A scalable architecture for peer privacy on the Web

Spyros Kotoulas and Ruud Stegers

Department of Artificial Intelligence, VU University Amsterdam, The Netherlands

Abstract. We consider the privacy implications of exchanging Web data and present the design of a peer-to-peer discovery mechanism with strong privacy guarantees. Its benefits are that it does not rely on a trusted third party, spreads the computational cost among participants and places minimal restrictions on privacy policies. Furthermore, it provides scalability both in terms of system size and level of privacy offered. We outline an implementation that relies on two overlays: a distributed hash table for discovery and an anonymising overlay. Finally, we evaluate the proposed mechanism against a number of privacy threats and analyse its complexity.

1 Introduction

Organizations holding personal information (e.g. government, social networking websites, banks) provide a Web Service where user agents, after authenticating, can retrieve and modify information that concerns them. User agents cannot keep track of the organizations with data that concerns them since the latter is not inserted or maintained by them. *How can these Web Services be located while maintaining the privacy of users and data?* The basic challenges in designing such a system are that the number of such Web Services may be large, thus it is not possible to contact all of them for each query and that user agents and web services should only be able to see data and queries they are authorized to.

Centralized architectures where a single organization maintains an index of all data and enforces access control policies suffers from the following drawbacks: This organization must be completely *trusted* by all participants. Additionally, the **maintenance** of such a gargantuan volume of data would not be an easy, let alone financially viable, task. Finally, the centralized infrastructure should be able to enforce the **access control** and **privacy policies** of the information providers. This inhibits or precludes the use of locally calculated policies and policies that use private criteria. For example, some Web Service wants to authenticate users itself and does not share user passwords with the central index. This is an especially important restriction for the Social Web or any system where we do not have a priori trust agreements [7]. For these reasons, we advocate the use of a peer-to-peer paradigm, where information providers will be able to keep data locally, or at least in the same authority domain and enforce their own privacy and access control policies.

The focus of this work is on *scalable privacy*, with *scalable* referring to both the *size* of the system and the *level* of the privacy guarantees provided. The threat to the privacy of an individual is related to the ability of an adversary to associate information or other individuals with it. Instead of a one-size-fits-all approach, the level of privacy in our method is tunable. We explore the trade-off between privacy and performance ranging from *high performance*, providing similar performance and slightly better privacy guarantees than current resource discovery systems and search engines to *high privacy*, providing anonymity for the participating parties and non-disclosure of the content, its description or a unique identifier for it. Note that the focus of this work is not on privacy and access control *policies*, but on an *architecture* and *mechanisms* that allow a more open set of such policies.

We propose a Peer-to-Peer framework for private discovery and querying of (Semantic Web) data and develop a method to locate data providers without revealing their location, identity or the descriptions of the content. To this end we describe a distributed scalable index, implemented on top of a Distributed Hash Table (DHT). Additionally we show a method to obfuscate and approximate the descriptors in the index, so as to prevent disclosure of resource descriptions and combine it with an anonymising network to protect the identities of the parties involved. We test a selection of the possible privacy settings against a set of possible attacks and analyse their performance, paying special attention to the trade-off between the additional privacy mechanisms and the associated computational and network overhead.

In section 2 we describe the privacy threats covered by our work. Related work is described in section 3. In section 4 we introduce our architecture for scalable privacy, including a description of the technologies we employ to achieve this. Finally, we analyse our architecture in relation to a set of privacy attacks and the associations defined in section 2.

2 Privacy threats

We present the threats within a discovery/sharing system in terms of the ability of an adversary to associate aspects of content (e.g. a descriptor or the content itself) to a user. We use the term *provider* for a peer that shares content and *seeker* for a peer that consumes content. Content may be anything (e.g. RDF data or Social networking profile or a medical record) while a *content identifier* is a unique identifier for some content (e.g. a cryptographic hash of the content or a URI). A discovery mechanism facilitates the sharing process by allowing a seeker to either locate content or potential providers based on a set of *descriptors* (e.g. keywords or derivatives). The discovery mechanism usually maintains an *index* of descriptors to providers.

Table 1 presents possible privacy impairing association. We observe that: (a) Any association in the top cells is detrimental for privacy (e.g. a mapping from *content* to *identity*). (b) It is possible to derive more sensitive information from less sensitive information, moving from the bottom of the table to the top (e.g. an adversary that knows the set of URIs that appear in a SPARQL query can get a good estimate of the query itself). (c) Even perceived insignificant associations can lead to significant privacy breaches (e.g. two users that are associated with

What		**Who**
Content (eg. document)	$\iff$	Identity (eg real name)
Content description (eg. keywords)		
Query (eg. SPARQL query)		Location (eg. IP address)
Query description (eg. set of URIs)		
Content ID (eg. GUID)		Any user in the system
System (usage of)		

Table 1. Privacy impairing associations. This table shows the different associations that can be made between content and identity ordered by descending significance.

similar content IDs have similar interests, thus user profiling is possible). (d) different association types can be combined. For example, an adversary that knows the IP addresses where the system is accessed from (*system - location* association) and has access to an anonymous query log (*query - any user in the system* association). She can infer a (weak) association between queries and IP addresses. We will refer back to this table in section 5.

3 Background

Discovery/sharing systems are widely used both in closed domains and on the public web. Although encryption schemes exist to ensure security, significant challenges concerning privacy and scale remain. Search engines can scale but have abysmal privacy guarantees. Peer-to-peer systems satisfy the basic principle of lacking a central trusted party, but mechanisms to guarantee privacy are not available yet.

We provide an overview of the privacy implications (for seekers and providers) of current discovery/sharing mechanisms in combination with their scalability properties:

Centralized index An index of all content or content descriptions is kept at a collection of centrally managed servers. Information seekers post queries to the index and receive the content, a content identifier or a provider to ask for the content. This can be a *scalable* solution, assuming adequate financial resources to maintain this infrastructure. Nevertheless, it requires that all parties (both seekers and providers) trust this central index with their privacy, since it has complete control and access to all content and queries. Furthermore, providers should entrust the index to correctly enforce the policies specified with the data (if any).

Broadcast-based unstructured overlays Unstructured overlays are a type of peer-to-peer overlay where peers maintain a set of ad-hoc connections to other peers [4]. Queries are broadcasted to all peers, or a significant subset of peers in the system. They provide good privacy guarantees concerning the information providers, since there is no index that can be used to derive privacy sensitive information from. Additionally, every provider maintains full control of it own data. The privacy of the seekers on the other hand is not protected since their queries can be seen by any peer in the network.

Broadcast-based systems are *not scalable* since queries need to be sent to a large subset of the total peers in the system.

Structured Overlays Structured peer-to-peer overlays impose a global structure on the peer connections [6]. They can be used to implement efficient and scalable indexes spread over a large number of nodes. They are *very scalable* but fare *poorly with provider privacy* since the index is readable by a large number of possibly untrusted nodes, which can be used to derive privacy sensitive information about the provider. Any single compromised index node is a direct threat to privacy of the seekers that uses it, since the queries are exposed..

Confidential indexes Confidential indexes aim at hiding information from the indexed content. The general pattern is to introduce noise to the results in order to avoid strong associations. We describe two recent implementions of confidential indexes that are very relevant to our work.

The confidential index proposed by Bawa et al [1] relies on a public index that divides the providers over a set of privacy groups. Within the group, they collaboratively create a bloom filter with the descriptors of their content using a randomized process so as to avoid exposure of the content of individual providers. These filters, along with the list of all providers in the group are sent to a central server, where an index is created that maps bits in the bloom filter to privacy groups. The privacy of the providers is protected since the privacy groups contain providers that have or do not have the requested content (introduction of false positives).

Zerber [8] uses a set of largely untrusted servers to maintain an inverted index of descriptors. Access control is enforced on the index. To defend against compromised index nodes, no single index server is given enough information to reconstruct a descriptor by itself. To this end, descriptors are encrypted using k out of n cryptography. This means that every descriptor is split and encrypted into n different parts, each owned by a different index node. Since every index node enforces access control locally, as many as k compromised index nodes are required for an adversary to obtain a complete descriptor. For additional protection, the index maps sets of terms instead of single terms to document descriptors (called a merged posting list mechanism). The decrypted descriptors contain the original index terms (so the irrelevant entries returned as result of the merging can be filtered out) and the location of the document. Zerber^{+R} [9] additionally provides top-k ranked results.

Our own method falls into this category and is further described in section 4.1. The major improvement over these systems is that our system additionally provides anonymity and can scale to a much larger index, since it is maintained by system participants.

Encrypted indexes Encrypted indexes protect the content by encrypting sensitive parts of the index entries. Since this generally requires complex key management schemes, scalability is a problem.

4 Our Architecture

We are using an index-assisted peer-to-peer model similar to the one in [1]. To have their information indexed, providers *(a')* extract a content descriptor (e.g. keywords, hashes of keywords, ...) and *(b')* store it in the index together with an identifier to contact the provider. To locate and access content, seekers need to take the following steps: *(a)* they create a set of content descriptors based on their query (e.g. a set of keywords, hashes of keywords, ...), *(b)* they send this set to the index which in turn *(c)* returns a number of providers which are (possibly) able to provide the information. *(d)* The seeker selects a number of these providers based on local preferences and *(e)* negotiates directly with the provider to retrieve the requested information. Our method places no restriction on the privacy and access control policies for step (e), except that providers need to be able to enforce them themselves locally. The focus of this work is on defining a scalable architecture for this model and develop *mechanisms* to guarantee privacy and anonymity.

4.1 Scalable architecture

We will outline a distributed and scalable implementation. The meaning of the word scalable is two-fold: First, the performance of the system does not deteriorate severely as the number of participants increases. Second, different levels of privacy are supported. The infrastructure that will be presented functions through the synergy of two peer-to-peer overlays: the *Indexing overlay* and the *Anonymising Overlay*. The former provides a global scalable index on top of a DHT. The latter provides a distributed and scalable mechanism to hide the identity of peers, whenever this is required.

Index-assisted query routing The *indexing overlay* is responsible for providing mappings from the set of descriptors in a query to a set providers that possibly have content that matches these descriptors. It should be able to store a very large number of descriptors, thus a scalable implementation is required.

Distributed Hash Tables(DHTs) are a type of structured peer-to-peer overlays that impose a global structure on the peer connections [6]. 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 [6].

DHTs are well suited for implementing a large, global index, maintained by the participants of the system. This is, in fact, straightforward: for every element of the descriptor, peers insert in the index a $\langle descriptor, peer{-}address \rangle$ pair. For querying, seekers make one search for each of the query descriptors to retrieve the relevant providers.

The performance gain aside, using a DHT to maintain the index shifts the responsibility from one organization to multiple organisations, which is both a blessing and a curse: the index is no longer held in one location, meaning that no single entity has complete control of the entire index. On the other hand, the fact that it is partitioned means that many entities have control over *parts* of it. Obviously this has consequences for privacy.

Hiding the descriptors Since we do not trust the index nodes, we cannot index the content descriptors in plaintext. We identify some methods to reduce the information conveyed in the indexed descriptors and make them non-understandable to prevent direct association of content and query descriptors to users or locations.

Obfuscation Instead of indexing the descriptor itself, we index its secure hash (e.g. using SHA-1) using hash function h. For example, for the descriptor $\langle$"*Soccer*"$\rangle =$ 134.23.32.2, we will store $\langle h(\text{"}Soccer\text{"})\rangle = 134.32.42.2$. When querying, we use the same hash function for the query descriptors. Note that secure hashes are a one-way function, so it is very difficult to retrieve the value that was hashed to produce a certain result.

Approximation It is possible to perform a dictionary attack by calculating the hashes of all descriptors the attacker is interested in and matching them to the indexed descriptors. To alleviate this problem, we introduce false positives in our results. Let $P(t)$ be the set of providers that actually have content with descriptor t and $P'(t)$ the false positives. We define exposure for an index entry with descriptor t as the ratio between the number of true positives relative to the total number of answers $e_t = \frac{|P(t)|}{|P'(t)|+|P(t)|}$, a low e meaning that the providers for t have low exposure (high privacy) and $e = 1$ that they are fully exposed.

This is implemented by varying the size of the hash, so as to increase collisions. In previous work [5], we have shown that decreasing the length of the hash leads to a proportional increase in privacy. If we truncate hash values to l bits, any query would match a portion of $r = {}^{1}/_{2^l}$ of all descriptors in the system. This is directly translatable to an average exposure of

$$e = \frac{1}{2^l} \tag{1}$$

For maximum privacy, l will be equal to 0, i.e. any query would match all descriptors stored in the index and return all peers in the system. For maximum performance, 2^l should be much higher than the total number of descriptors in the system, so as to completely avoid collisions. In [5]we have also shown that using an SHA-1 secure hash we can guarantee the privacy of individual terms for a typical webpage description corpus.

Anonymising network Hiding the descriptors in the index is not enough: identities and locations can be used to perform association attacks (see table 1 and section 5). To provide stronger security guarantees, it is desirable to hide the identity of the peers in the network. Onion routing emerged from this general wish [2]. Onion routing anonymises communication channel and protects the identities and locations of the participants.

In onion routing, a set of nodes forms an anonymising routing overlay. The key principle, as described by Chaum[2], is that every router over which a packet is send is only aware of the identity of the previous router and the next router. This ensures that a single trusted router is in essence enough to protect the total path. This is achieved using layered encryption. Starting from the destination router, the peer will add the address of the router A_n to the data, and encrypt it with the public key K^+_{n-1} of the router before that in the routing chain. The resulting package is an onion containing several layers of encrypted addresses and data which at each level can only be read by the router with the correct private key:

$$\left[A_1, \left[A_2, [P]_{K^+_2}\right]_{K^+_1}\right]_{K^+_0}$$

When a router receives a packet, it will decrypt it with its own private key, and use the now readable address of the next router to send the contained (encrypted) data part to the next node. The destination will find the original payload when decrypting the last layer. The simplified scheme described here has been extended with nonces, symmetric keys and other mechanisms to properly ensure robust and scalable anonymity[3].

The original onion routing proposal defined 'return envelopes' to protect the identity of the client from the server. However, this method does not scale well. Next generation implementations provide other means to publish anonymously, e.g. the *Hidden Services* in the TOR project, which hide the identities both of the server and the client.

We can use this mechanism to provide volatile peer identities, i.e. *pseudonyms*. Instead of publishing $\langle descriptor, ip\text{–}address\rangle$ pairs on the index, it is now possible to publish $\langle descriptor, pseudonym\rangle$ pairs. After a seeker has obtained a psuedonym, it will use the anonymising network to contact the provider to retrieve the content anonymously. This will protect the privacy of the provider by making ***association*** of content to a ***real-world identity*** or ***location impossible***.

In order to protect the descriptor from statistical correlation attacks, it is possible for a provider to publish using multiple pseudonyms. We will expand on this in section 5.

5 Analysis

Here we define four classes of settings based on different combinations of the techniques discussed above.

1. ***Minimal*** A DHT is used as an index. Keys are extracted from content and queries. They are published unmodified and associated with the (IP) address of the information provider. This setting protects associations with the content and the query, since only their descriptions are published or queried.

2. ***Obfuscate and approximate*** The key is obfuscated using a secure hash function. The level of privacy and the performance impact are inversely proportional to the length of the hash l. This setting addtionally protects query and content descriptors.
3. ***Anonymise*** The key is published unmodified but the identity of the information provider and the seeker is concealed using an anonymising network. Instead of a unique address, a pseudonym to contact the peer is published. The level of privacy can be chosen by the provider at indexing time, in the choice of how many terms are connected to a single service descriptor. Compared to setting 1, this setting additionally protects associations with identities or locations.
4. ***Full*** The key is obfuscated and an anonymising network is used. This setting offers the highest privacy guarantees. Both the level of obfuscation and anonymity can be chosen. This settings protects against all associations in table 1, except for the association that some unknown user is using the system and the association that the system is used by some location. Even though it is not significant privacy-wise, even these last associations could be prevented by using a public anonymising network (e.g. [3])

Attacks We analyse the susceptibility of the aforementioned approaches to a set of common attacks. We use table 1 as grounding to describe the consequences of successful attacks.

A1: Compromised index An adversary can seize control of the index. In our approach, this is possible by compromising the DHT (how this is possible is implementation-dependent and beyond the scope of this paper). The privacy implications for a read-compromised index are negligible: the data in the index was already readable for everyone in the first place, thus no extra information is gained by taking over one or more index nodes. All settings are equally susceptible to compromising the DHT. For the rest of our analysis, we will assume that the DHT is read-compromised, i.e. that the full index is readable by the adversary.

A2: Compromised anonymiser Anonymity is provided only while the anonymising overlay is not compromised. Low-latency anonymisers are highly desirable for performance reasons, however they generally are susceptible to end-to-end communication correlation attacks [3]. Nevertheless, in order to perform this attack, the adversary needs to be able to eavesdrop on a large part of the network. Generally speaking, any compromized communication channel in setting 3 and 4 reduces the privacy for the peer(s) involved in that communication to the same level as in setting 1 and 2 respectively. The degradation of privacy is gradual and dependent on the power of the adversary to observe communication channels and the size of the system. Approaches exist to reduce these risks [3].

A3: Taking the intersection of the returned providers for correlated descriptors An attacker may exploit the fact that descriptors from the same provider are correlated. By posting several related queries and taking the intersection of the sets of returned providers, the attacker can associate provider locations or identities with descriptors.

For a set of n queries with one descriptor $t_1 ... t_n$ each , the intersection of the sets of peers returned by the system is:

$$\begin{gathered}\cap_{i=1}^{n} P^{*}\left(t_i\right) = \cap_{i=1}^{n}\left(P'\left(t_i\right) \cup P\left(t_i\right)\right) = \\ \left[P\left(t_1\right) \cap P\left(t_2\right) ... \cap P\left(t_n\right)\right] \cup \underbrace{\left[P\left(t_1\right) \cap P\left(t_2\right) ... \cap P'\left(t_n\right)\right] \cup ... \cup \left[P'\left(t_1\right) \cap P'\left(t_2\right) ... \cap P'\left(t_n\right)\right]}_{2^n-1}\end{gathered}$$

the set $P\left(t_1\right) \cap P\left(t_2\right) ... \cap P\left(t_n\right)$ represents all true positives for the intersection, while the $2^n - 1$ conjunctions represent the false positives, since each conjunction contains at least one P'. From previously given definition of exposure (formula 1), we have that:

$$e = \frac{\left|P\left(t_1\right) \cap P\left(t_2\right) ... \cap P\left(t_n\right)\right|}{\left|\left[P\left(t_1\right) \cap P\left(t_2\right) ... \cap P\left(t_n\right)\right] \cup ... \cup \left[P'\left(t_1\right) \cap P'\left(t_2\right) ... \cap P'\left(t_n\right)\right]\right|}$$

We will consider the worst-case scenario where the adversary somehow knows that descriptors $t_1 ... t_n$ are completely correlated in the index, i.e. they appear exactly on the same providers. Thus, $P\left(t_1\right) = P\left(t_2\right) = ... = P\left(t_n\right)$. In this case, $P\left(t_1\right) \cap P\left(t_2\right) ... \cap P\left(t_n\right) = P\left(t_1\right)$ and

$$e = \frac{\left|P\left(t_1\right)\right|}{\left|\left[P\left(t_1\right)\right] \cup \left[P\left(t_1\right) \cap P'\left(t_2\right)\right] \cup ... \cup \left[P'\left(t_1\right) \cap P'\left(t_2\right) ... \cap P'\left(t_n\right)\right]\right|}$$

In the denominator, all conjunctions with $P(t_1)$ will be absorbed by the disjunction with $P(t_1)$. Thus, $e = \frac{\left|P\left(t_1\right)\right|}{\left|P\left(t_1\right) \cup \left[P'\left(t_1\right) \cap P'\left(t_2\right) ... \cap P'\left(t_n\right)\right]\right|}$. We also have that $P\left(t\right) \cap P'\left(t\right) = \emptyset$, thus

$$e = \frac{\left|P\left(t_1\right)\right|}{\left|P\left(t_1\right)\right| + \left|P'\left(t_1\right) \cap P'\left(t_2\right) ... \cap P'\left(t_n\right)\right|}$$

The attacker has no control over the false positives $P'\left(t_1\right) \cap P'\left(t_2\right) ... \cap P'\left(t_n\right)$, which depend only on the properties of the dataset. Nevertheless, since $A \supseteq A \cap B$, the size $\left|P'\left(t_1\right) \cap P'\left(t_2\right) ... \cap P'\left(t_n\right)\right|$ will decrease as the number of conjunctive terms increases. On the other hand, it becomes increasingly difficult to find a larger set of correlated descriptors.

Note that this attack is not possible in settings 3 and 4 since the providers are using pseudonyms. It is enough to publish each correlated descriptor under a different pseudonym to leave the adversary with no additional information about the set $P\left(t_1\right) \cap P\left(t_2\right) ... \cap P\left(t_n\right)$. If it is not possible for providers to know which terms are correlated, they can publish every descriptor under a different pseudonym.

The vulnerability of setting 2 depends on the length of the hash l (since the ratio of true positives $P\left(t\right)$to false positives $P'\left(t\right)$ is dependent on l).

A4: Malicious provider tries to profile seekers A compromised index may collude with a provider to profile seekers based on their searches. Similar to the previously described attack, but profiling using queries instead of content thus attacking the privacy of the seeker instead of the provider. Seekers may prevent this by using several index nodes.

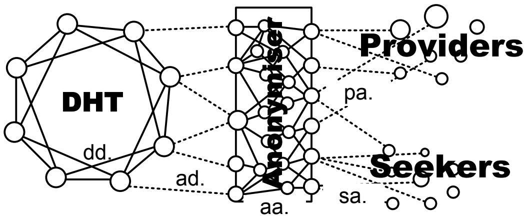

Fig. 1. Communication channels

A5: Censorship and denial of service by index nodes Though strictly speaking not a privacy attack, censorship and denial of service are related attacks that can be prevented by providing peer privacy.

A malicious index node may *censor* all descriptors that correspond to a given descriptor, by not returning any answer to queries for those. In settings 1 and 3, this is easy for any subject, since the descriptors are stored in plaintext. In settings 2 and 4, this is more difficult: since multiple descriptors map to the same hash values, the malicious node would have to censor all of them, since there is no way of knowing which ones refer to the given descriptor. The number of censored descriptors would be $1/2^l \cdot N$, where l is the length of the hash and N the total number of descriptors in the system. Detecting such a malicious index node is easy: it is enough to create some honeypot providers advertising descriptors that are likely to be censored and match results returned by the suspect index node.

Denying services to a given seeker or provider is possible in settings 1 and 2: a seeker could be presented with wrong or limited results, and the index could refuse storing the descriptors for a given provider. In settings 3 and 4, this is not possible, since providers use pseudonyms and seekers are anonymous. Assuming that malicious nodes also deny service to particular provider pseudonyms, the level of protection depends on the number of pseudonyms per provider.

Performance We will analyse the performance of our architecture focusing on the trade-off between privacy and performance. Furthermore, we will focus on its scalability in terms of network size, index size and the throughput of the anonymiser.

In figure 1, we can see the communication channels in our system. Note that the circles represent *roles* fulfilled by a physical host. E.g. a single physical host may be part of the DHT, part of the anonymising network and a content provider. In fact, we will assume that the number of hosts participating in the DHT and the anonymiser increases linearly with the number of providers and seekers.

We have the following communication channels: *dd* and *aa* for communication among DHT nodes and anonymiser nodes respectively, *ad* for DHT nodes and anonymiser nodes, *sa* for seekers and anonymiser nodes and *pa* for providers and anonymiser nodes. The cost associated with sending one message through a communication channel is defined as $c_{ChannelName}$. E.g. the cost of doing a DHT search is c_{dd}.

Considering that DHTs and anonymiser networks can have an arbitrary number of contact points (or entry nodes), channels ad, pa and sa will not be congested. The probable scalability issue may lie within the DHT network maintaining the index(dd) or the anonymiser network(aa).

The communication resulting from a query with x descriptors is as follows: The seeker will divide it to x sub-queries with one descriptor each. Each of these will have to be routed through the anonymiser to reach the DHT, where a lookup will take place. The DHT will return s providers through the anonymiser, with a ratio r of false positives. The protocol the seeker will use to negotiate with the providers is beyond the scope of our approach. We will assume that negotiation has a cost of c_n.

Thus, the cost associated with the query is:

$$c_q = x \cdot (c_{sa} + c_{aa} + c_{ad} + c_{dd} + c_{aa} + c_{sa}) + s \cdot c_n \cdot (c_{sa} + c_{pa})$$

A typical DHT lookup cost, in terms of IP messages is $O(c_{dd}) = O(\log(N_{DHT}))$, where N_{DHT} is the number of nodes in the DHT and a typical anonymiser routing cost is $O(c_{aa}) = O(1)$. Sending messages over channels sa, ad and pa has a cost of one, since they are directly on top of the underlying network protocol. From these, we can bound the total cost of a query to:

$O(c_q) = O(x \cdot (c_{sa} + c_{aa} + c_{ad} + c_{dd} + c_{aa} + c_{sa}) + s \cdot c_n \cdot (c_{sa} + c_{pa})) = x \cdot [2 \cdot O(1) + 2 \cdot O(1) + O(\log(N_{DHT}))] + 2 \cdot s \cdot c_n \cdot O(1) = O(x \cdot \log(N_{DHT}) + 2 \cdot s \cdot c_n)$

From the definition of exposure and equation 1, we have that $e = \frac{a}{s} = \frac{1}{2^l}$, thus, $s = a \cdot 2^l$ where a is the average number of providers that match a query(true positives). We consider the cost of the negotiation constant, thus $O\left(c_n\right) = O(1)$. Then, an upper bound for query cost is:

$$O(c_q) = O(x \cdot \log N_{DHT} + a \cdot 2^{l+1})$$

where x is the number of descriptors in the query, N_{DHT} is the number of nodes in the DHT, a is the number of providers that match the query and l is the length of the hash.

It is interesting to note that the cost for using the anonymiser does not increase the overall complexity for querying. Moreover, the size of the DHT, and thus the size of the index, increase only logarithmically, indicating good scalability. The size of the hash l also plays an important role, signifying a trade-off between privacy and scalability. The limiting factor in our system is the number of matching peers a, since it is expected to grow linearly with the number of providers in the system. This makes the need for ranking in the index evident, but we will have to leave this for future work.

The cost of indexing consists of merely indexing every descriptor using the anonymiser. Due to space restrictions, we only present the final result:

$$O(c_i) = O(x \cdot \log N_{DHT})$$

where x is the number of descriptors and N_{DHT} the number of nodes in the DHT. This clearly scales well.

6 Conclusions

As pointed out in [7], privacy control has moved well beyond settings where an access control list or group access control would apply. Parties are no longer known in advance and privacy policies and trust have to be calculated on-the-fly. In an open (Semantic) Web, information sharing consists of *locating* and *acquiring* the data. Most privacy control approaches have focused on the latter. Our approach focuses on the former, providing an open discovery infrastructure where providers and queriers reveal minimal information. We have shown that our design is scalable in terms of system size and privacy level provided and that associations between aspects of content and identity can be protected. Furthermore, we have explored the trade-off between privacy and performance and analysed the message complexity for querying. A general recommendation about which setting to choose cannot be given, since it depends entirely on the requirements of an application and costs involved.

Future work includes implementation of this approach as a Sesame plug-in and integration of privacy policies.

We would like to thank Eyal Oren for the fruitful discussions and pointers and Frank van Harmelen for reviewing our work. This work is supported by the European Commission under the LarKC project (FP7-215535).

References

1. Mayank Bawa, Roberto J. Bayardo Jr., and Rakesh Agrawal. Privacy-preserving indexing of documents on the network. In *In Proc. of the VLDB*, pages 922–933, 2003.
2. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.
3. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *In Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.
4. G. Kan. Gnutella. In A. Oram, editor, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, pages 94–122. O'Reilly and Associates, 2001.
5. Spyros Kotoulas and Ronny Siebes. Scalable discovery of private resources. In *IEEE SECOVAL at SECURECOMM*, Nice, France, 2007.
6. Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, pages 72–93, 2004.
7. Alexandre Passant, Philipp Kärger, Michael Hausenblas, Daniel Olmedilla, Axel Polleres, and Stefan Decker. Enabling trust and privacy on the social web. In *W3C Workshop on the Future of Social Networking*, Barcelona, Spain, January 2009.
8. Sergej Zerr, Elena Demidova, Daniel Olmedilla, Wolfgang Nejdl, Marianne Winslett, and Soumyadeb Mitra. Zerber: r-confidential indexing for distributed documents. In *11th International Conference on Extending Database Technology (EDBT 2008)*, volume 261 of *ACM International Conference Proceeding Series*, pages 287–298, Nantes,France, March 2008. ACM.
9. Sergej Zerr, Daniel Olmedilla, Wolfgang Nejdl, and Wolf Siberski. Zerber+r: Top-k retrieval from a confidential index. In *12th International Conference on Extending Database Technology (EDBT/ICDT 2009)*, ACM International Conference Proceeding Series, Saint-Petersburg, Russia, March 2009. ACM.