on the peer-to-peer network. DHTs is a fecund research field with a plethora of applications [11], [12], [13], [14], [15]. Typically, each item stored in the DHT is associated with a hash ID chosen from a large key space. This space is partitioned, in a way similar to hash tables, but instead of bins, they have peers. This distributed data-structure is self-maintaining, self-organizing and guarantees lookups and insertions using, in most cases, $O(\log(N))$ messages, where N is the number of peers in the DHT overlay. Each peer is required to maintain a number of connections¹ to other peers according to the structure of the DHT. Therefore, their scalability properties are much better compared unstructured overlays, albeit with increased maintenance costs and reduced robustness.

To discover resources using a single attribute, the overlay can act as a distributed index, mapping from keywords to locations or identifiers. Structured overlays need only a single peer lookup. Furthermore, some of them also support range queries[13], [14]. For multi-attribute search, more complex approaches have been developed. Other systems place data according to their semantics. For example, pSearch[16] uses Latent Semantic Indexing to map documents in a multi-dimensional real-valued space maintained by a CAN.

Privacy-wise, current discovery systems have the worst characteristics. Firstly, resource descriptions have to be forwarded and indexed by the peer determined by the hash function used in the system. Not only does this not guarantee privacy, but it is also a potential security threat since peer placements in most overlays are not secure. This enables peers to ‘select’ the keywords they want to index, making targeting of specific topics easy. This means that the identity of the publisher, the content description and the resource location are disclosed to all peers routing the respective messages and to the peer(s) indexing the resource description.

¹logarithmic to the number of peers in the network [11], [12] or constant [14]

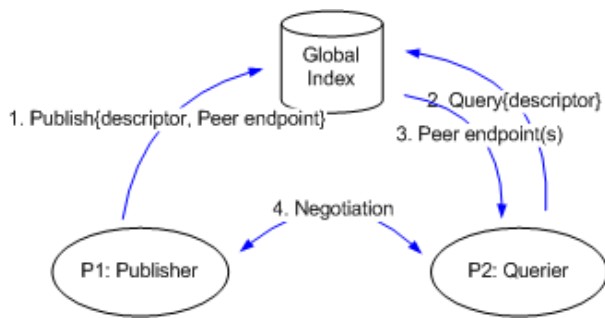


Fig. 1. A framework for discovery

Secondly, queries are propagated by a set of peers determined by the DHT mechanisms, again, not guaranteeing the privacy of the querier or the query content. Similarly, peers may choose to be responsible for the propagation of the queries they are interested in, further compromising privacy.

In short, structured overlays and systems based on them clearly outperform unstructured ones as far as scalability is concerned. Nonetheless, privacy has been taken in little or no consideration in their design. In the next section, we will define a framework, outlining where privacy attacks can take place, followed by an implementation of a mechanism to improve the privacy properties of peer-to-peer discovery systems.

III. A FRAMEWORK FOR DISCOVERY OF PRIVATE RESOURCES

As shown in the previous section, disseminating information about peer contents either in the form of advertisements, as in semantic unstructured peer-to-peer overlays, or in the form of a distributed index, as in structured overlays, alleviates the scalability issues of peer-to-peer discovery systems to a great extent. Nonetheless, it gives rise to significant privacy issues. In this section, we will provide a framework for the design of a peer-to-peer discovery system, where resource descriptions will be published in a global index, and fine-grained matching and negotiation will be done on peer level.

In figure 1, one can see an overview of the proposed model. *Publishers* send a *descriptor* of the resource they want to share along with the endpoint where they want to be contacted, for example their IP address and port number. This information is stored in a global index. A *querier* can send a query to this index, specifying a descriptor for the resource to be matched. Assuming a matching descriptor is stored in the index, the endpoint of the peer that has published it is returned to the querier. The latter contacts the publisher directly and they negotiate further, disclosing or requesting additional information about the requested and offered resource and participating parties.

The above model will be used to elicit additional scalability and privacy requirements. Concerning the former, it is obvious that the global index is the performance bottleneck of the system, as all other operations scale linearly with the number of peers. Therefore, a scalable global index is required(1). As far as privacy is concerned, the issue becomes slightly more complicated: Firstly, we cannot assume a trusted

global index, since that would hamper scalability. Therefore, published descriptors should convey as little information as possible. Secondly, these descriptions should preferably not be understandable by the index(2). Thirdly, queries should also not be easily understandable by the index(3). Fourthly, it should not be easy to probe for peers that published or queried for a particular type of resource, i.e. it should not be easy to identify the peers offering resources with a given description(4).

IV. A DISTRIBUTED OBFUSCATED INDEX

Following the requirements above, we propose a scalable peer-to-peer discovery system based on a distributed obfuscated index, on top of a DHT. Transposed to a real world scenario, our model is similar to the following: Arnold leaves a note on a public message board:

-I have some ho.... Call me at 234435

Barbie is searching for somebody that can give her some “hot...”. She goes to the message board and checks for messages that match hers. She can see that Arnold’s message matches hers, so she calls Arnold and says:

-I am Barbie and these are my credentials. I saw that you might have something for me. I am searching for something starting with “hot...”

Arnold responds giving his credentials. Barbie considers that he is trustworthy and tells him that she is searching for “hot soup”. Arnold responds that they do not match (without disclosing any additional information about his message).

Obviously, this message board would become crowded very fast. We need a more efficient way to find interesting notes. Assume that there is a message board on the door of each room in a corridor. These doors are numbered in sequence from 1 to 100. To determine where each message should be posted, assume the existence of a function that maps words to numbers in the range $[1, 100]$ and that similar words are mapped to similar values. For example, say that Arnold’s note “ho” maps to 5.4 and that Barbie’s query “hot” maps to 6.1.

Furthermore, assume that Maria, a student in the university, has to post the notes behind the door with number that lies closest to the number of the word, (i.e. a word that maps to 32,45 will be posted behind room 32).

In the example above, Arnold’s note would be posted behind door number 5(5,4 is rounded to 5).

Maria will map the query of Barbie to room number 6. Assuming that no matching note will be found there, she will search in the message boards of nearby rooms. In room 5, Maria will find Arnold’s note reading “5.4, Arnold@653435”. She will return a copy of the note to Barbie. Barbie will then contact Arnold in person, and they will negotiate further.

It is important to note the following:

- It is difficult to find out the content of the message from its number/hash. Furthermore, in order to improve security, we can round numbers. For example, assume that “Hot s” maps to 6,234. If we round it to 6, it is much more difficult to guess what word or phrase is meant, since there are many words that map to 6. Nevertheless,



Fig. 2. System using an obfuscated index

that would come at the expense of searching in more rooms.

- Persons in the rooms may be malicious and want to spy on the information from the notes. This is not possible, since they can not infer their meaning from their number to which they were mapped.
- Students gossip, so Maria may want to spy on information in the notes. Again it is not possible, since Maria cannot infer neither what Arnold’s query means, nor Barbie’s note, for the same reason as mentioned before.

Mapping the terminology above to computer science terms is as follows: The number of a word is its hash-value using a similarity-preserving hash-function, a corridor of rooms with message boards is a DHT, the student Maria is the message routing algorithm, a message board is a node in the DHT and rounding numbers is comparable to truncating hash-values (i.e. using a prefix of the hash-value).

Figure 2 outlines this functionality of a system based on an obfuscated index. We are assuming that resource descriptions and queries are made up by sets of keywords (e.g. “country”, “car”). To publish a resource descriptor, peers replace each keyword in that descriptor with a prefix of its hash, denoted as $p_i(h)$, where i is the length of the prefix and h the hashed keyword. This obfuscated descriptor is stored in the distributed index (maintained by the DHT), therefore DHT nodes do not have access to the original descriptors. Similarly, queriers replace each keyword in the query with its hash. Again, the original queries are not visible by the nodes in the DHT. The distributed index will return the endpoints of the peers which have registered descriptors with the same prefix as the query.

The role of i in the hash function acts as a knob to adjust the trade-off between privacy and performance. A high i means that few keywords will map to the same hash-value. This means that precision, which is the chance that the returned peers actually contain relevant resources, will be high. Nevertheless, the system would be susceptible to dictionary attacks, i.e. malicious peers trying to guess descriptors. On the other hand, a small i would increase the number of collisions in the hashes. This would make it impossible for attackers to make guesses about the content of a descriptor, since multiple uncorrelated keywords will map to the same hash-value. Of course, this will come at the expense of additional negotiation

steps (marked with 4 in fig. 2). Note that for $i = 0$, the system would degrade into a broadcast-based system where all peers would be returned by the index while for a very high i , the obfuscated index would behave as a distributed inverted index.

It should also be noted that i can be set per keyword and per peer basis. Therefore, for public descriptors and queries, a high i ensures scalability, while for private ones, a lower i ensures that it is very difficult to guess their content.

In the next section we provide a method how to determine i for individual keywords. In algorithm 1, we formally describe how the process of storing pointers in a privacy preserving way. We assume that a document or any other resource o is described by a set of keywords, which we give the symbol D_o . For each keyword in D_o we calculate the key p_t using one of the following options:

- **Option 1: Secure hash** Calculate the secure hash of the keyword and take a prefix of length l of this hash.
- **Option 2: Fixed prefix length** For each keyword, take a prefix l equal to the length of the keyword multiplied by ratio $r < 1$. The higher the r , the longer the prefix would be.
- **Option 3: Determine the prefix of the keyword according to its information value** This method is further described in section V-B.

Then, the descriptor of the resource, along with a pointer to its location should be routed based on the key p_t calculated above. The way that this key is routed is DHT implementation specific.

In algorithm 2 we formally describe how resource discovery is done. The functions to calculate the information value, the prefix, the key and the routing are identical to storing. Then the DHT is used to retrieve the locations of the resources matching these keys. Finally, the querier negotiates with each of the returned peers.

V. DESIGN CHOICES

In this section, we further describe some important design choices and discuss their implications.

A. Mapping function

DHTs work within an key space, which may be implementation-specific. For example, Pastry uses a single dimensional space with identifiers of 160 bits whereas CAN uses a multi-dimensional, real-valued cartesian space. Therefore, it is necessary to map keywords in the key space maintained by the DHT. Trivial as it may seem, it gives rise to load-balancing problems and has privacy and performance implications. In this section, we will give an overview of three ways to perform this mapping and discuss their characteristics.

1) *No hash*: Keywords are stored in the DHT in their original representation, possibly transcoding them to match the representation in of the DHT. Apart from simplicity, the most important advantage of this approach is that it is likely that keyword similarity is preserved. For instance ‘computer’ and ‘computers’ will have very similar keys and, most likely, will be stored in the same peer, providing good data locality.

Algorithm 1 Privacy preserving storage

Let resource descriptor $D_o := \{t_1, t_2, \dots, t_n\}$, be a set of keywords, where each $t \in STRING$ is a keyword describing the resource to be published.

for $i := 1$ to $i := n$ **do**

[OPTION 1: secure hash]

$p_t \leftarrow secure_hashkey(t_i, l)$ // *securehashkey* is a function that calculates a secure hash key, e.g. via SHA-1 into the key space of the DHT.

[OPTION 2: fixed prefix length]

$p_t \leftarrow prefix(t_i, r)$ // *prefix(t, l)* is a function that calculates the prefix of string t with length $t.length * r$.

[OPTION 3: Information value-based prefix length]

$i_t \leftarrow infValue(t_i, Dic, d)$ // *infValue* is a function that calculates the information value i_t of the keyword t_i according to a dictionary *Dic*. This function will be explained in the next section.

$p_t \leftarrow prefix(t_i, i_t)$

$dhtRoute(p_t, addr)$ // *dhtRoute* routes the location of the resource (*addr*) via the DHT with key k_t

end for

Algorithm 2 Privacy preserving retrieval

let α be a system parameter indicating the maximum number of peers to start a negotiation with, once the are found via the algorithm

Let query descriptor $Q := \{t_1, t_2, \dots, t_m\}$, be a set of keywords, where each $t \in STRING$ is a keyword in the search query.

Let $A := \emptyset$ be an empty set of (*addr, rank*) tuples containing peer addresses *addr* with a ranking *rank* indicating the number of prefixes it shares with the query Q .

for $i := 1$ to $i := m$ **do**

[OPTION 1: secure hash]

$k_t \leftarrow secure_hashkey(t_i, l)$

[OPTION 2: fixed prefix length]

$k_t \leftarrow prefix(t_i, r)$

[OPTION 3: Information value-based prefix length]

$i_t \leftarrow infValue(t_i, Dic, d)$

$k_t \leftarrow prefix(t_i, i_t)$

end for

$A = dhtRetrieve(k_{1\dots m})$ // retrieves a, possible empty, set of addresses via the DHT given the keys $k_{1\dots m}$ and stores it in the variable A

$startNegotiations(A, \alpha)$ // starts direct negotiation with the α best ranked peers

Nevertheless, privacy becomes an issue, since peers in the DHT will be able to see resource descriptions.

To make matters worse, keys will be unevenly distributed in the key space of the DHT. To illustrate our case, in English there are much more words starting with an 't' than with ab 'x'. This would limit the choice of DHTs to ones that support non uniform distribution of keys (e.g. [13], [14]).

2) *Secure hash*: An alternative would be to use a secure hash (e.g. SHA-1) and store the hash-values of keywords in a description. Such functions have the property that they map values uniformly in the key space, which means that descriptions are evenly spread across peers, leading to good load-balancing properties. Furthermore, they are one-way functions and it is very difficult to find a value that hashes to a specific key. This property is desirable to make sure that descriptors are kept private. Nevertheless, a dictionary-like attack is still possible. For example, an attacker participating in the DHT may calculate the hash-values of a large number of keywords from a dictionary and try to infer the original values in the descriptions it is storing. This is the reason that it is still required to use only a prefix of the hash, instead of all of it (also see next section). Finally, secure hashes do not preserve similarity between keywords: for example, 'coffee' and 'coffees' would map to completely different keys. This makes it impossible to support approximate matches, like edit distance matches or *startswith* matches, in the case it would have otherwise been possible.

3) *Similarity-preserving hash*: A plethora of similarity-preserving hash functions have been proposed [17] for a variety of applications. To the best of our knowledge, there are no secure or load-balanced variants of such functions. Therefore, the challenges presented are the same as when using no hash-function at all.

B. Determining key length

As previously mentioned, we assume that resources are described by a set of keywords t_1, t_2, \dots, t_n ². Please recall that the goal of having a prefix instead of the whole keyword is that the shorter the prefix the more other resources match, which makes it more difficult to know what the resource of interest was, and therefore increases the privacy of the querier and publisher. In other words, when a peer wants to determine the length of the prefix from a key for an resource, the peer should have the guarantee that, in most cases, this key maps to more than X other resources stored in the network. The bigger the X, the larger the privacy, but also the larger the network load. Thus, this X is an indication of privacy and network load. This means, the larger the desired X, the shorter the key needs to be. A key of zero length means retrieving all resources. The variance of the desired average is of influence to the choice on X. For example, (an extreme case) if in 99% of the times a key of length K maps to one keyword and in 1% of the time to 901, the average is 10 resources. In this extreme case, in

²Note that *keyword* should be interpreted in the broadest sense, as a descriptive string which could be meaningful for an application but meaningless in natural language. E.g. "ID43XT" or "cntry_code=NL"

99% of the cases K is a unique identifier for the resource and therefore 99% of the cases completely intrudes the privacy of the user which would be unacceptable both in keywords of privacy and large (but rare) network usage peaks.

We want to keep this variance in number of matching descriptions to a minimum. Imagine that the keys would be prefixes of words from the English dictionary. The popularity of characters is not evenly distributed and neither are their combinations. For example, a key length of three would give sufficient privacy to store the word “computer”, because many words start with “com” (e.g. “compatible” and “communication”). However, it also potentially generates quite some network load because the identifiers of these peers will all be communicated to the querying peer resulting in either large messages or many messages or both, depending to the kind of DHT used. On the contrary, if the keyword is “xantype”, the key “xan” would be a strong indication that you are storing “xantype” because not many words start with “xan”.

Combined with the type of mappings mentioned in the previous section, we can identify three possible methods to determine key length:

1) *No mapping and fixed length keys*: For all keywords, the same prefix length is used. For instance, we define that the prefix length is equal to 40% of the number of characters in the keyword. The obvious advantage of this approach is its simplicity. Secondly, assuming that i^β , where β is the base of character representation, is much larger than the number of peers in the DHT, there is no need for the DHT to support range queries. This would mean that the key space is large enough so as to be partitioned by the peers in the DHT. The disadvantage of this approach, as we will see in the evaluation, is that the variance is large.

2) *Key length automatically determined using the prefix information value*: In this method the length of the prefixes for each individual are based on its information value, i.e. popularity. The more popular the prefix, the more keywords match to it, the lower the information value, the higher the privacy. Using a dictionary, one can easily find out how popular a prefix is by simply counting the number of words starting with it. We now assume that the distribution of prefix popularity in a dictionary is identical with the keywords used in the network to describe the resources.

If a dictionary is not present, it is easy to make one by counting the prefix frequency of a set keywords representative of the keyword distribution in the system. A data structure suitable to store this information is a trie. Figure 3 shows an example. Such a data structure can be pruned to substantially reduce storage space.

3) *Using a secure hash*: Secure hash-functions map values uniformly to the key space. Therefore, no extra measures need to be taken to reduce variance. The only challenge is to find an appropriate key length.

VI. EVALUATION

The goal of this section is to evaluate the three typical options described above: Fixed ratio size prefixes, information

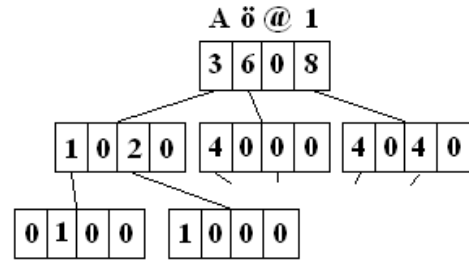


Fig. 3. A Trie data-structure is a tree of arrays containing counters. Each position in each array corresponds to a character. Following the trie one can derive the frequency of prefixes by starting at the root and recursively following the pointers for the current character. For example, the frequency of “A” is 3, and the frequency of “1@” is 4.

value-based prefixes and secure hashing. We will not evaluate the performance of DHT’s, because it has already been extensively studied [11], [12], [13], [14], [15]. We have two objectives:

- **Influence of the parameters on the balance between privacy and performance.** As mentioned before, the desired privacy influences the performance of the system. All of the options presented before have one parameter that adjust the trade-off between privacy and performance. We perform a series of experiments to assist future implementors in choosing the right values for their applications.
- **Variance on the number of returned peers and its impact on the privacy and performance.** To preserve the privacy of the querier and publisher, a minimum number of peers need to be returned that do not match the original query but only the prefix. On the other hand, if this number is too big, it will have an adverse effect on the performance. Clearly, the variance in the number of peers returned should be as low as possible because a high variance may result in compromising privacy and/or unacceptable performance loss. We use this criterion to evaluate our three options.

A. Dataset

Since, to the best of our knowledge, there exists no large dataset for resource descriptions, we decided to use a dataset created for general-purpose information retrieval, developed for [18]. Due to the semantic correlation between the keywords we assume that this dataset is an adequate substitute. Future research should give additional insight on several “keyword” distributions. For now, we assume that it is similar which the distribution of keywords in documents. Our dataset was created by crawling a large number of real user queries from SearchSpy³ and applying a natural language processing method on the results retrieved for these queries using Google to get relevant descriptions. The input to our system was derived from the following:

³<http://www.infospace.com/info.xcite/searchspy>

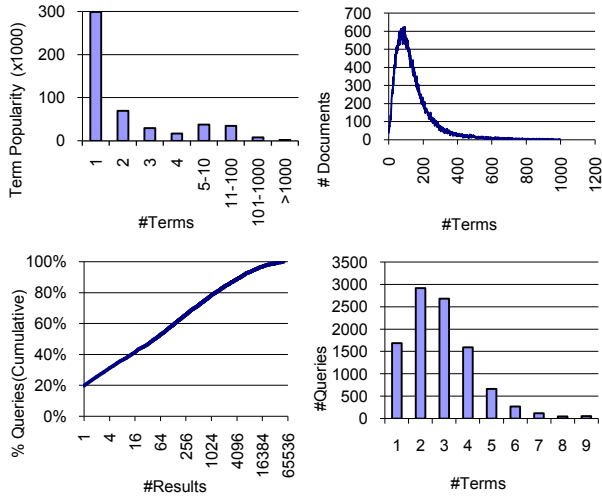


Fig. 4. **Top left:** Keyword popularity. For example, around 290.000 keywords appear only once in the dataset, and around 60.000 appear 2 times. **Top right:** Distribution of the number of keywords per document. **Bottom left:** Number of results per query (cumulative). We can see that for approx. 50% of the queries, we have less than 50 results and for 30% of the queries, we have less than 4 results. **Bottom right:** Number of keywords per query.

- **Corpus** We have used a corpus of 260.000 documents, resulting in the same number of descriptions. Each description consists of a set of keywords.
- **Descriptions** From these descriptions, we have selected a random set of 75.000. On average, each document contained approx. 104 keywords (the distribution is shown in fig.4). The distribution of keywords, as expected, follows a zipf-like distribution(fig.4). It is interesting that more than half of all keywords appear only 1 time, while 1 keyword appears in more than half of the descriptions.
- **Keywords to make the trie** We split the set of descriptions into 25K and 50K, where the keywords in the former set is used to make the trie. As shown in fig. 5(e), for the settings shown below, the trie contains less than 9000 nodes. In this scenario, each node consists of 16 integers along with 16 pointers to other nodes, which totals to 128bytes per node. Without any compression, the total storage required for this trie would be less than 15MB.
- **Keywords to evaluate the methods** The keywords in the remainder description set (50K descriptions) is used to evaluate our methods, which are in total 320K keywords.

B. Results

We performed a series of experiments to gain insight into interesting values for the parameters from our three different options.

In figure 5, we show how many keywords map to each key, for the ‘secure hash’(fig. 5a), the fixed length prefix (fig. 5b) and the information-based prefix (fig. 5c). So, in the X-axis we represent the keys and on the Y-axis the number keywords that map to each of these keys. In the legends we describe the values chosen for the parameters to adjust the privacy/performance trade-off and the used average

number of keywords per key. For each key, a low number of mapping keywords indicates that the privacy for these keywords is not guaranteed. On the other hand, a very high number of mapping keywords indicates that many peers need to be contacted, resulting in low performance. So, ideally, the graphs should be a horizontal line, meaning that for all keys there will be enough mapping keywords to guarantee privacy and few enough keywords to keep performance acceptable. Furthermore, the horizontal line should be vertically adjusted for the desired privacy level. For example, with a horizontal line at $Y=10$, for all keys there are 10 mapping keywords. In figure 5a, we can see that the secure hash approach is closest to the ideal case, the horizontal line. For different key-lengths, one can find the corresponding number of mapping keywords. For example, for a 48bits key, on average 78.22 keywords map for the key and the minimum is 50. In figure 5b we can see that the fixed-length prefix approach has both performance and privacy issues. For example, if 33% of the key length is used it could be that the average of 10.39% mapping keywords is fine, however in +/- 50 cases a key maps to more than 1000 keywords resulting in low performance. Even more important, in most cases it matches to only one keyword which means that privacy is diminished for that corresponding key. Figure 5c shows the results of the information value-based prefix approach. By only looking at the slope of the curve we can already see that the results are much better than the approach with the fixed prefix length. In case of an average of 10.29 mapping keywords per key, the maximum number is 206 keywords and more than 98% of the keys map to more than one keyword. Figure 5d shows the variance of number of mapping keywords per key. As expected, the secure hash approach has the least variance. The information-based prefix approach is an order of magnitude better than the fixed-length prefix approach. Figure 5e shows the number of trie nodes are created for our dataset. When the threshold d on the minimum number of identical prefixes is increased before creating a node, we see that in the beginning there is a steep drop of nodes needed. As in figure 5c, the results are shown for the three different thresholds, the reader now has insight into the balance of the storage size of the trie and the mapping number of keywords per key.

Results concerning keywords alone are not enough to give us insight on what would be the influence on a system with multiple keywords per description. To this end, we have performed numerous experiments to determine the performance of several options on a realistic set of descriptions. To keep complexity to a minimum, we will discuss the results of three comparable parameter settings, one for each option⁴. In figure 5 (f), the distribution of query precision can be seen. The number after each setting represents the average number of keywords per key. Our first observation is that it is possible to obtain roughly the same distribution for all three options. Nevertheless, to achieve this, we had to select a parameter

⁴For the complete set of results obtained during our experiments, the reader is referred to an uncompiled version in <http://www.few.vu.nl/~kot/privateresources/>

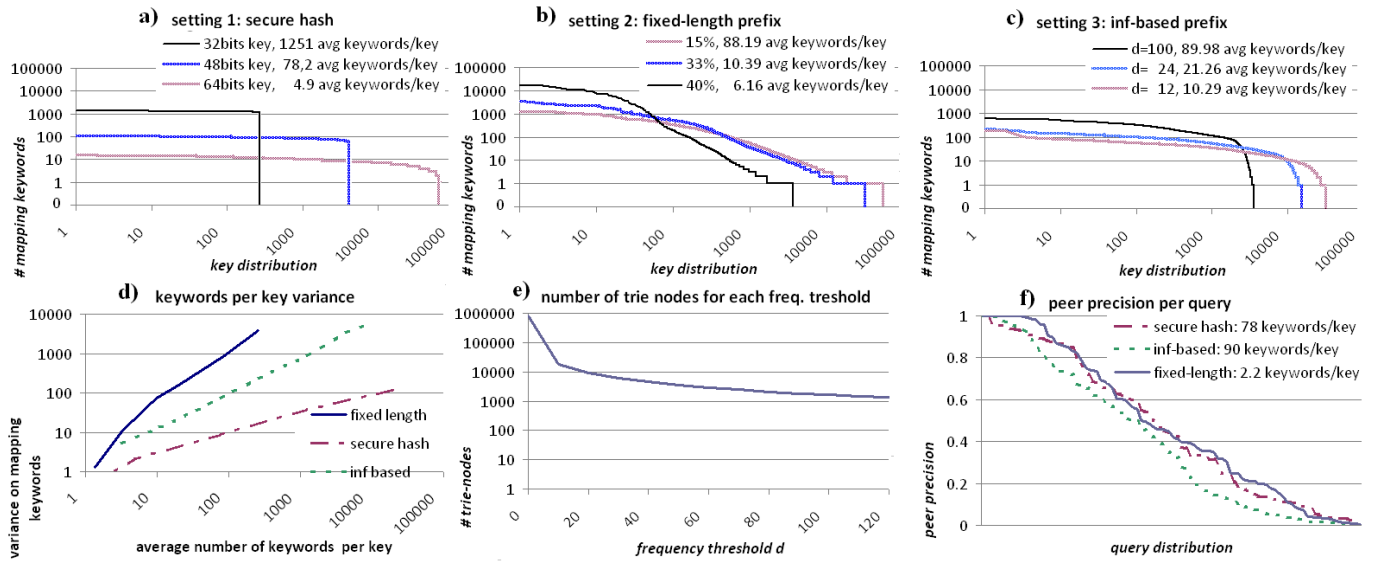


Fig. 5. **a) option 1: secure hash.** The number of mapping keywords for each secure hash key (in our case SHA-1), for different prefix sizes (32bits and 64bits). As can be seen, the lines are very horizontal indicating a small difference between the minimum and maximum number of keywords mapping to the different keys. The legend, e.g. *48bits key, 78,2 avg keywords/key*, should be read as follows: setting with a 48 bits key, where on average 78,2 keywords map to a key. **b) option 2: fixed-length prefix.** The results for the option with fixed ratio length prefixes, i.e. 15%, 33% and 40% of the keywords. As can be seen, the slope is far from horizontal and also many keys map to only one keyword resulting in privacy infringement. **c) option 3: inf-based prefix.** The results showing for three different values of d (=minimum frequency of the prefix in the dictionary), the number of mapping keywords for these keys. As can be seen, compared to *b)* the slope is much more horizontal, together with less keys having only few mapping keywords. **d) keywords per key variance.** This figure shows how the variance on mapping keywords looks like for each average number of keywords per key, where the latter are reached by adjusting the parameters of the corresponding option (fixed length, secure hash and information based). **e) number of trie nodes for different frequency thresholds.** This figure shows how many nodes are needed in the trie to store the keywords, where d is the minimum number prefixes should occur in the dataset before a node is created for it. **f) peer precision per query.** Showing that for each of the three options it is possible to get almost the same query distribution by tuning the right parameter settings.

setting for the fixed length prefixes with a very low average number of keywords per key. This means that, although we can preserve the privacy of complete queries or descriptions, the privacy of single terms can be defeated. In other words, in the case of fixed length keys, in order to maintain the same precision for queries (i.e. preserve query privacy to an equal degree), we need to sacrifice privacy of individual keywords in the descriptions. This is made evident by the fact that a key in the fixed length approach maps only to 2.2 keywords on average. As far as the secure hash and the information-value based prefix prefixes are concerned, we can see that both have acceptable performance with a reasonable number of keywords per key. Finally, note one more problem of the fixed-length prefix: for approx. 7% of queries, precision is 1, which compromises privacy in a large extent.

VII. USE CASE: OPENKNOWLEDGE DISCOVERY

The OpenKnowledge project aims at knowledge sharing through open web-service interactions. One of its main goals is to build a P2P system, which we call the OK-system[6], for sharing knowledge not only in the form of data, but also in the way data is processed⁵. Using this system, people can publish workflow descriptions (also called *Interaction Models or IMs*), and peers can subscribe themselves to play one or

more of the roles in these descriptions. The role-code (i.e. software) that a peer needs to have to play such a role can also be shared and downloaded via the OK-system. Peers can also subscribe themselves to be coordinators of IMs, being responsible for controlling the process flow between the services. All components of the OK-system are implemented in a way that everything runs distributedly without any central control. Sometimes the subscriptions or their constraints need to be kept private, which imposes special requirements on the discovery system. For instance, imagine an interaction model for finding jobs. It goes without saying that it is not acceptable that a subscription to find a new job is visible to all peers in the network, including the current employer.

We will integrate the aforementioned methods in the discovery system of OpenKnowledge to safeguard privacy in subscriptions for participation in interactions. There currently exists a functional implementation of the system, scheduled to be released to the public over the course of 2007.

VIII. APPLICATION DOMAIN

Resources described by a set of keywords are only an example of the application domain of the model, architecture and methods in this paper. In this section, we will elicit the requirements for our methods to be applicable and outline other suitable classes of resources.

The basic requirement is that it should be possible to give approximations of descriptors. As a consequence, the range of

⁵for additional information about OpenKnowledge the reader is referred to: <http://www.cisa.informatics.ed.ac.uk/OK/deliverables.html>

descriptor components should be large. In our experiments, we have used words from web pages, which have an unbounded range. For resources described by boolean attributes, our methods would be ineffective because boolean values contain too little information to be able to hide part of it. For random identifiers, our methods would perform ideally, since it is impossible to infer an identifier from a part of it.

Apart from Openknowledge discovery, possible applications include the following:

- **Discovery of ontologies** New ontologies are being engineered in an ever increasing pace. Considering their development costs, it may be expected that, in the future, they will be considered so valuable assets that organizations may be unwilling to share. Even nowadays, some institutions are reluctant to share their ontologies. Our methods can be used to publish a flattened ontology as a descriptor. Interested parties would be able to retrieve the location of the system managing the ontology and negotiate further to retrieve the entire ontology (or parts of it) and answer queries. This is possible without having to make the ontology public.
- **Image search** Practically all image descriptors (e.g. histograms) can be approximated, for example by rounding or truncating values. It is also possible to approximate image queries in a similar fashion.
- **Resources described by attribute-value pairs** Resources can be described using attribute-value pairs. The methods presented in this paper are directly applicable in this domain.

IX. CONCLUSIONS

In a world where data privacy is becoming increasingly important, and a field (peer-to-peer systems) where little has been done to protect it, we have taken a first step towards developing a scalable privacy-preserving discovery system by outlining the fallacies of current systems, proposing a framework and an architecture based on DHTs and evaluating settings for our proposed methods. Our framework follows a peer-to-peer paradigm, where peers publish descriptions of their resources to an index and are contacted directly by interested peers for additional information. We have proposed the use of a DHT maintained by a network of untrusted peers to maintain this index and three methods to hide the contents of the descriptions from the peers in the DHT. The first method is to use a secure hash, like SHA-1, to generate keys for the keywords in each description. The results show that this method gets very close to the desired 'horizontal line' when looking at the distribution of mapping keywords per key. The disadvantage of this approach is that the hash algorithm does not preserve keyword similarity (e.g. 'computer' and 'computers' hash to completely different keys), which makes approximate matching impossible. The second method we investigated is to use a fixed ratio of the prefix of each keyword as a key. Although we now have a similarity preserving representation, it comes with the price of a high variance on mapping keywords per key because some prefixes are much

more popular than others. The disadvantage of a high variance is that it could result in some cases only very few or only one keyword maps to a key, revealing the intentions of the querier or storing peer. A high variance can also lead to a high network load because when a key maps to many keywords, all peers registered for the key will be returned to the querying peer. The last proposed method calculates the information-value (i.e. popularity) of each keywords and uses it in determining the prefix length. The assumption is that we have access to a representative set of keywords to generate a trie which is an space efficient data-structure to store the popularity of prefixes. The results of the information-based approach indicate that the last method resembles much more the ideal 'horizontal line' than the method using a fixed prefix ratio.

We hope to have provided some insight in the right parameters for these methods and their applicability. Finally, we have an implementation of this system that will be used on the Openknowledge system. Future work lies in analyzing the influence of different types of data (e.g. image histograms or concepts from ontologies) on the performance of the methods presented in this paper.

X. ACKNOWLEDGEMENTS

We would like to thank Frank van Harmelen for making very useful comments and suggestions and Anna Tordai for proofreading this paper. This work was supported by the EU OpenKnowledge Project (IST-2005-027253).

REFERENCES

- [1] "The gnutella protocol specification v0.4," 2000.
- [2] "Universal Description, Discovery and Integration of Business for the Web." [Online]. Available: <http://www.uddi.org>
- [3] "Google search." [Online]. Available: <http://www.google.com>
- [4] "Yahoo." [Online]. Available: <http://www.yahoo.com>
- [5] D. K. Gifford, "Weighted voting for replicated data," in *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP)*, 1979, pp. 150–162. [Online]. Available: citeseer.ist.psu.edu/gifford79weighted.html
- [6] A. P. de Pinninck Bas, D. Dupplaw, S. Kotoulas, and R. Siebes, "The openknowledge kernel," in *International Journal of Applied Science, Engineering and Technology (IJASET)*, 2007.
- [7] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi, "Peer-to-peer resource discovery in grids: Models and systems," *Future Gener. Comput. Syst.*, vol. 23, no. 7, pp. 864–878, 2007.
- [8] "Napster," Web Page. [Online]. Available: <http://www.napster.com/>
- [9] B. Yang and H. Garcia-Molina, "Designing a super-peer network," *icde*, vol. 00, p. 49, 2003.
- [10] R. Siebes and S. Kotoulas, "pRoute: Peer selection using shared term similarity matrices," 2006, to appear in: *Journal of Web Intelligence and Agent Systems*.
- [11] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.
- [12] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350. [Online]. Available: <http://research.microsoft.com/~antr/PAST/pastry.pdf>
- [13] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Ponceva, and R. Schmidt, "P-grid: a self-organizing structured p2p system," *SIGMOD Rec.*, vol. 32, no. 3, pp. 29–33, 2003.

- [14] G. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world," 2003. [Online]. Available: citeseer.ist.psu.edu/manku03symphony.html
- [15] M. S. Artigas, P. G. López, J. P. Ahulló, and A. F. Gómez-Skarmeta, "Cyclone: A novel design schema for hierarchical dhts." in *Peer-to-Peer Computing*. IEEE Computer Society, 2005, pp. 49–56.
- [16] C.-Z. X. H. Shen, A. S. Brodie and W. Shi, "Scalable and secure p2p overlay networks," in *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks* edited by Prof. Jie Wu.
- [17] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala, "Locality-preserving hashing in multidimensional spaces," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1997, pp. 618–625.
- [18] R. Siebes, "pNear - combining content clustering and distributed hash tables," in *Proceedings of the IEEE'05 Workshop on Peer-to-Peer Knowledge Management.*, San Diego, CA, USA, July 2005.