Chapter 7

A Search Network

"By believing passionately in something that still does not exist, we create it. The nonexistent is whatever we have not sufficiently desired. "
Nikos Kazantzakis

7.1 Introduction

We have evaluated each of the three remaining PET categories that showed potential to deliver search privacy. Whilst all three do show promise in so far as enhancing the control one has over one's search profile, neither has eliminated the search engine as a threat to search privacy.

In chapter 3 we discussed the privacy violation that occurred when AOL released a portion of their search logs to the public. In trying to determine the point at which the privacy violation occurred, we concluded that since the emphasis of search privacy is control over one's own information then search privacy was lost the moment each user started searching on AOL's search engine. Once the user issues queries to the search engine, a search profile can be constructed and controlled exclusively by the search engine. Since control of this profile lies in the hands of the search engine, the search user has lost search privacy.

This does not mean search and search privacy are mutually exclusive. Consider the following:

- We have a search query
- We want a search result

We could submit the query to a search engine, but this would eventually lead to a violation of search privacy. If we move the context of this scenario away from search and consider it from the perspective of the Web alone, then we have a query (Web request) for which we want a result (Web response). To alleviate bandwidth contention, Web proxies often facilitate this scenario. One Web request is essentially submitted on behalf of a first caller. The Web response is then cached and served for subsequent callers of the same request. If we can apply this to search in a manner that does not result in the violation of search privacy at the proxy, then the user would be spared from submitting all search queries to the search engine and search privacy would be a reality.

In this chapter we use the database involved in the AOL privacy violation of 2006 to confirm that the search queries of users are not necessarily always mutually exclusive, i.e., users are sharing queries. With this in mind, we outline the basic architecture of a search network with the primary objective of providing search privacy in a manner that is fast and scalable.

7.2 A Case for Sharing

In their analysis of the findings from nine Web search studies, Jansen and Spink [71] point out that the results of one search engine study "cannot

be applied wholesale to all search engines." In this section, we confirm the findings of Markatos [83] from the analysis of the Excite search engine logs: search queries have a significant amount of temporal locality (shared queries).

7.2.1 Analysis of Search Data

Of the 650,000 search profiles available in the AOL database, we picked a random sample. This was made up of 65,517 profiles with a combined total of 3,558,412 queries (t). Figure 7.1 depicts the number of times that each of the top 1,000 queries was used.

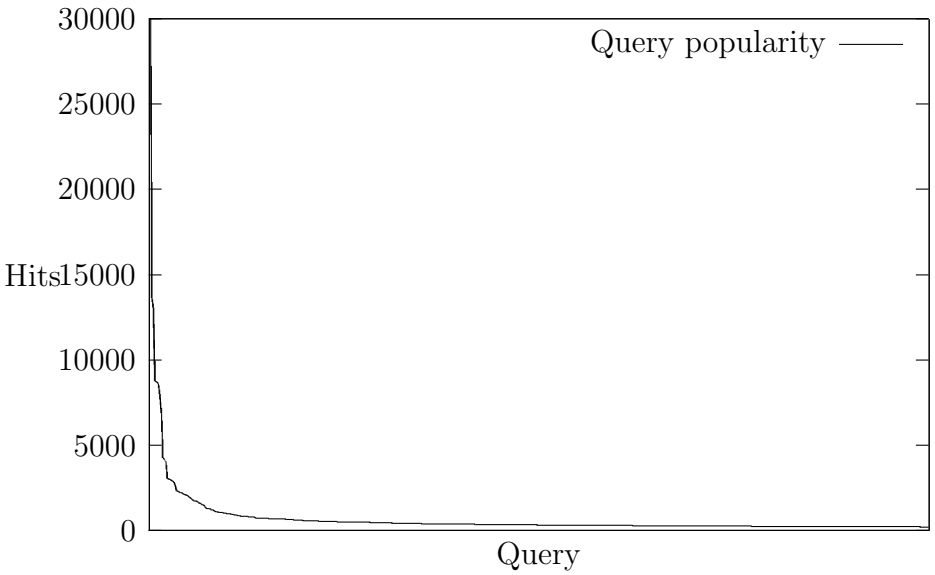

Figure 7.1: The top 1,000 queries ranked in descending order.

A unique query (Q_u) is a query that was issued only once in the sample. A shared query (Q_s) is defined as a query that has been issued more than once by any user/profile in the sample. Query 1 (Q_1) is said to be unique if it has only been issued once by a single user in the system. Q_2 on the other hand, is said to be shared if user x has used it multiple times, or if user x and user y have each used it once or more times. The rapid fall and long tail depicted in figure 7.1 hints at there being far more unique queries than shared. Further analysis of the data shows that this is indeed the case: $n(Q_si) < n(Q_uj)$ where Q_si is the set consisting of all shared queries, Q_uj the set of all unique queries and n the function returning the number of elements in a set. Specifically $n(Q_si) = 479,688$ and $n(Q_uj) = 736,967$.

Figure 7.2 depicts the relationship between users and the types of queries that they have submitted. Each user has two points of interest: (1) the number of shared queries he/she issued in black ($n(Q_si)$ for the user) and (2) the number of unique queries he/she issued in gray ($n(Q_uj)$ for the user).

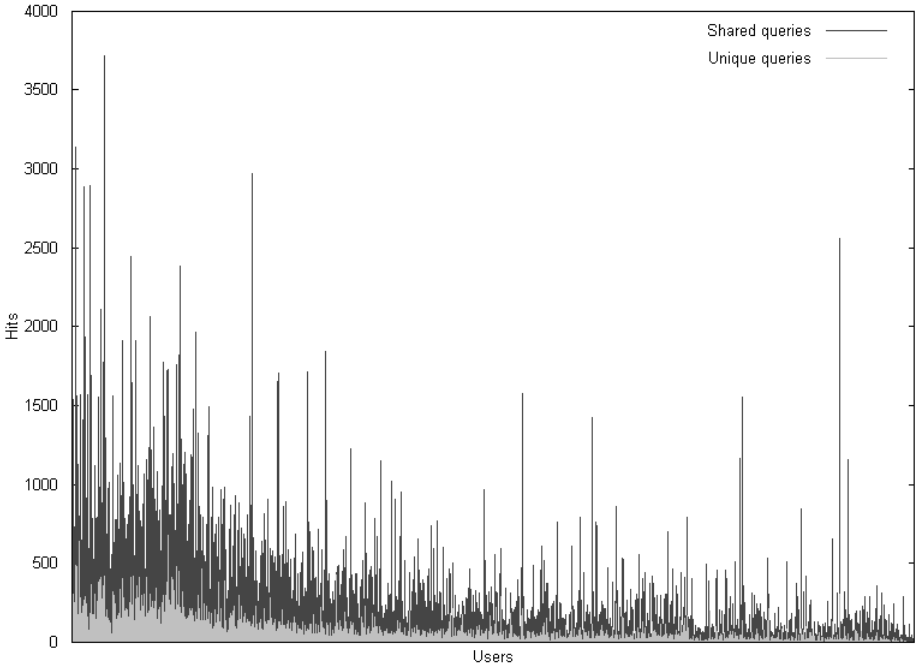

Figure 7.2: Each user is plotted against the number of shared and unique queries that he/she submitted.

Despite the apparent contradiction, the large amount of shared queries is consistent with our previous observation of $n(Q_si) < n(Q_uj)$. This may not seem initially evident but must certainly be the case since we know that there were a total of approximately 3.5 million queries (t) issued. If 736,967 of those queries were unique then the remaining Q_si queries must make up the difference. What is now clear is that although $n(Q_uj)$ is almost double the amount of $n(Q_si)$, if each Q_s were treated as a separate query, then shared queries are far more popular than unique queries: $n(Q_si) = t - n(Q_uj) = 2,821,445$.

It may be the case however that each user is simply reissuing their own queries. If this is so then a network which depends on sharing query results amongst its users would be useless since there would not be many queries to share.

Figure 7.3 depicts the number of different users that used each query.

The figure shows that at least 50% of the shared queries were used by two or more different users: $0.5 * n(Q_s i) = 1,410,723$. Since this is almost half of the total number of queries issued, we believe this makes an excellent case for a network dependent on sharing search queries.

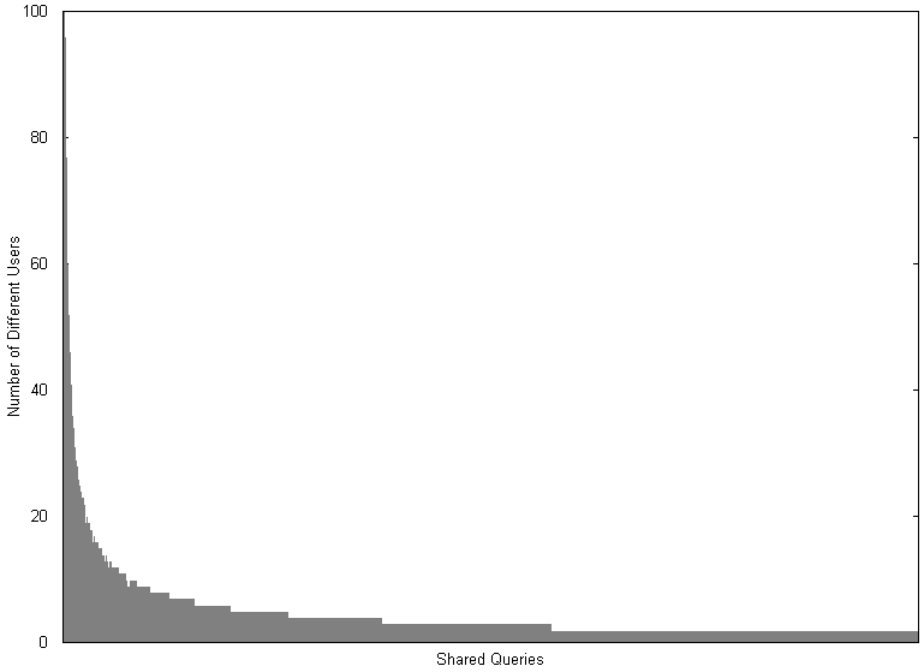

Figure 7.3: In this figure the number of users that used each shared query is plotted.

7.3 A Search Network

We have shown that there is a surprisingly large number of shared search queries. The network we propose leverages off of this fact in a bid to better protect the privacy of its users. In this section, we discuss a high level design of the network. This is achieved through an analysis of its requirements:

1. The network must be scalable. The number of search queries per day for a single search engine may be in the hundreds of millions [25]. A network which aims to operate as a source of search query results for multiple search engines across the Web must scale to accommodate a staggering number of users and queries.

2. The network must be fast. A search query conducted through the network should be comparable to querying the search engine directly.

3. The network must protect the privacy of its users not only from the search engines, but from each other. We must assume that not all users of the network are trustworthy.

7.3.1 A Scalable Network

A centralised search network would not lend itself well to massive scalability unless tremendous costs were incurred. From a privacy perspective, this approach would also mean a single point of vulnerability. A decentralised approach on the other hand, may be far more cost effective in addition to not forcing users to trust a single entity.

A decentralised and scalable approach is that of P2P networking. There are a number of P2P networks which exhibit scalability [2, 127, 105, 113, 54]. Most of these networks are based on the notion of a Distributed Hash Table (DHT) [11].

A DHT is essentially a decentralised approach to storing and retrieving data. Load in a DHT is distributed in a manner that facilitates change in the set of participants with minimal impact to the network itself. Participants in a DHT employ a consistent hash function [74] so as to establish a keyspace partitioning scheme amongst themselves that balances the load fairly.

From a very high level perspective, a DHT offers two sets of functions:

1. $put(k, data)$: k represents a hash of the key used to index $data$ into the DHT. When putting data into the DHT, the key and the data will *hop* through nodes of the DHT until the node responsible for that key is reached (this is determined by the globally known hashing function), it is this node where the data will be stored.

2. $get(k)$: returns the $data$ indexed by k.

If we briefly examine the search process, a search network built upon a DHT seems to be a logical fit: a user issues a query ($get(k)$) to a search engine which responds with a search results page ($data$). For the sake of simplicity, we only store the first page of search results. Note that Jansen and Spink [71] highlight the trend of more search users viewing only the first page returned from a search query. This finding was initially highlighted by Beitzel et al [21] in their analysis of a week's worth of search query logs from approximately fifty million AOL users.

The network we propose would act as a scalable storage depot operating in conjunction with the basic search process. Once a search results page

has been retrieved from the search engine it would be stored in the DHT ($put(k, data)$) so as to save other users from having to query the search engine in the future.

7.3.2 A Fast Network

Besides the benefit of being scalable, DHTs offer the advantage of only needing a small number of hops through neighbouring nodes in order to satisfy requests. The order differs across varying implementations of DHTs but it is typically $O(log(n))$ or better (where n is the number of nodes participating in the DHT). Unfortunately, a disadvantage of the quick lookup scheme using the hashing function means that only exact matches for k can be returned.

7.3.3 A Privacy Preserving Network

As the reader has probably inferred from the discussion thus far, the proposed search network works as follows:

- Results of queries submitted to a search engine are stored in a DHT.
- Users wishing to make the same queries (this has been shown to be a likely occurrence) have the option of retrieving the results from the DHT instead of making a submission to a search engine.

At this point, the contribution of the network is that of disconnecting searchers from the search engine. We assume that search queries and their associated results are not personalised. This assumption means that there is a one to many relationship between a result for a query and the number of consumers of the result, i.e., a result for one user can be provided as a result for another user submitting the same query.

Unfortunately, this disconnection from the search engine is not the only concern. Since there may be users of the network that cannot be trusted, the proposed solution must provide all users with a degree of privacy from each other. In a bid to maintain the scalability and speed of the search network, we adopt a simple approach.

Since the users will be relying on each other to retrieve and store results, anonymity within the network is not our goal, i.e., the network does not employ any anonymisation techniques in so far as *who* is searching for data. Privacy is provided in the sense of *what* is being searched for. We know that the DHT stores a search result ($data$) for the hashed version of a query (k). At the very least, we will not store the query or the search result in plain text. Since k is a result of a one way hashing function this only leaves

the problem of ensuring that *data* cannot be observed. In the absence of a centralised authority or third party we propose an encryption scheme that uses the plain text version of the query (q before it is hashed) as the key when encrypting the result: $data = E_q(result)$. Since q will form part of the one way hash, the search results will only be accessible to those who already know what they are looking for, in other words, they know q.

At this stage, we have only dealt with storing the results of a query in the DHT. We have discussed a solution where participants of the network are disconnected from the search engines and are offered a degree of privacy from each other. Unfortunately, the privacy introduced in this section brings with it another problem: if the query is hashed and unobservable in plain text, then who is going to submit the query on behalf of a user when a cached result does not exist in the DHT? After all, the search engine cannot be queried with a hashed version of the query.

7.3.4 Submitting a Query

Submitting a lone query to a search engine is not as much of a problem as submitting a number of queries over time since a search engine may construct a sequential log (search profile) of the user's queries which may be used to infer something private about him or her.

A search profile consisting of other users' queries contributes to the privacy of the user in question since each query has lost its context. In order to submit queries to a search engine so that they are inherently stored this way, users make submissions on behalf of each other. Whilst this approach seems to be the simplest solution it has an obvious disadvantage: if user U_2 is to submit query Q_1 to a search engine on behalf of U_1, then U_2 must know what Q_1 is. This may be considered a violation of privacy. However, if our goal is to prevent the search engine from violating privacy, then it could be the case that this potential for violation of privacy may be acceptable in the event that not all queries from U_1 are submitted by U_2. This would have U_2 only submitting ad hoc queries for U_1 and, just like the search engine, these queries would have no context.

Since a DHT is essentially treated as a decentralised collection of nodes used to store data, we introduce a rule that each node must obey in order to have users successfully submitting queries on behalf of each other. The rule is simple: if a predefined data structure is present in the data to be stored, the necessary elements required to make a submission to a search engine must be extracted from this data. The search engine is then queried, the result of which is encrypted and stored in place of the original data.

The data structure serves only as a means for nodes to recognise when to

store the data and when to use the contents thereof to formulate a query to a search engine. For example, *data* containing "**BING,Anonymity**" may signal a node to send the query "Anonymity" to the bing.com search engine. The result would then be encrypted (using "Anonymity" as the key) and stored. Note that the proposed search network is not bound to a specific search engine.

Of course, a large number of colluding nodes could possibly thwart the search network's attempt to preserve the privacy of users from each other. Increasing the number of attackers in the DHT increases the probability of receiving a number of queries on behalf of the user under attack and, in doing so, increase the likelihood of generating a log with context. There are a number of options for this scenario. The first option is to bypass the DHT nodes entirely and have a foreign node join the DHT to submit the request on the user's behalf. The foreign node could be part of another PET, for example, Tor [42]. Unfortunately, making use of another network will most likely introduce one of the very problems the search network proposed in this chapter tries to remedy: scalability and performance.

Another option is a variation on the technique employed by "John Doe" nodes in the Crowds model [100]. In this model, a John Doe picks a random node from the crowd (which could be itself) and forwards the message to the node. This node, upon receiving the message, flips a biased coin to determine whether to send the message to another John Doe node or to deliver the message to its recipient. Nodes in the search network could employ a simpler process: upon determining that a search result is not available for a particular query, a node can flip a coin to determine whether to issue a request into the DHT to have the query addressed (using the predefined data structure) or address the query itself and place the result into the DHT. Since the node is part of the DHT it will have already conducted a number of queries on behalf of other nodes. As a result, detecting the user's queries from a search engine's perspective will be cumbersome. Ultimately, the queries still have no context.

We have opted to adopt the latter approach when submitting queries in the proposed search network.

7.4 Formalisation

In this section, we highlight the relationship between an accurate search profile and a log of queries submitted by a user. We then analyse a number of methods that can be used to submit queries to a search engine. The focus of our analysis deals with the logs that can be derived by an entity the likes

of a proxy or search engine. We will show that constructing an accurate search profile of a user participating in a search network requires collusion with all parties in the network.

A log of search queries can be prepared from any one of the following perspectives: the user issuing the queries, a proxy user issuing the queries or the search engine for which the queries are destined. In this section we place emphasis on the number of queries in a log. Correct knowledge of a log (and its contents) implies that one knows how many queries have been issued. Similarly, if one does not know how many queries were issued then the contents of the log are not entirely known.

Within the context of this chapter, if a user submits n queries to a search engine, the log of this user from his/her perspective is entirely known and will contain n queries. If the log generated for this user from the perspective of a search engine contains the *same* n queries then the search engine has correct knowledge of the log.

L is a function with the set of users, U, as its domain and the number of queries associated with his/her log, V, as its range. With $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \mathbb{N}$, L is the function that maps a user to the number of queries issued.

$$L : U \mapsto V \tag{7.1}$$

For any user u_i, the number of search queries issued is $L(u_i) = n_i$.

Let $L_s(u_i)$ denote the number of queries issued by u_i from the perspective of search engine s. Similarly, $L_{u_j}(u_i)$ denotes the number of queries issued by u_i from the perspective of u_j where $j \neq i$. If $L_s(u_i) \neq L(u_i)$ then the contents of u_i's log are not entirely known by the search engine (it does not have an accurate search profile for u_i).

7.4.1 Submitting Queries Directly

In the absence of a proxy, all search queries issued by u_i are submitted directly to the search engine. Since the number of queries in the log generated from the perspective of the search engine equals the number of queries in the user's log, the search engine has correct knowledge of the log (also referred to as an accurate search profile), i.e., $L_s(u_i) = L(u_i) = n_i$.

7.4.2 Submitting Queries Through a Proxy

Consider the introduction of a proxy where u_j submits queries on behalf of u_i. From the perspective of a search engine, the logs generated yield the

following: $L_s(u_j) = n_i$ (there are n_i queries for user u_j) and $L_s(u_i) = 0$ (there were no queries for user u_i). Whilst the search engine can't derive anything about user u_i (there is no log of his/her queries), the user through which u_i is proxying is in a much better position to build a search profile since $L_{u_j}(u_i) = L(u_i) = n_i$. Essentially, the problem for user u_i is that the accurate search profile has now been shifted from the search engine to the user that is acting as a proxy.

7.4.3 Using a Proxy and Direct Submission

We try to alleviate the problem of shifting the profile from one entity to another by introducing the flipped coin approach discussed in the previous section. Instead of forwarding all queries to a proxy we let p denote the probability of forwarding a query to a proxy. Since there are n_i queries submitted by u_i, an average of pn_i of these will be forwarded to the proxy and $(1-p)n_i$ to the search engine. The result is that a number of queries go to u_j and a number to s (note that we are now dealing with expected values):

$$L_{u_j}(u_i) = L_s(u_j) = pn_i \tag{7.2}$$

$$L_s(u_i) = (1-p)n_i \tag{7.3}$$

By splitting the queries between the two entities, we decrease the chance of a single entity forming an accurate search profile.

However, if it is known that u_j is only a proxy for u_i, the search engine can easily construct the complete search profile by combining its log of u_i with the log of u_j. We refer to this log as $L_s\prime(u_i)$:

$$L_s\prime(u_i) = L_s(u_i) + L_s(u_j) \tag{7.4}$$

Using equation 7.2 and 7.3:

$$L_s\prime(u_i) = (1-p)n_i + pn_i = n_i = L(u_i)$$

7.4.4 Using Multiple Proxies and Direct Submission

If we increase the number of users acting as proxies for u_i to m, constructing $L_s\prime(u_i)$ becomes tedious for s since it would have to know each of the m users that u_i is proxying through. We know that the probability of forwarding a query to a proxy is p. With m proxies, the chance of forwarding a query to any particular proxy is $\frac{p}{m}$. As a result

$$L_s(u_j) = \frac{pn_i}{m} \tag{7.5}$$

Compiling the search profile for u_i from the perspective of s would be achieved by adding the logs of all m proxies to $L_s(u_i)$:

$$L_s\prime(u_i) = L_s(u_i) + \sum_{j=1}^{m} L_s(u_j) \tag{7.6}$$

Using equation 7.5, 7.6 expands so that $L(u_i)$ is determined:

$L_s\prime(u_i) = (1-p)n_i + \sum_{l=1}^{m} \frac{pn_i}{m} = (1-p)n_i + m(\frac{pn_i}{m}) = n_i = L(u_i)$

As the number of proxies for u_i increases, a successful attack from a search engine perspective becomes difficult only in the sense that it has to know which users are acting as proxies for u_i.

7.4.5 Submission of Queries Through a DHT

In this chapter, we have proposed a network where all users act as proxies for one another. This very simple act adds significant complexity to an attack from a search engine since, as we are going to show, it would have to collude with each of the users proxying for the victim in addition to knowing who is proxying for the victim. For the sake of simplicity, in this section we assume that users in the search network act as non-caching proxies.

Since all users in the proposed network submit queries on behalf of other users in the network, the log of queries submitted by any user from the perspective of the search engine will include all queries submitted by the user directly (remember from 7.3 that this is $(1-p)n_i$) in addition to queries for which the user acted as a proxy. To be precise, in a network of m users:

$$L_s(u_i) = (1-p)n_i + \sum_{j=1}^{m} \frac{pn_j}{m}; i \neq j \tag{7.7}$$

Applying this to 7.6, we have

$$L_s\prime(u_i) = (1-p)n_i + \sum_{j=1}^{m} \frac{pn_j}{m} + \sum_{j=1}^{m} L_s(u_j) \tag{7.8}$$

We have shown that the search engine must take into account that u_i is submitting queries on behalf of other users (depicted in 7.7). Note that each proxy u_i is using is also acting as a proxy to users other than u_i. Furthermore, the proxies are submitting queries themselves. Unless the search

engine colludes with each of the proxies (ensuring that they are not submitting any queries themselves), then the log of queries submitted to the search engine from each proxy is similar to that of u_i in 7.7:

$$L_s(u_j) = (1-p)n_j + \sum_{l=1}^{m} \frac{pn_l}{m}; l \neq j \tag{7.9}$$

With this in mind, we expand 7.8:

$$L_s\prime(u_i) = (1-p)n_i + \sum_{j=1}^{m} \frac{pn_j}{m} + \sum_{j=1}^{m} L_s(u_j)$$
$$= (1-p)n_i + \sum_{j=1}^{m} \frac{pn_j}{m} + \sum_{j=1}^{m}((1-p)n_j + \sum_{l=1}^{m} \frac{pn_l}{m})$$

Table 7.1 summarises the logs derived on the search engine and proxy for each of the approaches discussed in this section. Note that in the search network approach, a search engine colluding with a number of proxies will only be successful (generate an accurate search profile) when the number of proxies colluding in the network equals $m - 1$.

7.5 Conclusion

Using the database of 2006, we took a random sample and confirmed that there is a significantly large number of queries which are shared by users when using search engines. If we assume that the results of these queries are not specific to the user conducting the initial query, then the results from a single query can be shared with other users of the network, sparing them from having to conduct the query themselves. In doing so, the user would be disconnected from the search engine.

The disconnection from the search engine and the sharing of queries forms the basis upon which the search network is proposed. Essentially, the search network acts as a cache of search queries and search results. Built on top of a distributed hash table, the search network allows users or nodes to place and retrieve queries and their associated search results from the cache. A simple encryption scheme using the query to encrypt the search results allows for a degree of privacy between the users of this network. The major contribution is that of removing control from the search engines. In this network, exclusive control over one's search profile is no longer in the hands of a single entity.

DHTs have shown themselves to be fast and massively scalable. These two characteristics are of paramount importance in a network of this nature.

Method	**Log on Proxy**	**Log on Search Engine**
Search engine	0	n_i
Proxy	n_i	0
Search engine and proxy	pn_i	$(1-p)n_i$
Colluding search engine and proxy	pn_i	n_i
Search engine and m proxies	$\frac{pn_i}{m}$	$(1-p)n_i$
Colluding search engine and m proxies	$\frac{pn_i}{m}$	n_i
Search network (m users)	$\frac{pn_i}{m}$	$(1-p)n_i + \sum_{j=1}^{m} \frac{pn_j}{m}; i \neq j$
Search network with colluding search engine and proxies	$\frac{pn_i}{m}$	n_i iff number of colluding proxies $= m-1$, otherwise $(1-p)n_i + \sum_{j=1}^{m} \frac{pn_j}{m} + \sum_{j=1}^{m}((1-p)n_j + \sum_{l=1}^{m} \frac{pn_l}{m})$

Table 7.1: A comparison of the different approaches used when querying a search engine.

A greater number of users will result in a higher probability of a query already being conducted by someone else on the network. If the network cannot scale well, there would not be more incentive to use it over conventional PETs.

Ultimately, the proposed search network is a form of distributed proxy. It differs from conventional privacy preserving proxies the likes of Anonymizer.com because the queries, in addition to not always being in the clear, lack context. This is not the case with a centralised proxy since although one is protected from the search engine, the proxy itself has the potential for privacy violation.

Chapter 8

Conclusion

Si finis bonus est, totum bonum erit – If the end is good, everything will be good

In this work, we looked at the privacy problem within the context of search. Having identified the search engine as an entity that could not be trusted to provide search privacy, the aim of this work was to determine whether or not there were existing technologies that could be trusted to do so. Having found that this was not the case, we investigated an alternative solution.

We began this work by looking for a definition of privacy. After examining several approaches to privacy in existing literature, we adopted Westin's definition: "the claim of individuals, groups or institutions to determine for themselves when, how and to what extent information about them is communicated to others". In this definition, emphasis is on control over one's own information.

We then looked at search and search profiles. Given the nature of privacy violations related to search in the past, we view the search profile of an individual as information belonging to that individual. With this in mind, search privacy is essentially defined as the control that one has over one's search profile.

Participating in the search experience verbatim (using a search engine directly) leaves complete control over one's search profile in the hands of the search engine. By extending control of one's search profile to the search engine, one is effectively relinquishing control and in doing so giving up one's search privacy. As a result, search privacy in this scenario is non existent.

With an understanding of what search privacy means and the knowledge that it does not exist within the common use case scenario offered by the search engines of today, we began our search for search privacy through other means. As we examined the potential for search privacy that each PET category could offer, we asked two questions: (1) does the technology enhance control over one's search profile and (2) is the search engine still a threat to search privacy? Within the context of the definition of search privacy that we have offered, a suitable technology must not only enhance control but it must also eliminate the search engine as a threat to search privacy.

We concluded the following from each of the categories examined:

Organisational Safeguards. This category was quickly dismissed as an option for providing search privacy. The reason for this is because total control lies in the hands of the organisation, i.e., it is up to the search engine to implement internal policies that provide users with control. When control of a search profile has already been lost or willingly relinquished by a user, there can be no privacy and as a result, we did not consider this category any further.

Anonymity. From a privacy perspective, anonymity is control over one's identity. This category was initially quite promising. Although one can greatly enhance one's control over a search profile through anonymity using an anonymity network, there exists a motive for the search engine to identify participants in this network. The motive is that of thwarting click fraud. We discussed and simulated an external attack on an anonymity network that was based on two principles: (1) the user responsible sending messages from the network will not represent himself and (2) each user employs a mechanism for search where related queries can be recognised (cookies for example). The result of the attack confirmed that the search engine may be in a position to expose/identify participants in the network. By identifying members in the anonymity network, the search engine remains in control of their search profile and therefore remains a threat to their search privacy.

Personal Control. In this category, the focus is on ensuring that information of an individual is only used in a way that the individual has deemed appropriate, i.e., the individual has control over the use of his own information. We turned to P3P and asked the same two questions. It was quickly evident that like the other categories, P3P did enhance control but there were problems. The first problem was that of trust. Any entity could easily declare that they would abide to a sterling P3P privacy policy, but how does one know that they will adhere to what has been promised? We offered a solution in the form of an already proven technology in similar scenarios: a reputation system. The second problem was that of proxies. P3P neglects to consider the role played by an entity in the middle of a Web transaction. After highlighting this problem, we provide a solution in the form of subtle changes to P3P that essentially considers the privacy policy of any proxy involved in a Web transaction. Despite having addressed these two issues, there exists a more fundamental problem. The user is still communicating with the search engine and a search profile will still be constructed. Therefore, full control remains in the hands of the search engine. A search engine that adheres to its P3P policy can still be compromised and along with it, the profiles of its users.

Private Communication. Encryption and/or obfuscation are considered as a means to achieve search privacy in this category. We first looked at encryption but eventually dismissed this option for although it will provide the user with a more secure channel when communicating with the search engine, it does not protect the user from the search

engine itself (a search profile is still constructed and controlled by an entity other than the user). We then considered obfuscation and looked at a tool which uses this technique to protect its users from the search engine. By sending waves of machine generated false queries around the legitimate queries of the user, the idea is to leave the search engine with what is essentially an inaccurate search profile. Since the user has full control over the queries submitted to a search engine, this is a legitimate approach which initially looked to be promising. Although search privacy is enhanced in this scenario we found that the search engine remains a threat to privacy. With a motive to determine who is behind a query (something the search engines have a vested interest in doing so as to thwart click fraud), we discussed the scenario of search engines determining if a human is behind a query or a machine. Whilst the query itself may seem perfectly legitimate, it is the way in which the user/machine interacts with the result that can leave the search engine deciding on whether the user is indeed human. In doing so, the search engine can distinguish between false queries and legitimate queries and ultimately generate an accurate search profile.

Of the three categories which proved to enhance the control one has over one's search profile, none of them eliminated the search engine as a threat to privacy. The fundamental problem in each category is that the original search scenario is being replayed when communicating with the search engine: the user is submitting queries to the search engine (either directly or indirectly – in the case of anonymous networks). In each case, the search engine receives the query and when possible will construct a search profile for the user in question over which it will have ultimate control.

In offering a solution to this problem, we asked if it was always necessary to submit the query to the search engine. By not having the user's query to begin with, the search engine will not have anything from which to construct a profile. We turned to proxies, an already incredibly useful technology on the Web. If we could construct a distributed cache of search queries and their associated results, queries would only be sent to the search engine under specific conditions.

The primary use case of a proxy is that of sharing data. If an entity has requested and received data from a remote source through a proxy, one can save having to make another request to the remote source for the same data by making a copy of the data available to other entities sharing the proxy. We believed that this scenario was applicable within the context of search. For any given search query, why retrieve the search results from the search engine when they may already exist in the distributed cache?

We made a case for this by confirming that there is indeed a significant number of search queries that are shared by users. As a result, a search network of this kind would be very useful. In the event that a search query was not in the cache, we proposed a simple mechanism for users in the network to retrieve the result from the search engine and place it in the cache. Since the cache is distributed, no single user will have complete control over a user's queries. Similarly, in the event that the search engine can identify the original source of queries sent to it, the queries form only a subset of all queries sent by the user in question.

The contribution to search privacy comes in the form of an almost complete disconnect between the search engine and the user conducting a search. Since each of the PETs we examined in previous chapters failed to introduce this disconnect, the original search scenario was maintained because every query eventually resulted in a request to the search engine. As a result, the search engine remains in a position to construct a search profile ergo control of the search profile is lost as is the search privacy of the user.

In chapter 3, we described the typical search scenario as follows: online users issue a query to a search engine which results in a set of links which redirect to Web pages with more information. Note that the network we have proposed to address this scenario in a manner which maintains search privacy did not explicitly deal with the redirection problem. In order for there to be a disconnect between the searcher and the search engine, the redirect through the search engine that has been embedded in a search result can not be honored. If the redirect is honored then the original scenario is maintained for the search engine is still directly involved in the search process – it will be in a position to determine what the context of the redirect is and in doing so construct a search profile. In treating the links like search queries in the DHT though, the problem of redirection can be overcome, i.e., $put(k, data)$ where k is the result in the search results page and $data$ the link after redirecting through the search engine.

8.1 Does this enhance search privacy?

The search network can be said to enhance search privacy through two contributions:

> *Disconnection.* By interrupting the traditional search scenario, ultimate control over a search profile has been shifted from the search engine to the user. In a limited sense, one could argue that control is distributed amongst users of the distributed proxy for, after all, the

search results must come from somewhere and there is no disconnection between the users.

Search Democracy. The search network introduces the notion of free and equal representation of search results on the Web, one in which no single entity has ultimate control over deciding which Web sites are right, which Web sites are wrong and, more importantly, who is searching for them.

8.2 Is the search engine still a threat?

If the search engine has the motive to be considered a threat, there is sufficient reason to believe that it will be a threat. Given the nature of search today, we believe the following two scenarios would serve as enough of a motive:

1. A search network acting as a distributed cache of search queries and search results would be threatening one of the search engine's primary revenue streams: advertising. Reducing the number of queries going through a search engine has a direct impact on the number of adverts that can be displayed. Fewer advertisement impressions will ultimately result in much lower click-throughs to advertisers and inevitably less profit for the search engine.

2. Not knowing exactly what results users have chosen for the majority of queries passed through it (via redirects) may have a dramatic impact on the validity of the results offered by the bigger search engines. A dominant search engine that knows what the majority of the world is searching for (in addition to what the majority of the world deems as most relevant from the results provided) does not want to lose this advantage over other search engines.

For as long as the disconnect between the searcher and the search engine exists though, we believe that the search engine is of little threat to search privacy because it will be unable to construct accurate search profiles. However, given the incentive, the search engine has only to restore the connection to the searcher in order to gain full control of the original search scenario (and search privacy). We see two ways of achieving this:

Providing personalised search results. The search network is based upon sharing results. Since there are a significant amount of shared queries, a distributed cache of shared results seems natural and somewhat evolutionary (much like the introduction of the proxy was to the Web). If

the search engine can challenge this simple premise and provide search results in a manner that would make no sense to share them, then the network would have lost the foundation upon which it was built. By serving personalised search results the search engine would reduce the benefit gained by sharing queries and in doing so, restore the connection between the searcher and itself.

Attacking the search network. We know that control of a search profile is in some ways distributed to participants throughout the search network. We have not investigated the feasibility of gaining control over a profile by inserting a number of false participants into the network on behalf of the search engine. Since the search engine has a fairly good idea of what queries are shared and likely to be cached in the network, will it be possible for it to only target those queries in the search network? We hope that the work in this thesis has set the foundation upon which to answer these questions in future research.