Censorship-Resistant and Privacy-Preserving Distributed Web Search

Michael Herrmann*, Ren Zhang*, Kai-Chun Ning†, Claudia Diaz* and Bart Preneel*
*KU Leuven ESAT/COSIC, iMinds, Leuven, Belgium,
Email: firstname.lastname@esat.kuleuven.be
†National Chiao Tung University, Hsinchu, Taiwan
Email: kaichun.ning@gmail.com

***Abstract*—The vast majority of Internet users are relying on centralized search engine providers to conduct their web searches. However, search results can be censored and search queries can be recorded by these providers without the user's knowledge. Distributed web search engines based on peer-to-peer networks have been proposed to mitigate these threats. In this paper we analyze the three most popular real-world distributed web search engines: Faroo, Seeks and Yacy, with respect to their censorship resistance and privacy protection. We show that none of them provides an adequate level of protection against an adversary with modest resources. Recognizing these flaws, we identify security properties a censorship-resistant and privacy-preserving distributed web search engine should provide. We propose two novel defense mechanisms called node density protocol and webpage verification protocol to achieve censorship resistance and show their effectiveness and feasibility with simulations. Finally, we elaborate on how state-of-the-art defense mechanisms achieve privacy protection in distributed web search engines.**

I. Introduction

Finding the most relevant information in the World Wide Web (WWW) is a challenging task. The sheer size of the WWW makes this task for a normal user virtually impossible. Therefore, search engines are among the most important and most frequently used services of the WWW. *Centralized search engine providers*, such as *Google* and *Bing*, build server farms with massive computation and bandwidth resources in order to analyze the structure and content of the WWW, create a centralized index, and enable users to query this index.

In the past years significant concerns have emerged with respect to centralized search engine providers. On one hand, concerns about censorship have been raised because the users' search results are opaquely created. Since a centralized search engine has total control over its index, it also has the power to remove/censor parts of the index or manipulate the ranking. On the other hand, privacy concerns are raised because these providers collect all search queries. This collection of information could be used to infer sensitive information about a user, such as income level, religious beliefs or health conditions [1]. This information may be sold to or stolen by unforeseeable entities, such as advertisement companies, criminals or intelligence agencies.

As a promising alternative to centralized search engine providers, *distributed web search engines* (DWSE) have been proposed to solve censorship and privacy issues. The key idea is to distribute the massive workload of running a search engine to a peer-to-peer (P2P) network. This decentralized structure promises to overcome censorship and privacy concerns, because there is no central instance that is in control of all the index data or user requests. Instead, each peer stores a part of the index on its local machine and all peers build together the *distributed index*. Similarly to the *store* and *retrieve* functionality of P2P file-sharing networks, every peer is able to store information on what webpages are relevant for a given search term and to retrieve relevant URLs to a certain search term, respectively. Additionally, these systems are usually open source. The execution of self-compiled source code on the user's device provides transparency in how the search results are obtained and ranked.

In this work we present an analysis of real-world DWSE on their censorship resistance and privacy protection with respect to common P2P threat models. To the best of our knowledge there exist only three real-world DWSE: *Faroo*[1], *Seeks*[2] and *Yacy*[3]. After investigating the designs of these systems, we find that by deploying an eclipse attack [2] with a few malicious nodes, an attacker can easily remove a certain topic from the distributed index or get the list of IP addresses of the nodes that searched for the topic. In addition, we conduct the attacks in a real-world YaCy network that we set up in the Planet-Lab[4] network. We conclude that none of the existing DWSE actually keeps up to their promises to provide censorship resistance and privacy protection.

Acknowledging the flaws of current DWSE, we identify security properties for a censorship-resistant and privacy-preserving DWSE. This contributes to the proper evaluation of DWSE. Furthermore, we provide design recommendations for future DWSE. Firstly, by utilizing the network topology secured by the state-of-the-art P2P anonymous communication systems [3], [4], we propose the *node density protocol* to mitigate the eclipse attack used in censorship. Secondly, we provide a protocol named *webpage verification* to remove malicious peers performing eclipse or index poisoning attacks [5] from the network. The protocol exploits the fact that search results are verifiable by any node in the network and thereby shows that censorship resistance in DWSE can be achieved in an easier way than in, for example, file-sharing networks. Finally, we argue that essentially, privacy protection in DWSE is the same issue as privacy protection in P2P anonymous communication systems.

[1] http://www.faroo.com/
[2] http://seeks.fr/
[3] http://yacy.net/en/index.html
[4] http://planet-lab.eu/

The remainder of the paper is structured as follows: Section II provides the description and evaluation of real-world DWSE. We deploy our attacks in a real-world Yacy network and present the result in Section III. We identify security properties of censorship-resistant and privacy-preserving DWSE in Section IV and show how they can be achieved in Section V. We provide simulation results to evaluate our protocols in Section VI. We review related work in Section VII, give an overlook on future work in Section VIII and conclude this work in Section IX.

II. Evaluation of Distributed Web Search Engines

In this section we describe and evaluate the design of Faroo, Seeks and Yacy. Peers in all three systems organize the distributed index with reverse word index (RWI) entries, i.e. the peers store *search term* $\rightarrow$ *URL* if a particular webpage is relevant to a search term. Faroo is a closed source DSWE that claims to be powered by 2.5 million peers. Although there was little activity in the past two years in the Seeks community, a public instance of Seeks still exists. YaCy has a very active user and development community which provides regular updates. We have verified with the developers of these systems that our description is accurate. Although all three DWSE have no central entity that is able to observe all user queries or to censor the distributed index, none of them provides a security analysis with respect to common P2P threat models. We outline how all the considered DWSE are vulnerable to one or several of the following well-known P2P attacks: The eclipse attack [2], the index poisoning attack [5] and the route capture attack [3].

A. Threat Model

In this work we consider an attacker able to control a set of colluding nodes inside the P2P network with a modified client program that can launch both active and passive attacks. The network was first established by an honest user and the attacker does not attack the bootstrap machines. In terms of the definitions of Raymond [6] the attacker is internal, active and static. In line with most other works of P2P security, such as [3] and [4], we do not consider a global attacker.

B. Faroo

Faroo implements a modified Kademlia [7] distributed hash table (DHT). The ID of a node is generated from the node's MAC address and some other local information. After a webpage is crawled, the client program generates relevant RWI entries and computes the target ID for each entry, which is the hash value of the search term. An entry is then stored on 20 *index nodes* whose IDs are closest to the target ID.

To perform a search, the initiator of a search query computes the target ID of the search term and queries the relevant index nodes. The IP address of the initiator is sent along with the query so that the responsible index nodes can reply in a direct connection. Only the top 100 results are replied.

The communication is secured with two layers of encryption. The protocol layer encryption can prevent unsophisticated passive observers from understanding the packets. RWI entries are further encrypted with the search term as key. The client program is obfuscated to make reverse-engineering difficult.

Now we present a sketch of deploying an eclipse attack to censor a keyword. Note that we assume that the client program is successfully reverse-engineered, because generally, security through obscurity is not accepted in the security community. First, an attacker learns the closest honest nodes by searching the keyword. Second, the attacker uses the client program to compute the keyword's ID and generate 20 node IDs closer to the target ID than these honest nodes. Illegal node IDs will not be spotted since there is no way for another node to check a node's local information. Third, the attacker deploys 20 malicious nodes with these IDs. These 20 nodes' IPs need to be in different /24 subnets, and no more than 5 can reside in the same country. Now the new index nodes are all malicious nodes. The attacker can then censor the keyword by accepting all index requests and by replying with only a subset of all locally stored entries. The attacker also gets the IP addresses of search initiators, thus user privacy is breached. Moreover, it is difficult to judge that an index node is malicious simply by observing that a stored entry is discarded, because the entry may not be in the top 100 results.

C. Seeks

Seeks does not have a crawler. It does not aim at re-indexing the WWW but enables collaboration and re-ranking via a distributed architecture. The client program displays every search result as an aggregate from centralized search engines designated by the user and the Seeks P2P network.

The Seeks P2P network is built on Chord [8]. Seeks utilizes the network fundamentally differently from the approach of YaCy and Faroo. A search query is not mapped to the relevant RWI entries but to users who issued similar queries. Users can also "push" entries into the network so that they will show up in other users' search results. Seeks implements a locality-sensitive hash function that maps similar queries to the same ID and thus the same group of index nodes. A user is able to register herself at the index nodes for her search query and to retrieve the IP addresses of other peers that issued similar queries. This enables the users to communicate and share interesting search results.

The goal of Seeks is in direct opposite of providing anonymity: An attacker can get a list of peers interested in a search term simply by issuing the query himself. In terms of censorship resistance, while it is difficult to censor search results from centralized search engines, Seeks does not serve as an alternative for users seeking to mitigate the risk of being censored by centralized search engines. Meanwhile, an attacker can also easily deploy the eclipse attack to occupy index nodes for a search query, therefore prevents users from communicating with each other. An index poisoning attack of injecting fake URLs into the network is also possible.

D. Yacy

Although there is an in-depth description of YaCy available [9] we summarize here the its design characteristics that are important for the security evaluation in Section III. The YaCy software connects by default to the public YaCy network called *freeworld*. Besides, YaCy can also be configured to run a private distributed search engine. In a Yacy network, the address space is defined as a ring similar to that of Chord.

Every peer maintains a global view of all peers in the network, and therefore can directly contact every other peer. This is achieved with the following two mechanisms. Firstly, there are super peers called *principal peers* that collect and distribute information about every peer in the network. Secondly, peers frequently exchange information about all peers they know about, known as *peer ping protocol*.

On a peer A, information of another peer B is stored in a *seed* data structure. All the seeds together form A's *seedlist*. For the scope of this work, there are 5 important values in the seed about B. The *last-seen* value indicates the time when A last communicated with B. The *root-mode* flag is set if B's last peer ping was finished in less than one second. The *RWI count* depicts the number of RWI entries that are stored on B. The *remote-index flag* indicates whether B receives and stores entries from other peers. If this flag is cleared, B is referred to as *Robinson peer* and only contributes to the distributed index with entries that are locally stored by itself. At last, optionally, a few keywords may be listed in a Robinson peer's *search tag* field, indicating some search terms that have related entries in this peer. Please note that the last four values are reported by the peer itself, or may be overwritten by information provided by other peers.

A YaCy peer is able to choose its position in the network freely. For every network join, a peer first contacts the principal peers for their seedlists, and then starts a peer ping with the 20 youngest peers with respect to their last-seen values. Subsequently, the peer periodically engages in peer pings with the three oldest peers in the network.

The main purpose of YaCy is to store and retrieve information from the distributed web index. To this end, every peer maintains two databases: a self-implemented RWI database with a well-tried storage/retrieval mechanism and a Solr[5] database with a rather experimental storage/retrieval mechanism. Both databases store entries of the form similar to Faroo's. Whenever a peer crawls a webpage it stores the RWI entry for every word in the webpage, i.e. every search term, at the designated address. The address is a 63-bit string with the 4 most significant bits determined by the hash of the URL and the others determined by the hash of the word. The crawling peer stores an entry at the 3 closest successor peers according to the address. Furthermore, both the crawling peer and the 3 index peers store the entry in their local Solr databases.

When a peer wants to retrieve search results from the YaCy network, it creates a candidate set for sending RWI and another one for sending Solr search requests. The candidate sets depend on the number of terms in the user's search query. For this work it suffices to outline the process for search queries with one word. As a result of the storage mechanism, all entries for the same search term are found on $2^4 = 16$ positions in the network, because the storage addresses for the same search term only differ in the most significant 4 bits. These 16 positions are called the *vertical positions* for the search term. The part of address space that shares the same 4 most significant bits are called a *vertical partition*. For retrieving entries from the RWI database, which we refer to as *Yacy search*, the candidate set consists of the 2 closest successor peers to every vertical position. Additionally, each of the 5 most heavily loaded peers, i.e. the peers with the highest RWI count in the network, is added to the candidate set with probability 80%. To construct the Solr candidate set for the *Solr search*, the following peers may be added: 20 random peers that have the root-mode flag set; all Robinson peers that have a search tag that matches the user's search query; and finally up to 25% of all Robinson peers in the network. After the two candidate sets are created, the search initiator contacts every peer in the sets and transmits the hash of the search term along with its requests. The receivers then inspect their local databases and reply with relevant search results, if available.

The fact that each peer can choose its own ID and report for themselves values in the seed enables an attacker to easily deploy two attacks to fill the candidate sets: An eclipse attack to occupy successor positions of target IDs, and a route capture attack to bias the seedlists of honest peers and redirect search requests to its malicious peers. The attacker can therefore censor the related entries or learn the IP addresses of users searching for a certain keyword or URL to break the users' privacy.

III. Attacks on Yacy

In this section we conduct attacks on Yacy, one of the real-world DWSE. Yacy was selected because Faroo is a closed source project, and Seeks does not have as solid a user base and mature source code as Yacy does. We show that YaCy provides little censorship resistance and no privacy protection in the presence of malicious peers in the network.

A. Setup the Experiment Environment

We set up our private YaCy network with Planet-Lab nodes. Note that our network has the same security properties as freeworld, since honest peers are still using the unmodified Yacy client software. Furthermore, as we will see below, the attacks we deployed are scalable to the size of freeworld. The test network consists of 149 Planet-Lab nodes and one machine from our lab in the role of the principal and bootstrap peer. Among the 149 peers, we added 3 peers after every vertical position of the RWI entry of *Jediism*. Then we added 61 peers in an ordinary, standard-user-like configuration. Finally, we added 40 peers with various configurations that allow the investigation of every possible YaCy peer type. In particular we added 5 peers with highest RWI count, 25 peers with root-mode flag set, 9 peers with cleared remote-index flag (Robinson peers), and 1 peer with cleared remote-index flag and a search tag '*Jediism*'.

We did not launch any attacks in freeworld in order to protect the privacy of its users. Only for parts of the investigations in Section III-D we conducted a privacy non-invasive experiment in the freeworld network in order to learn its churn rate. Furthermore, we obtained the permission of the YaCy community in advance.

B. Verification of the Candidate Sets

It is crucial for an attacker who wants to monitor and censor search requests to understand how the candidate sets are constructed. Therefore we conduct an experiment to examine whether our understanding of the Yacy source code is correct.

[5]http://lucene.apache.org/solr/

TABLE I: Percentage of received search requests in our experimental network.

YaCy search requests	
Two closest successors to vertical position:	100%
Third closest successor to vertical position:	0%
Peers with highest RWI count:	Mean = 75.5% StdDev = 4.79%
All others:	0%

Solr search requests:	
Peers in root-mode:	Mean = 79.23% StdDev = 5.22%
Robinson peer with tag *Jediism*:	100%
9 Robinson peers:	Mean = 26.07% StdDev = 3.32%
All others:	0%

We made 3 peers search the term *Jediism* for 50 times with different time intervals (200s, 250s, 300s). The results of our experiment are shown in Table I. It can be seen that they match the description in Section II-D.

C. Monitoring and Censoring Keywords in Yacy search

For the sake of brevity we only elaborate on search queries that contain one search term. Note that the attacker's cost increases linearly with the number of search terms.

Recall from our experiment in Section III-B that controlling two closest successors at each of the 16 vertical positions and 5 most heavily loaded peers is adequate for an attacker to receive all search requests and is thus able to censor the search request. Acquiring a desired ID is possible because a peer is free to choose its ID. Furthermore, since the RWI count is reported by the peer itself, a malicious peer is able to report the maximum value of $2^{63} - 1$. Please note that a malicious peer that is put as closest successor to a vertical position can also report to have the highest RWI count. Therefore only 32 peers are necessary for censoring a search term. Monitoring is successful if the attacker controls one of these peers. Note that the number of malicious peers is the same regardless of the size of the network.

The attacker still needs to determine the search term related to each search request. This is because the peers only receive the hash of the search term, and most likely only a subset of the requests they received are related to the ones to censor. Therefore, a malicious peer extracts and visits all URLs in its local databases and computes the hash of every word of the respective webpage. Although this may include the computation of thousands of hashes, this attack is still feasible. Firstly the computation of a hash is cheap and secondly the result of a hash can be reused when stored in a lookup table.

To monitor a search term, the attacker logs every search initiator's IP address. For censoring a search term the malicious peers send either an empty or filtered reply to a search request. In our experiment, the attacker has an 100% success rate.

D. Monitoring and Censoring URLs in Yacy search

In this attack, the attacker's goal is to learn the users searching for information that can be found on a particular webpage or to censor content of the webpage. An approach similar to the one in the previous attack does not work, because there are potentially thousands of words on a webpage stored as different RWI entries in the distributed index, and having 32 malicious peers for each keyword is not feasible. As we will show in the following, monitoring and censoring URLs is possible in YaCy with a variant of the eclipse attack.

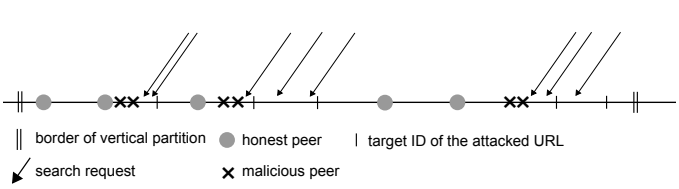

Fig. 1: Illustration of an attacker blocking all the incoming requests to honest peers in one vertical partition.

The attack exploits the fact that YaCy stores all RWI entries that belong to the same URL in the same region of the address space. In particular, the first 4 bits of an RWI entry's ID is given by the hash of the URL. Consequently, all RWI entries regarding a webpage share the same 4 bits and are thus stored in the same vertical partition. Therefore, an attacker can place malicious peers as closest successors to every target ID of the attacked URL, i.e. for every ID that results from the 4 bits of the URL and the 59 bits of every word on the URL. Please note that the adversary may be able to leave some target IDs out if there are no honest peers between the regions of two consecutive target IDs. We illustrate this blocking in Figure 1.

A complete overview of honest peers in the partition can lower the attacker's cost. Updating seedlist from principal peers is not enough since newly joined peers may appear with delay on the seedlist. To mitigate this delay, the attacker frequently pings an amount of youngest peers from the principal peers' seedlists and learns the peers they know. The attacker thereby exploits the behavior of newly joined peers that they first send pings to the 20 youngest peers in the network.

In Section III-C we have shown that placing two peers in front of another peer successfully blocks all requests to this third peer, and we therefore do not present further data on the blocking in a vertical partition. Here, we only present experimental data that shows the effectiveness for the attacker to ping other peers to obtain a more accurate snapshot of the network. In particular, we queried the principal peers of the freeworld in the frequency they publish a new seedlist (30 seconds). Subsequently, we ping the 50 peers that have the most recent last-seen tags in order to obtain their seedlists as well. With this strategy we were able to learn from 1% to 6% more peers than with the public seedlists alone.

Now the attacker can place two malicious peers at two consecutive positions after some IDs of the attacked URL according to the distribution of honest peers in the desired vertical partition. To continuously monitor/censor all search requests, the attacker may need to adjust the placement of its peers according to the honest peers in the partition.

E. Monitoring and Censoring Solr Search

For monitoring the Solr search the attacker needs to ensure that at least one of its malicious peers is added to the Solr search candidate set. The easiest option is to add one Robinson

peer with a matching search tag, because these are always added to the candidate set. However, to completely censor a Solr search request, the attacker needs to control all peers in the candidate set. While this is a hard task, the attacker can still significantly increase its success chances and occupy a large proportion of peers in the candidate set with a variant of the route capture attack as follows.

Since a peer A sets its root-mode flag itself based on the communication with other peers, an attacker can guarantee that only malicious peers in the network have the root-mode flag set: When receiving a peer ping from A, a malicious peer B delays the request by more than 1 second so that A clears its own root-mode flag; afterwards, B transfers a modified version of its seedlist to A with root-mode flags of malicious peers enabled and other peers disabled. Also in this seedlist, the last-seen values are modified such that the malicious peers have the oldest values. Thus we can make sure that honest peers in the network eventually only send peer pings to malicious peers, because they appear to have the oldest last-seen values.

The time required by this attack depends on the ratio of malicious peers in the network. We conducted an experiment in our test network of 150 peers and a ratio of 6.7%, 10% and 20% malicious peers. The results are 42, 36 and 28 minutes, respectively. Although the attack can still be performed with fewer peers, at least 3 malicious peers are necessary. This is because an honest peer sends peer ping messages to 3 peers simultaneously every round.

To control the random subset of Robinson peers is most difficult. The attacker cannot deceive an honest peer B with a modified seedlist by saying a peer C is not a Robinson peer. This is because B may try to store RWI entries on C, and once the request is rejected, B learns that C is a Robinson peer. We note however that an attacker can increase its chances to censor a search term by spreading this false information. The attacker could also deploy a DoS attack to remove Robinson peers from the network. Finally we note that the adversary's censor attack could still be successful although an honest Robinson peer is added to the candidate set. This is because there is no guarantee that this Robinson peer actually stores information the adversary aims to censor.

IV. Security Properties of Distributed Web Search Engines

As we have seen in the previous section, none of the real-world DWSE provides censorship resistance or privacy with respect to our threat model in Section II-A. As the first step to design censorship-resistant and privacy-preserving DWSE, we enumerate here what properties such a DWSE needs to provide. To the best of our knowledge, no other work has listed such properties.

For censorship resistance we take possible real-world threats as a starting point. From our perspective, common censorship practices can be classified into three categories: (1) the adversary may wish to remove a subset of search results of a search query so that the remaining results are in his favour; (2) the adversary may have the goal to remove an entire topic from the distributed index; (3) the adversary may want to simply censor any search result but does not care which ones exactly, in order to decrease the functionality of the service. The third situation happens when the adversary is not able to achieve the first two goals, and wishes to affect the functionality so that users may turn to censored search engines with better service.

With acknowledgement that any P2P protecting mechanism can be overcome by an adversary with sufficient resources, we conclude that a censorship-resistant DWSE should provide the following properties when facing an adversary with modest resources:

- *Result completeness*: Given a search query it should not be possible to remove part of the search results.
- *Topic visibility*: It should not be possible to remove a topic from the distributed index.
- *Service functionality*: It should not be possible to remove arbitrary search results to decrease the functionality of the service.

Similarly, two kinds of privacy breaches may happen in DWSE: (1) the adversary may want to learn the identity of users interested in a certain topic. (2) the adversary may link different search queries to the same user and thus build a profile about the user. Search profiles could be used to deanonymize users. Therefore, in line with existing literature [4], [10], we argue a privacy-preserving DWSE should provide the following properties:

- *Searcher anonymity*: Given a search query it should not be possible to determine the initiator.
- *Query unlinkability*: Given two or more search queries, it should not be possible to determine whether they come from the same initiator.

Finally we note that a DWSE should be open source. Although an open source program allows an attacker to modify the client software more easily and to attack protecting mechanisms in unforeseeable ways, a closed source approach has several fundamental flaws. Firstly, it would impinge the ability for users to verify that results are not being censored, that no data about the user is being collected and how the search results are ranked. Secondly, it is common practice to make implementations of programs available in order to allow for proposing and studying of novel attacks.

V. Recommendations

In this section we propose two novel protocols that we combine with state-of-the-art P2P anonymous communication designs in order to make DWSE resilient against the attacks in Section III. Consequently, this allows us to achieve the aforementioned censorship-resistant and privacy-preserving properties with respect to all currently known attacks.

In Section V-A we propose countermeasures to mitigate the eclipse and index poisoning attacks. The *node density protocol* significantly increases the cost of an adversary to put its malicious nodes in the necessary position by leveraging existing work on secure DHT routing [3], [4], and node ID generation [11]. This allows us to preserve result completeness and topic visibility for most part of the network. In addition, the *webpage verification protocol* detects nodes conducting censorship, has the malicious nodes removed from the routing

tables of all honest nodes, and thus forces the adversary to periodically exchange its malicious nodes in the network. We achieve this by exploiting the fact that verifying search results in DWSE is feasible. Combined, an adversary is only able to attack a small proportion of the network and has to continuously spend resources for maintaining the attack. Since every successful attack can be identified shortly, service functionality is also achieved, and the first two properties are further strengthened. Subsequently, we address that an adversary is also able to misdirect search queries by a route capture attack. We find state-of-the-art protocols to achieve secure DHT lookup to be sufficient to mitigate this threat. We discuss this in more detail in Section V-B. For achieving privacy, we discuss how state-of-the-art protocols on P2P anonymous communication can be used for achieving searcher anonymity and query unlinkability in DWSE in Section V-C.

Our recommendations are designed to work on DHTs, because they have better scalability than unstructured networks, and avoid private information to go through a central server like in many hybrid designs, such as Skype. Specifically, we use Chord as the underlying DHT in accordance with the literature. Meanwhile, our architecture still allows the option of having a trusted certification authority (CA) that is known by all peers. After all, users need to download the source code or executable files and probably a set of bootstrap nodes from a server. Therefore, we argue that using a CA is feasible as long as (1) no private information goes through the service; (2) the economic cost of maintaining such a service is low even when the network has millions of peers and a few attackers with moderate resources. These choices are in line with most P2P anonymous communication systems, such as Octopus [4] and Torsk [12].

Our attacks also exploit that YaCy uses self-reported values, such as a node's RWI count. This allows the attacker to easily get selected by other peers, i.e. launching a route capture attack. A possible solution would be, such as in I2P [13], to only rely on values that can be verified. This means that a node should only assume another node to possess so many RWI entries as it has received from this node in previous queries. We acknowledge however that this has an impact on the functionality of the network.

Two additional definitions are necessary to illustrate our defense mechanisms. In a P2P anonymous communication system, to locate the responsible nodes for a key, a node may query many nodes iteratively for their entire routing tables. We refer to these queries as *lookup queries*. We use the term *operation queries* to refer to the queries for searching and storing after the responsible nodes are located.

A. Censorship Resistance Against Eclipse and Index Poisoning Attacks

1) Node Density Protocol: The protocol consists of two parts. The first part relies on existing work of solving cryptographic puzzles to increase the cost for putting malicious nodes at a certain place. The second part makes a censorship attack detectable by analyzing the network topology.

The countermeasure in [11] can largely raise the required effort of an attacker to generate his desired IDs. The defense requires a user to solve a moderately hard computational puzzle, in order to compute one valid ID from the solution to the puzzle. A latest value of Dow Jones Index and the relevant date is incorporated in the ID so that it cannot be pre-computed. Solving a puzzle takes several seconds of CPU time on a common laptop. The ID is on a random position and has a validity period of a week. Therefore, in order to get enough IDs at the lookup target, the attacker has to generate a lot of IDs every week. Note that puzzle-based ID generation methods are more effective as the scale of the network grows.

For the second part of the node density protocol we note that in order to censor a certain target, the adversary needs to put several nodes between the target ID and the first honest successor. As a consequence the attacker increases the node density of this particular region which is then likely to be above the average density of the network. In an ordinary P2P network, this is not much of an advantage, because malicious nodes could always pretend to be the only peers in the neighbourhood. However, in P2P networks that have secure routing, such as [3] and [4], the malicious nodes cannot lie about the node population in their area.

We use the term *span* to represent the distance between the first and last responsible nodes of the same lookup target. We use s to denote the average span of lookup targets. If there is a CA, a signed version of s can be distributed by CA periodically. Otherwise every node can keep its own estimation of s as the average span of past queries. Once a node discovers that the span of the current operation query is smaller than $p_n s$ where p_n is a parameter less than 1, instead of searching/storing on these nodes, it sends the operation query to all nodes in the range of s to ensure that statistically, some honest nodes would be reached.

2) Webpage Verification Protocol: The typical practice to defend against index manipulation is to have replicas on multiple index nodes and hope that at least one of them is honest. However, such an approach can easily be broken by an eclipse attack in ordinary P2P networks and is not impossible even with our node density protocol. Besides, an adversary could also deploy index poisoning attacks to add fake entries to index nodes, either malicious or honest, to make censored entries come very low in the ranking.

Fortunately, there is a fundamental difference between DWSE and a typical P2P application, such as file sharing. In a file-sharing system, it is not possible to convict a node of altering search results. If a node does not reply with a certain entry, there is no way to distinguish between whether the node deliberately censors it or the entry was dropped due to expiration or unavailable copy of the file. Furthermore, poisoning attacks are very hard to mitigate, because it is too expensive for a node to verify whether the filename is consistent with the content. However in DWSE, it is inexpensive for every node to verify whether an URL is still available and related to a search term. Therefore, key to our defense mechanism is to utilize the public verifiability of web search entries.

The webpage verification protocol consists of the following 7 steps: (1) A webpage is crawled and published anonymously to the relevant index nodes, if it is a new entry, an index node should first visit the URL and do a sanity check on the entry. Sending queries anonymously is crucial in our protocol and we address this point in more detail in Section V-C. (2) If

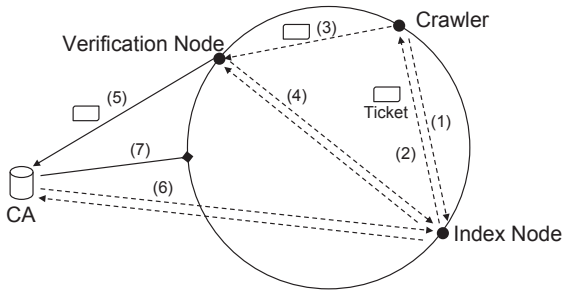

Fig. 2: Illustration of webpage verification protocol when a CA is present. A line of dashes mean the message is sent anonymously. (1) Publish; (2) Reply with signed ticket; (3) Send the ticket to verification nodes; (4) Query for search results; (5) Report malicious node to CA; (6) Verify; (7) Certificate revocation.

the sanity check passes, the index node replies a ticket with information of the entry and a signature acknowledging that the node stores the entry. (3) The node who receives the ticket then distributes it to v random nodes in the network, which we call *verification nodes*. (4) Every verification node maintains a list of all the tickets it has received. After a random interval with the average of a minute, with probability p_v, the node executes a random verification job to search for the keyword of the entry on the index node via an anonymous path. When executing a verification job, the verification node checks two things: Firstly, whether the protected entry is censored; and secondly, whether the returned URLs are valid for the search term, i.e. to detect poisoning attacks. If the answer to either question is yes, the index node is considered malicious. Note that there may be currently unavailable webpages among the search results. This does not necessarily mean that the search results are polluted and therefore a small proportion of unavailable webpages among the search results should be tolerated.

If a node was identified to be malicious by the verification node, further actions depend on whether there is a CA in the network. If there is a CA, the protocol proceeds as follows with step 5, 6, and 7: (5) the verification node sends the ticket to the CA, and (6) the CA anonymously verifies the search results of the accused node. The CA should wait a random interval before conducting the verification in order to avoid identification by a possibly malicious node. (7) We design our certificate revocation mechanism like in [4].

If there is no CA, the verification node can distribute the ticket and a warning message to every node that has the malicious node in their routing tables. Upon receiving a warning, a node verifies that the accused node indeed censors search results and, if that is the case, removes it from its routing table. Please note that contacting only the nodes which have the malicious node in their routing tables is possible in Chord, because a node's routing table is deterministic. Note that as long as the entries on the malicious node also exist on at least one honest node, the distributed index does not lose any information.

Additionally to the above described protocol it is necessary that every node in the network verifies with a probability p_s with $p_s > p_v$ the entries it stores. Otherwise an attacker is able to launch an attack that we refer to as *webserver attack*. In this attack an adversary that possesses a web server is able to have arbitrary honest nodes being falsely identified as malicious nodes and consequently have them removed from the network. To achieve this, an attacker would create text on webpages such that these webpages, after being crawled, are stored at the honest peers the adversary wants to have removed. Subsequently, the attacker would make the webpages unavailable with the result that the respective verification requests fail. Consequently, it appears that honest nodes censor the distributed index. Note that this would also render the node density protocol ineffective since the adversary is able to simply remove honest nodes in the area of its malicious peers. Similarly to step 4, nodes need to check on their own entries via an anonymous path. Otherwise the attacker would be able to have the webpages available for only those index nodes.

The security of our scheme depends on the fact that, similar to [4], a malicious index node would have to provide the correct answer to every search query in order to not risk being reported. This is because the queries from a verification node or CA is indistinguishable from ordinary queries.

We note that if a malicious node leaves and rejoins the network with a similar ID and a new public key every time it has issued a ticket, our protocol is bypassed: The verification node cannot use the ticket against it because the public key is different, but it is still responsible for the search term because the ID is similar. Therefore, a verification node re-stores an entry in the network if the node it is supposed to check upon leaves the network. As a result, a newly joined malicious peer would be again responsible for the entry.

B. Censorship Resistance Against Route Capture Attacks

We recommend ShadowWalker [3] and Octopus [4] to guarantee the attacker cannot misdirect search queries to malicious nodes by providing manipulated routing tables to lookup queries.

In Octopus, each node randomly and anonymously checks the routing tables of its fingers. If a node finds out that itself is not included in one of its predecessors' successor list, or one of its finger's fingers is not optimal, the malicious node would be reported to a CA. Since a surveillance query and a lookup query are indistinguishable, malicious nodes have to reply correctly. Their experiment has proved the feasibility of such a CA.

When a CA is not desired, ShadowWalker provides a solution worth considering to secure lookup results. A routing table entry is signed by the node A's closest predecessors and successors, called *shadows*. Any response to a lookup query is accompanied by signatures of all shadows, and the entry can only be trusted if they are consistent. As long as one honest node is presented in the neighborhood, a fake result cannot pass. We note that the optimal number of shadows is an open problem: A large number facilitates selective DoS attacks, whereas a small number makes the scheme vulnerable to eclipse attacks [14].

C. Privacy

We argue that the property of sender anonymity in P2P anonymous communication systems suffices in order to achieve the same property in DWSE. Sender anonymity can be achieved by sending queries via *anonymous paths* [15]. To construct such a path, typically two or three peers are chosen as relays to forward a query using onion encryption [15]. We propose to use ShadowWalker or Octopus to find the nodes for such paths. ShadowWalker uses a random walk over a secure topology to find entrance and exit nodes for such a path. However, ShadowWalker does not provide a strong query unlinkability, since queries from the same exit node are likely to come from a small set of peers [4]. To tackle the problem, Octopus distributes a user's requests among many anonymous paths and therefore makes it difficult for an adversary to link requests of the same node. Notably, Octopus achieves this at the cost of a higher communication overhead. Finally, we note that results from systems like Tor [15], such as padding and encryption, can be utilized to make traffic indistinguishable.

VI. Simulation Results

In the following we present simulation data showing the effectiveness and feasibility of the node density and webpage verification protocol. We fix one ID to be the target for a particular search topic. Further we define that to successfully censor a topic, the attacker needs to control the $r = 8$ closest successor nodes. Indeed 8 is a rather low number compared with existing designs like Faroo. However, a low number is actually in favor of the attacker, because the attacker only needs to add few nodes to a network, thus increasing its chance to not being detected by the node density protocol. For the webpage verification protocol it does not matter how large r is, because we only show the increased difficulty of maintaining censorship.

A. Node Density

The node density protocol uses an ID generation scheme as building block. We propose to use [11] and refer to this work for its evaluation. In the following we present simulation results on the second part of the protocol.

A lookup target is considered to be censored if the span of target is less than $p_n s$, where $p_n < 1$. We randomly distribute $1,000,000$ honest nodes and $20,000$ target IDs under attack. We conducted this experiment with different network sizes and the results are quantitatively the same. For every eclipsed target we add 8 malicious nodes between the target ID and the first honest node. To avoid detection, one malicious node is located on the preceding ID of the first honest index node. The false negative rate, i.e. the attacker's success rate, of our test is the fraction of eclipsed targets the protocol does not detect. Furthermore, we randomly distribute another $20,000$ lookup targets to measure how often an ordinary target would be misjudged as being eclipsed, i.e. false positive rate. The result of two rates for a given p_n is shown in Table II. We use the same s value for all honest nodes, computed as the average span of eight continuous nodes, be it honest or malicious. A network-wide fixed s is possible with the presence of a CA.

As can be seen from Table II, for $p_n = 0.5$ a balance between false negative and false positive rate is reached and about 97% of attacks are identified. At this point, if a user issues an operation query on every node in the range of s, all identified attacks are mitigated. The cost of the node density protocol is no more than sending a few more operation queries.

TABLE II: Performance of the node density protocol.

p_n	0.4	0.45	0.5	0.55	0.6
false positive	0.82%	1.73%	3.06%	5.0%	7.48%
false negative	5.83%	4.08%	3.0%	2.0%	1.36%

TABLE III: Performance of the node density protocol when a sybil attack decreases the network's average span.

f	20%	30%	40%	50%
false positive	2.06%	0.91%	0.37%	0.11%
false negative	3.68%	5.5%	8.37%	12.78%

To cope with the defense, the attacker may launch a sybil attack [16] to add a lot of malicious nodes to decrease s. Thus our second experiment evaluates the relation between the fraction of malicious nodes f and the success rate of the attacker. As it turned out to be efficient, we fix $p_n = 0.5$ in this experiment. The number of honest nodes remains the same, therefore when $f = 50\%$, there are altogether $1,000,000$ malicious nodes. Again we chose $20,000$ censored targets and $20,000$ targets not being censored. As can be seen from Table III, even when 40% of the nodes are malicious, the attack can still be identified in 91.6% of the cases.

Our last experiment computes the life expectancy of a successful attack. We give every honest node a lifetime that satisfies exponential distribution with mean value t, as in [4]. A new node joins the network at a random position when an old node dies. An attack is ended if a new node falls within the area of $p_n s$ after the eclipsed target. We fix $p_n = 0.5$. We run the simulation long enough to collect lifetime of 1240 successful attacks, of which the longest is $2.2t$, the average is $0.35t$.

B. Webpage Verification

In the following we show that our webpage verification protocol can identify and mitigate a successful censorship attack very quickly, and thus service functionality is also protected. For this simulation we fix 20 search topics in the network and assign every topic a popularity sampling from a *Zipf distribution* with exponent 0.5. For the network size, we simulated networks of $1,000$, $10,000$ and $100,000$ randomly distributed honest nodes. Every node generates $1,000$ entries and assigns them to the topics according to their popularity. Two tickets regarding every entry are randomly distributed. To censor a topic the adversary adds 8 malicious successor nodes to the target ID. Please note that without any of our proposed countermeasures, the adversary would be able to successfully censor a topic forever. Our goal is to measure how many search requests an attacker can censor before being caught by a verification node.

We simulate the above described network for the time period of a month. Every minute, an honest node performs two actions: (1) It issues a search query with a probability of

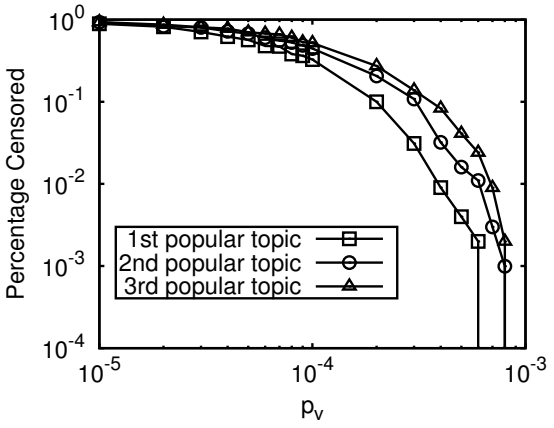

Fig. 3: The average number of censored search queries before detected with different p_v.

$p_q = 7\%$ following its predefined search topic interests; (2) It requests the search topic of a random verification job chosen from the tickets it received with probability p_v.

We measured the efficiency of the webpage verification protocol for different p_v in a network. We present the results for a network of size $100,000$ in Figure 3. Note that the results are quantitatively the same regardless of the network size. Further, we only show the results of the three most popular topics. The results are similar for topics with different popularity, since more popular topics have more entries, and thus more verification requests. With a rather low $p_v = 10^{-6}$, the attacker is able to censor most of the search requests. However, with $p_v = 10^{-4}$ the attacker's efficiency starts to significantly decrease and with $p_v = 10^{-3}$ the attacker is virtually not able to censor search results. We acknowledge that in reality, the attacker aims for result completeness may only choose to censor part of the entries in a topic, thus the optimal p_v should be larger.

The overhead of the verification processes results from the anonymous lookup of Octopus and the additional search replies to the verification node. With $p_v = 10^{-3}$ every node in the network performs a verification request on average every $1,000$ minutes. As stated in [4], Octopus consumes 4.3 Kbps if a secure lookup is performed every 10 minutes. So the overhead of the webpage verification protocol is negligible.

Although we present here no recommendation on the number of verification nodes v due to space constraints, we recommend that v should be higher than one. This is because if only one node performs a verification job, there is a rather high probability that the webpage ends up to be no longer verified upon. This happens when the verification node crashes or the node is by chance a malicious one.

VII. Related Work

Squirrel [17] is the first work that proposes to use P2P networks to build web caches. In Squirrel, every node maintains a cache of webpages that are shared among each other. The authors show that Squirrel performs similar to a centralized web cache.

The work in [18] shows that P2P networks are not feasible to create search engines that maintain an index of the entire WWW. This is because the storage, bandwidth and computation requirements cannot be met with P2P networks. However, we argue that DWSE is suitable to maintain an index of parts of the WWW that is then censorship resistant and privacy preserving.

Recently, Felber et al. [19] proposes CoFeed, a system that enriches the search results of a centralized search engine with recommendations of other CoFeed users. It therefore allows users to publish interesting search results in a P2P network. While this makes CoFeed similar to Seeks, CoFeed proposes mechanisms to protect the privacy of its users. However, please note that there is no implementation of CoFeed and that censorship resistance is not addressed by CoFeed.

Querying search engines with web anonymizers, such as Tor [15], or with *proxy search engines*, such as *Startpage*[6] and *DuckDuckGo*[7] is another possible solution to achieve privacy in web search. In particular, a search engine provider is not able to link a search query to the searcher's IP address. Furthermore, Tor is a suitable solution for escaping local censorship. However, the user can still not enjoy censorship-resistant and transparent search results, because the search results still origin from the same centralized search engines.

There are several technologies that try to prevent search engine providers to create an accurate user profile. Obfuscation-based private web search (OB-PWS), such as TrackMeNot [20], refers to techniques that automatically send dummy requests along with a user's real search query. Private Information Retrieval (PIR) [21], [22] allows users to download records from a database such that the database owner has no means in determining what particular data the user was interested in. However, neither OB-PWS nor PIR are suitable protecting mechanisms. As shown in [10], current evaluations of OB-PWS techniques overestimate the protection for users since they do not address a strategic adversary that is aware of the user's obfuscation mechanism. PIR mechanisms are unlikely to be deployed by search engine providers as they have no incentive to implement computationally costly mechanisms that hinder their business model. Finally, we note that neither OB-PWS nor PIR mechanisms are suitable to protect a user from using a censored web index.

VIII. Future Work

Future work includes an implementation of the node density and webpage verification protocols in a DWSE. This allows the in-depth analysis of the parameters in a real-world setting. In particular we would like to explore more sophisticated attack strategies in which the adversary strategically provides uncensored results in order to avoid detection. Furthermore, we will investigate the connection between the ratio of p_v and p_s and the nodes an adversary is able to remove from the network with the webserver attack.

Investigating the impact of search results ranking on censorship is future work. If the ranking mechanism is flawed, an adversary could try to flood the distributed index with

[6]https://startpage.com/
[7]https://duckduckgo.com/

irrelevant information and thereby effectively censoring some information as they appear very late in the search results.

Finally, in future work we would like to explore possibilities to formally verify whether a DWSE fulfills censorship resistance and privacy protection. This would allow for guarantees of the respective security properties while our current countermeasures only provide security in a heuristic manner.

IX. Conclusion

In the paper we introduce the designs of three real-world DWSE: Faroo, Seeks and Yacy. To the best of our knowledge these are the only real-world DWSE that are actually in use. We choose YaCy as example to show that an adversary with modest resources is able to censor the distributed index and to find out the users that are interested in a particular topic.

To address the flaws we first identify security properties every censorship-resistant and privacy-preserving DWSE should fulfill. We then propose two protocols which, combined with state-of-the-art P2P anonymous communication designs, achieve censorship resistance in DWSE. Firstly, the node density protocol allows the detection of malicious nodes by investigating the node distribution in the network topology. Secondly, the webpage verification protocol leverages the verifiability of search requests and thereby allows the detection of malicious nodes conducting censorship. Our simulation results show that both protocols are efficient and impose only little overhead to the network. In particular, the node density protocol detects about 97% of all attacks at a very low false positive rate. For the webpage verification protocol, our simulation shows that very few verification jobs in the network suffice in order to detect ongoing censorship. In addition, we argue that protecting a user's privacy in DWSE is similar to protecting it in P2P anonymous communication. Finally, we note that our countermeasures do not make a censorship attack impossible, but only increases the cost such that attackers with modest resources cannot do any significant harm to the network. However, defending against resourceful attackers is a known hard problem in P2P systems.

Acknowledgment: The authors thank Michael Christen, Wolf Garbe and Emmanuel Benazera for their kind support on YaCy, Faroo and Seeks. We also thank Christian Grothoff for feedback on an earlier version of this paper. Finally, we thank the anonymous reviewers for their helpful comments and Christoph Lenzen for his support on creating the final version of this paper. This research was supported in part by the projects: FWO G.0360.11N, FWO G.0686.11N, GOA TENSE (GOA/11/007), iMinds SoLoMIDEM, IWT SBO SPION, and KU Leuven OT project Traffic Analysis Resistant Privacy Enhancing Technologies.

References

[1] Bill Tancer. *Click: What Millions of People are Doing Online and why it Matters*. Hyperion, 2008.

[2] Atul Singh, T.W. Ngan, P. Druschel, and D.S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proc. IEEE INFOCOM'06*, 2006.

[3] Prateek Mittal and Nikita Borisov. ShadowWalker: Peer-to-Peer Anonymous Communication Using Redundant Structured Topologies. In *Proc. 16th ACM Conference on Computer and Communications Security (CCS'09)*, pages 161–172, 2009.

[4] Qiyan Wang and Nikita Borisov. Octopus: A Secure and Anonymous DHT Lookup. In *Proc. IEEE 32nd International Conference on Distributed Computing Systems (ICDCS'12)*, pages 325–334, 2012.

[5] Jian Liang, Naoum Naoumov, and Keith W Ross. The Index Poisoning Attack in P2P File Sharing Systems. In *Proc. IEEE INFOCOM'06*, 2006.

[6] Jean-François Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 10–29. Springer Berlin Heidelberg, 2001.

[7] Petar Maymounkov and David Mazires. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer Berlin Heidelberg, 2002.

[8] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *SIGCOMM'01*, pages 149–160, New York, NY, USA, 2001. ACM.

[9] Michael Herrmann, Kai-Ching Ning, Claudia Diaz, and Bart Preneel. Description of the YaCy Distributed Web Search Engine. Technical report, KU Leuven ESAT/COSIC, iMinds, 2014.

[10] E. Balsa, C. Troncoso, and C. Diaz. OB-PWS: Obfuscation-Based Private Web Search. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 491–505, 2012.

[11] Ren Zhang, Jianyu Zhang, Yu Chen, Nanhao Qin, Bingshuang Liu, and Yuan Zhang. Making Eclipse Attacks Computationally Infeasible in Large-Scale DHTs. In *Performance Computing and Communications Conference (IPCCC), IEEE 30th International*, pages 1–8. IEEE, 2011.

[12] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable Onion Routing with Torsk. In *Proceedings of the 16th ACM conference on Computer and Communications Security*, pages 590–599. ACM, 2009.

[13] Michael Herrmann and Christian Grothoff. Privacy-Implications of Performance-Based Peer Selection by Onion-Routers: A Real-World Case Study Using I2P. In *Privacy Enhancing Technologies*, volume 6794 of *Lecture Notes in Computer Science*, pages 155–174. Springer Berlin Heidelberg, 2011.

[14] Max Schuchard, Alexander W. Dean, Victor Heorhiadi, Nicholas Hopper, and Yongdae Kim. Balancing the Shadows. In *Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '10, pages 1–10, New York, NY, USA, 2010. ACM.

[15] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*. USENIX Association Berkeley, CA, USA, USENIX Association Berkeley, CA, USA, August 2004.

[16] John R Douceur. The Sybil Attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[17] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: A Decentralized Peer-to-Peer Web Cache. In *Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing*, PODC '02, pages 213–222, New York, NY, USA, 2002. ACM.

[18] Jinyang Li, Boon Thau Loo, Joseph M Hellerstein, M Frans Kaashoek, David R Karger, and Robert Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 207–215. Springer Berlin Heidelberg, 2003.

[19] P. Felber, P. Kropf, L. Leonini, T. Luu, M. Rajman, E. Rivière, V. Schiavoni, and J. Valerio. Cofeed: Privacy-Preserving Web Search Recommendation Based on Collaborative Aggregation of Interest Feedback. *Software: Practice and Experience*, 43(10):1165–1184, 2013.

[20] Daniel C Howe and Helen Nissenbaum. TrackMeNot: Resisting Surveillance in Web Search. *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society*, pages 417–436, 2009.

[21] E. Kushilevitz and R. Ostrovsky. Replication is not Needed: Single Database, Computationally-Private Information Retrieval. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 364–373, 1997.

[22] I. Goldberg. Improving the Robustness of Private Information Retrieval. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 131–148, 2007.