On Unstructured Distributed Search over BitTorrent

William Mayor
University College London
Gower Street, London, UK
Email: w.mayor@ucl.ac.uk

Ingemar Cox
University College London
Gower Street, London, UK
Email: i.cox@ucl.ac.uk

Abstract

Current BitTorrent tracking data discovery methods rely on either centralised systems or structured peer-to-peer (P2P) networks. These methods present security weaknesses that can be exploited in order to censor or remove information from the network. To alleviate this threat, we propose incorporating an unstructured peer-to-peer information discovery mechanism that can be used in the event that the centralised or structured P2P mechanisms are compromised. Unstructured P2P information discovery has fewer security weaknesses. However, in this case, the performance of the search is both probabilistic and approximately correct (PAC) since it is not practical to perform an exhaustive search. The performance of PAC search strongly depends on the distribution of documents in the network. To determine the practicality of PAC search over BitTorrent, we first conducted a 64 day study of BitTorrent activities, looking at the distribution of 1.6 million torrents on 5.4 million peers. We found that the distribution of torrents follows a power law which is not amenable to PAC search. To address this, we introduce a simple modification to BitTorrent which enables each peer to index a random subset of tracking data, i.e. torrent ID and list of participating nodes. A successful search is then one that finds a peer with tracking data, rather than a peer directly participating in the torrent. The distribution of this tracking data is shown to be capable of supporting an accurate PAC search for torrents. We assess the overheads introduced by our extension and conclude that we would require as little as a 0.24% increase over current global P2P traffic. We also simulate our extension under network churn in order to confirm its effectiveness.

On Unstructured Distributed Search over BitTorrent

I. Introduction and Motivation

BitTorrent is a popular method of distributing multimedia content, software and other data. BitTorrent requires users to interact with a centralised service called the tracker. This central focus point for the protocol is a potential security weakness that could be exploited to disrupt the network. Studies have shown tracker failure to be a common and disruptive occurrence [1]. Disrupting the tracker's service effectively halts the running of the BitTorrent network. In order to strengthen the network to attack, a distributed hash table (DHT) extension has been introduced and widely adopted. The DHT spreads the responsibilities of the tracker across the network and thereby makes it more difficult to disrupt the tracking service. However, whilst this improves the security of BitTorrent, the DHT mechanism is also vulnerable to attack. For example, [2], [3] show that the most popular DHT implementation for BitTorrent allows malicious nodes to passively monitor nodes and even remove nodes from the network.

Information discovery or retrieval in an unstructured P2P network is more resistant to attacks attempting to censor that information [4]. However, since it is not practical to search the entire network, the accuracy of such search is both probabilistic and approximate. Recent work on modelling probably approximately correct (PAC) search has provided a strong mathematical framework for modelling the search accuracy, i.e. the probability of finding a torrent, which is primarily a function of the number of nodes queried and the number of nodes a torrent is replicated onto [5], [6], [7].

To determine whether PAC search in feasible on the BitTorrent network, we first conducted a 64 day study of BitTorrent activities, looking at the distribution of 1.6 million torrents on 5.4 million peers. Our measurements show that the torrent distribution across nodes follows a power-law. The vast majority of torrents are known to very few nodes. Section III describes this work. Given the current distribution of torrents, a probabilistic search is unlikely to succeed.

Currently, nodes in a BitTorrent network are only aware of the torrents that they are participating in. In order for a PAC search to be successful, a search query must reach at least one node that is aware of the torrent searched for. Unfortunately, finding an aware torrent is difficult, as Section III reveals. To improve the probability of a successful search, Section IV proposes a modification to the BitTorrent protocol such that, if a node receives a query for a torrent it is unaware of, it stores the torrent's ID, together with the address of the querying node. If the queried node then receives a subsequent query for this torrent, it responds with the address of the previous querying node(s). This modification assumes that the previous querying node successfully found the torrent and, as a participating node, can now service requests for the torrent. This is a strong assumption, but previous research [8] has shown that P2P users are persistant, and our theoretical analysis indicates that a user need issue a query no more than three times to have a very high probability of finding the torrent. This modification substantially improves the awareness of a torrent in the network and thereby significantly increases the probability of a successful PAC search. Section IV provides a detailed analysis.

Section V then considers the overheads associated with the modification of the protocol. Additional bandwidth is required in order to discover torrents and each node must provide a small amount of local storage for indexing purposes. We show that even under extreme conditions the overheads introduced by this extension are not prohibitive. Search queries cost between 7KB and 20KB and increase the global P2P bandwidth consumption by as little as 0.24%.

In order to verify our model we run a series of extensive simulations in Section VI. These simulations confirm our theoretical analysis. The simulations also consider various models of network churn in order to demonstrate the extension's effectiveness in real-world environments.

II. Background and Related Work

We first provide a brief introduction to the BitTorrent protocol, including defining the terminology associated with BitTorrent. We then summarize some results from PAC search that we require for our theoretical analysis.

A. The BitTorrent Protocol

The BitTorrent protocol [1] is a mechanism for distributing files around a large number of computers in a peer-to-peer network. It was designed to allow many users to concurrently download files without demanding excessive bandwidth from any single machine. This is achieved by first partitioning a file into many *pieces*. A node then downloads these pieces from many other nodes in the network and subsequently merges the pieces to recover the original file. Each piece of a file is small enough that, individually, they are easy to supply. The requesting node receives a *torrent* of these small pieces that it can combine to form the desired file. When the requesting node receives a piece of the file it can also start to offer that piece to other nodes. Since pieces of popular files will reside on many nodes, BitTorrent also provides an inherent load balancing capability.

In order to share data over BitTorrent, an author or publisher of a file must create a meta-data file called the *.torrent file*. Each .torrent file contains (i) a list of the pieces that constitute the file, (ii) the URL of the torrent's tracker, and (iii) an identifier for the torrent. This identifier, called the *infohash*, is used to negotiate the download.

[1]http://bittorrent.org/beps/bep_0003.html

A *tracker* is a centralised server that monitors the sharing of data in the network. For every torrent a tracker is responsible for, the tracker keeps a list of the peers that are participating in the uploading or downloading of the torrent's data. When a new node enters the network it can request a copy of this list from the tracker. The new node can then contact the nodes listed and start to request pieces. Nodes that are downloading or uploading data must periodically communicate with the tracker in order to keep the list up-to-date. We refer to the combination of the torrent's unique ID, i.e. the infohash, and the list of nodes participating in the torrent as the *tracking data*.

The centralised nature of the tracker is a security weakness as attackers can attempt to disrupt the tracker. This weakness is well recognised by the BitTorrent community and several solutions have been adopted. The multi-tracker extension[2] to the protocol allows torrent authors to list more than one tracker URL in the torrent. If one tracker fails, the node can attempt to contact a second and is therefore less affected by individual tracker failure. HTTP[3] and FTP[4] seeding extensions allow torrent authors to make their files available via a direct HTTP/FTP connection. If the trackers are unavailable then nodes can fall back to a more traditional form of direct downloading. Tracker exchange[5] lets nodes share information on BitTorrent trackers. Using this extension, nodes can learn which trackers are the most popular or robust for a particular torrent. Each of these solutions provide additional security by replicating the centralised services. This tactic protects against accidental failure but can do little to prevent a coordinated attack. Each individual service remains susceptible to disruption.

The peer exchange (PEX) [9] extension enables node discovery without using a tracker. Peer exchange lets nodes share tracking data directly. There are several, independent, implementations of the peer exchange protocol[9], each achieves the same goal. When two nodes, participating in a torrent, communicate with each other, they can optionally choose to exchange tracking data. Each node sends a list of other nodes that it knows to have recently joined or left that torrent. In doing this nodes are made aware of new nodes to contact and old nodes not to contact without needing to poll the tracker. Reducing traffic to the tracker means that it is less likely to become overloaded. If the tracker were to fail nodes can still successfully gather tracking data. PEX only works if a node is aware of at least one other node. It does not, therefore, remove the requirement for the tracker in the first place. Our BitTorrent modification in Section IV uses the fact that if details of a single node can be discovered then details of other nodes can be shared using PEX.

The distributed hash table (DHT) extension[6] moves the tracking data from the tracker into a shared and distributed database. BitTorrent uses an implementation of a Kademlia DHT system[10] that enables nodes that are new to the network to retrieve a torrent's tracking data without requesting anything from a tracker. The DHT extension to the BitTorrent protocol successfully removes the requirement for a centralised tracking service, a single point of failure in the original protocol. Unfortunately, there are some side effects to this DHT implementation which may introduce new security concerns as well as potentially undermining some of the previously assumed benefits. For example, in [2] a Sybil attack, where many nodes are controlled by a single entity, is performed that successfully pollutes the DHT and manages to eclipse targeted torrents. Eclipsed torrents are effectively removed from the DHT by making them undiscoverable.

B. Probably Approximately Correct Search

Probably approximately correct (PAC) search, introduced in [7], is an information retrieval mechanism that operates under a similar maxim to BitTorrent; let many machines do small amounts of work. PAC search is a system that enables full-text search over a document collection. The collection is randomly distributed around a network of nodes, each node holding a small fraction of the total collection in a local index. Documents are duplicated across nodes to provide redundancy. To perform a search, a node issues a query to a randomly sampled set of nodes. Each node applies the query to its local index and returns any matching documents. The results from all queried nodes are collated and duplicate results removed. If the resulting document set does not meet the search requirements the query can be re-issued to a newly sampled set of nodes.

PAC search distributes text-search tasks across multiple nodes and so reduces the workload of each node. PAC can scale to accommodate large collections, as additional nodes can be easily added to the network[7]. Documents can be added and removed from the collection without requiring any complex re-partitioning. It achieves these goals at the expense of accuracy and efficiency[11], [6]. In comparison to a deterministic full-text search system, PAC search is unlikely to be able to return the exact same results. Given the overheads introduced for network communications, a PAC search request is also likely to take longer to return and may require more bandwidth.

There are three factors that influence the ability of a PAC search system to correctly retrieve a document, d_i, namely (i) the number of nodes in the network, n, (ii) the number of nodes that index the document, r_i, and (iii) the number of nodes contacted per query, z. A search over a collection of documents distributed across n nodes involves a node querying z other nodes. Each of the z nodes will perform a search for the document across their local index and return any matching results. A document can only appear in PAC search results if it is present in the local index of at least one of the z nodes that were queried. From [7] the probability, $P(d_i)$, that document d_i is present in at least one of the z local indexes is given by:

$$P(d_i) = 1 - (1 - \frac{r_i}{n})^z \tag{1}$$

In the context of BitTorrent, we are performing a *known item* search, where we are searching for one and only one document,

[2]http://bittorrent.org/beps/bep_0012.html
[3]http://bittorrent.org/beps/bep_0017.html
[4]http://bittorrent.org/beps/bep_0019.html
[5]http://bittorrent.org/beps/bep_0028.html
[6]http://bittorrent.org/beps/bep_0005.html

uniquely identified by its infohash. As such, broader definitions of accuracy introduced in [11], [6] are not relevant and Eqn (1) provides the probability of a successful search. In Section III we observe 5.4 million unique nodes in the BitTorrent network. Using this value and Eqn (1) we can calculate the number of nodes that a document needs to be replicated over in order to achieve a given accuracy. For example, if $P(d_i) = 0.8$ and $z = 100$ then we would require a document to be replicated across 86,214 nodes, i.e. 1.6% of the network. If we were to contact more nodes per query then our replication requirement decreases, for example $z = 500$ requires a document to be replicated across 17,354 nodes (0.32%).

Information retrieval in unstructured networks is a well-researched area. In [12] the authors study the performance of search using different replication strategies; uniform, proportional and square-root. They conclude that uniform and proportional strategies, where documents are distributed uniformly or according to their popularity respectively, provide equal average performance. Square-root replication, where documents are distributed over a number of nodes proportional to the square root of their popularity, performs optimally. In [13] the authors introduce BubbleStorm, a system for search over unstructured peer-to-peer networks, very similar to PAC search. BubbleStorm provides a gossiping algorithm that is very resilient to network churn and wide-scale crash.

In this paper we do not consider the issue of peer sampling, that is, we assume that a PAC search client is capable of taking a uniformly random sample of nodes from the network. Methods that achieve this goal are numerous. BubbleStorm uses local-view gossiping to achieve this. In [14] the authors consider using random walks over an unstructured networks to replace flooding found in systems such as Gnutella. In [15] the authors introduce Brahms, a system for random peer sampling in unstructured networks that uses another gossip-based protocol. Brahms also provides security measures for sampling in a Byzantine environment.

III. The Measurement Study

In order to evaluate the applicability of PAC search to BitTorrent tracking data we conducted a measurement study of public BitTorrent networks. In order to evaluate PAC search over BitTorrent it is necessary to know: (i) how many nodes are in the network and (ii) how torrents are distributed over those nodes.

A. Design

The data was collected from public BitTorrent trackers discovered using the TrackOn[7] API. Using the API we gathered a list of online public trackers. Public BitTorrent trackers are public facing trackers that place few restrictions on use. Any torrent can be registered at the tracker and any BitTorrent node can communicate with it. Each tracker was periodically polled with a *scrape* request. A scrape request asks the tracker to return a list of all of the torrents that it is tracking. For each torrent in the scraped list, the tracker was asked to list all of the nodes that were currently sharing that torrent's file. For each result we stored the following information: `timestamp tracker ip:port infohash`.

This process was initiated at most once an hour. In practise a complete scrape took much longer than an hour to complete so additional scrapes were only started when the previous finished. Nodes were identified by IP address, it was assumed that each unique IP address represents a unique node and that every node has a single, unchanging IP address. This means that we cannot tell the difference between nodes that operate behind network address translation (NAT) and so may, as a result, undercount the number of nodes. We also cannot distinguish between nodes that share an IP address, for instance if an ISP reallocates an IP address to a different node. Again, this means that we undercount the number of nodes. We assume that these issues impact only slightly on our figures.

B. Results

Between 1st May and 3rd July 2012, 13 public BitTorrent trackers were periodically scraped[8]. A total of 1.6 million distinct torrents were observed on over 5.4 million distinct nodes over 64 days. Considering each torrent as a document in the collection, the number of documents, $m = 1,600,000$ and the number of nodes $n = 5,400,000$. A PAC search is heavily influenced by the distribution of torrents over the nodes. In order to determine this distribution, the number of nodes registered to each torrent was counted. The frequencies at which these counts were observed was then calculated. Figure 1 shows these frequencies on a log-log scale, along with a line of best fit. The distribution follows a power law with the vast majority of torrents being found on very few nodes. These figures align roughly with those observed in [16], [17]. The analysis shows that 25% of all torrents are only found on a single node and 76% of all torrents are found on 10 or fewer. Only 2% of observed torrents were found on more than 100 nodes. Torrents were found on anywhere between 0 and 21,445 nodes, the average torrent was owned by 27 nodes and the median torrent by 3. We saw in Section II that documents needed to be replicated on tens of thousands of nodes in order to have a high probability of successful search. We observe that very few torrents meet those requirements.

IV. The PAC BitTorrent Extension

Until now, our analysis has assumed that in order to find a torrent and begin the download process a node must directly

[7]http://www.trackon.org

[8]Those trackers were:
http://bttrack.9you.com
http://exodus.desync.com:6969
http:/announce.xxx-tracker.com:2710
http://h33t.com:3310
http://bt.rghost.net
http://61.154.116.205:8000
http://fr33dom.h33t.com:3310
http://announce.opensharing.org:2710
http://bttrack.9you.com:8080
http://tracker.torrentbay.to:6969
http://bigtorrent.org:2710
http://tracker.coppersurfer.tk:6969
http://a.tv.tracker.prq.to

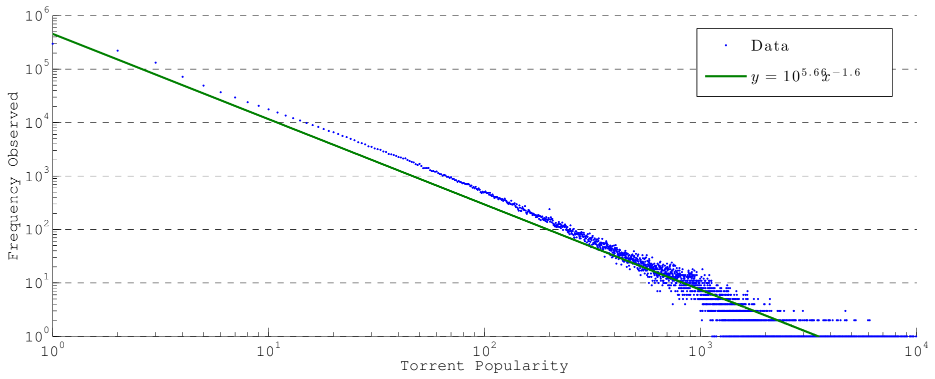

Fig. 1. The frequency of observations of torrent popularity, i.e. the number of nodes that are participating in the torrent

identify any node currently participating in that torrent. This need not be the case. Rather, to find a torrent, we could first discover the torrent's tracking data. The tracking data will contain a list of nodes thought to be participating in the torrent, these nodes can be used to initiate the download. This is, of course, analogous to the original BitTorrent protocol's torrent-discovery-via-trackers mechanism. To accomplish this in a unstructured P2P environment, we need a mechanism by which each peer indexes a random, non-disjoint subset of torrent tracking data. Given this mechanism, we can apply PAC search on the collection of tracking data, where each torrent's tracking data is equivalent to a document. The success of PAC search is then dependent on the distribution of torrent tracking data, rather than the distribution of the torrents themselves.

In Section IV-A we first introduce the modification to the BitTorrent protocol that enables peers to index a random, non-disjoint subset of torrent tracking data. Section IV-B provides a mathematical model of our extension. Section IV-C then analyses the distribution of torrent tracking data and the associated performance of PAC search.

A. Indexing

The ability of a PAC search system to return successful results is influenced by these two, controllable, factors: (i) the distribution of documents across the network, r_i, and (ii) the number of nodes contacted per query, z. The distribution of torrents over nodes discovered in Section III prevents the vast majority of torrents from being discovered no matter how large z. Thus, as well as modifying z, we must also alter the distribution of the tracking data, r_i. From an information retrieval perspective, we do not need to replicate the file, or any pieces of the file, to more nodes. Rather, we need more nodes to be aware of where the torrent can be downloaded from. We need nodes in the network to store the torrent's unique ID and the list of peers participating in the torrent (tracking data), *even if the node itself is not participating in the torrent*. To this end, we introduce an index at each node that contains details of previously received requests. When a node is queried for a torrent that it does not own, the queried node stores details of the request in its index, i.e. storing the torrent's unique ID alongside the details of the node that made the request. If the queried node receives any future requests for that torrent, it can now respond with details of a node that it believes to own the torrent. When the querying node receives such a response, it can the contact these nodes directly. Thus, when a node performs a *search* it issues one or more queries for a torrent, until such time as a query is successful. A *query* consists of a node sending a request to z randomly sampled nodes in the network. Each repeated query for the same torrent selects z different nodes. A successful query is one that results in at least one successful request. A *request* consists of a querying node sending the desired torrent's infohash to one of the z random nodes. The queried node responds with either a list of nodes it believes are participating in the torrent, or an empty list. A successful request returns a list of actively participating nodes; nodes that are online and that are downloading or uploading the torrent's data. A unsuccessful request returns either an empty list, or a list of nodes that are not participating in the torrent, e.g. because they have left the network.

B. Model

When querying a fixed number of random nodes, z, the probability of a successful query is now determined by (i) the number of nodes participating in the torrent, and (ii) the number of nodes indexing the torrent's tracking data, r_i. Given our proposed modification to the BitTorrent protocol, the more queries the network receives for a torrent, the more that torrent's tracking data is replicated, and the easier it becomes to find. At any point in time the number of nodes indexing a torrent, $r(t)$, additionally depends on: the number, u, of requests made for the torrent, and the proportion, c, of nodes that have left the network. The change in replication over time can be expressed as:

$$\frac{dr(t)}{dt} = u(1 + z(1 - \frac{r(t)}{n})) - cr(t) \tag{2}$$

Here, $(1 - \frac{r(t)}{n})$ gives the proportion of the z nodes that were not already indexing the torrent. We can solve this ODE to give us an equation for the replication as a function of time:

$$r(t) = ke^{-t(c+\frac{uz}{n})} + \frac{un(1+z)}{uz+cn} \tag{3}$$

The constant, k, is given by the initial condition, $r(0)$. Conceptually, this value is the number of nodes that index the torrent before any queries have been made for it. $r(0)$ can be decided by the torrent's authoring node and represents a bootstrap value that enables early queries to be successful. For a fixed z the value for $r(0)$ can be tuned to enable a desired probability of success using Eqn 1. The authoring node simply makes dummy requests to $r(0)$ nodes in order to push tracking data into the network.

NOTES.

1) With this constant u and c the replication exponentially tends towards $\frac{un(1+z)}{uz+cn}$
2) I have removed the assumption that nodes continue to query until they are successful. This makes the equations simpler. It removes the safety of knowing that a query definitely increases the replication because it is very possible that a single query is not successful and therefore those requests have not increased tracking data, merely lots of red herrings.
3) We can add this assumption back in by: increasing the query rate according to the probability of success at t, or fixing $P(d)$. I'm more inclined towards the former. EIther might make the ODE equation too hard to solve again.
4) Replication increases to the limit if $r(0) < \frac{un(1+z)}{uz+cn}$ and decreases if $r(0) > \frac{un(1+z)}{uz+cn}$

C. Discussion

We saw in Section **??** that because of the distribution of torrents over nodes, PAC search would only be successful for X% of the queries made. Our BitTorrent extension replicates torrent tracking data in order to improve PAC search performance. In order to analyse the performance gain consider a BitTorrent network in a steady state. Where the number of queries for any torrent is constant over time and the proportion of nodes that leave the network (the churn) is such that total network size remains constant also. We assume that the popularity distribution measured in Section III is proportional to the distribution of queries made for the torrent (a reasonable assumption in this steady state). The performance of PAC search over time is then dependent on the behaviour of the replication of the tracking data for the torrent searched for. If the replication increases over time then the torrent will always be discoverable and a worst-case can be guaranteed. If the replication is decreasing over time then eventually the torrent will become undiscoverable. From Eqns **??** we know that...

V. The Overheads

The BitTorrent extension described in the previous section introduces a number of overheads. The introduction of an additional index at each node requires additional local storage space. Discovery of tracking data is achieved by having nodes make requests of relatively many other nodes. This requirement is likely to demand more bandwidth than the current BitTorrent protocol. We shall now quantify these overheads.

Bandwidth consumption is calculated using the following assumptions and generalisations about communication costs:

1) TCP/IP overhead amounts to 64 bytes per packet[9]
2) We never send more bytes than can fit into a single packet (1500 bytes)
3) BitTorrent requires an initial 51 bytes for a handshake[10]
4) BitTorrent adds 4 bytes of length prefix per message[10]
5) A torrent infohash is 20 bytes[10]
6) Results can be communicated using 6 bytes per node[11]

A. Single Node Sending Single Query

Using these assumptions we can estimate the communication cost for a node to send a query. For each query the node will contact z other nodes, and receive a response from all of them. An unsuccessful response will contain no details of other nodes. A successful response will contain some number of results. A successful query is one in which at least one response was successful. For a successful query, the total cost depends on, (i) the number of successful requests and, (ii) the number of nodes listed in each response. For simplicity, we consider three cases; responses contain the details of a single node, of ten nodes and of 100 nodes. In practise, the size of a response will vary according to how many nodes each of the queried nodes are aware of. The cost, in bytes, to send a query is $z(64 + 51) = 115z$. An unsuccessful response costs $64 + 4 = 68$ bytes. A successful response costs $64 + 4 + 6a = 68 + 6a$ bytes, where a is the number of results returned, $a = 1, 10,$ or 100. The mean number of successful responses to a query can be determined by taking the expected value of the binomial distribution where each trial (each request) has probability of success $r(x)n^{-1}$. There are z trials (requests) made per query and therefore $zr(x)n^{-1}$ successful responses on average. The expected cost of a query is:

$$\begin{aligned}
C_1 &= \text{cost to send query} && (4)\\
&= 115z && (5)\\
C_2 &= \text{cost to receive responses} && (6)\\
&= 68z(1 - r(x)n^{-1}) + (68 + 6a)zr(x)n^{-1} && (7)\\
C &= \text{total cost for a single query} && (8)\\
&= C_1 + C_2 && (9)\\
&= z(183 + 6ar(x)n^{-1}) && (10)
\end{aligned}$$

Table I shows cost of querying for torrents of varying popularity using several PAC Search strategies. Communication cost increases linearly with z but we know from Eqn (1) that the probability of a successful query increases exponentially with z. When the tracking data is highly replicated, e.g.

[9]http://sd.wareonearth.com/~phil/net/overhead/

[10]http://bittorrent.org/beps/bep_0003.html

[11]Details of the IP and port for each node will make up the results list. http://bittorrent.org/beps/bep_0023.html explains how BitTorrent clients communicate IP:port combinations in 6 bytes.

TABLE I. THE COMMUNICATION COST, C, IN BYTES, FOR COMPLETING A SINGLE QUERY FOR A TORRENT UNDER DIFFERENT SITUATIONS. THE COST OF THE QUERY IS SHOWN FOR DIFFERENT POPULARITIES, x, WITH VARYING NUMBERS OF NODES CONTACTED, z, AND ALSO ACCORDING TO HOW MANY RESULTS ARE RETURNED, a.

Strategy		Popularity	Awareness	Single Query Cost (bytes)		
z	r(1)	x	r(x)	$a = 1$	$a = 10$	$a = 100$
100	1,000	1	1,001	6,916		
500	1,000	1	1,001	34,116		
500	5,000	1	5,001	34,116		
1,000	5,000	1	5,001	68,117		
100	1,000	10	11,273	6,917	6,929	
500	1,000	10	12,986	34,122	34,183	
500	5,000	10	12,999	34,122	34,183	
1,000	5,000	10	15,843	68,129	68,251	
100	1,000	100	35,731	6,920	6,958	7,344
500	1,000	100	60,505	34,137	34,330	36,259
500	5,000	100	60,514	34,137	34,330	36,259
1,000	5,000	100	105,080	68,158	68,544	72,403
100	1,000	1,000	146,306	6,933	7,091	8,671
500	1,000	1,000	486,543	34,203	34,993	42,894
500	5,000	1,000	486,551	34,203	34,993	42,894
1,000	5,000	1,000	912,316	68,291	69,871	85,672
100	1,000	10,000	956,159	7,030	8,063	18,389
500	1,000	10,000	3,170,983	34,689	39,852	91,485
500	5,000	10,000	3,170,986	34,689	39,852	91,485
1,000	5,000	10,000	4,328,606	69,263	79,589	182,854

$x = 10,000$, we observe that the communication cost can increase sharply with a. This is due to the fact that more of the z queried nodes are able to respond with tracking data for such a high popularity torrent. Of course, for such a popular torrent, we can decrease z, and therefore the communication cost, and still maintain a high probability of success. For instance, if a torrent is owned by $x = 10,000$ nodes then $r(x) >= 950,000$, substituting these values into Eqn (1) and using $P(d_i) = 0.95$ we see that we would need only $z = 15$. Such a query would cost less than 3KB, depending on the size of the responses. This example illustrates the need to adaptively adjust the size of the number of nodes queried, z, based on the awareness of a torrent. The difficulty is that a querying node is unlikely to know the awareness beforehand. Given our assumption that a node repeats a query until successful, a simple strategy would be to initially query a small number of nodes and then to increase this value if a query is unsuccessful. A more detailed analysis of adaptively selecting the number of nodes queried is outside the scope of this paper, and we assume, for simplicity, that z is fixed. The corresponding communication cost can therefore be considered a worst case estimate.

B. Authoring Node Bootstrapping

In order for a low popularity torrent to be discoverable in the network it is necessary for the few owning nodes to gossip details of themselves around the network. This is achieved by sending dummy requests to nodes. We have previously seen values of $r(1) = 1000$ and $r(1) = 5000$. Performing this "bootstrap" mechanism for these values would require $C_3 = r(1)(64+51)$ bytes; respectively 115KB and 575KB. Note that this communication cost may be incurred over minutes, hours or days and therefore requires almost negligible bandwidth on the part of the publishing node.

Eqn (1) shows the relationship between z, the number of nodes contacted per query, r_i, the replication of a torrent, and $P(d_i)$, the probability that the query is successful. We can see that if we desired a certain probability of success from the system, that there are multiple strategies of z and r_i that would achieve it for us. We might wish to select a strategy that minimises the total communication cost of the system. Consider discovering a single torrent in the network. At any point in time the total communication cost for the torrent is the sum of (i) the costs incurred replicating that torrent's tracking data to its current state, and (ii) the costs incurred by any nodes currently searching for the torrent. When the tracking data is being bootstrapped into the network the authoring node has complete control over the replication and can therefore control the probability of success and the communication cost. If u nodes query for the torrent after the author has released it then we can calculate the communication cost of the torrent as:

$$z = \frac{\log(1 - P(d_i))}{\log(1 - \frac{r(1)}{n})} \quad \text{from Eqn 1} \tag{11}$$

$$uC_1 = \frac{115u \log(1 - P(d_i))}{P(d_i) \log(1 - \frac{r(1)}{n})} \tag{12}$$

$$uC_2 = \frac{(68 + 6a\frac{r(1)}{n})u \log(1 - P(d_i))}{\log(1 - \frac{r(1)}{n})} \tag{13}$$

$$C_3 = r(1)(64 + 51) \tag{14}$$

$$\text{total} = C_3 + u(C_1 + C_2) \tag{15}$$

Where $P(d_i)$ is the desired accuracy of a query and $r(x)$ is the replication. By minimising the total cost we can calculate the optimal bootstrap replication, $r(1)$, and therefore the optimal z for future nodes. To demonstrate, we calculate these optimal values for a range of query volumes, $0 < u < 1,000$. Figure 2 shows these optimal $r(1)$ and z values over the range of query volumes. If a torrent is likely to be rarely requested, then the best strategy is to replicate to very few nodes and let searching nodes repeatedly query for the torrent. If, on the other hand, a torrent is going to be often requested, then it will cost less to replicate to many nodes, when the single authoring node incurs the majority of the cost, not the many querying nodes.

The authoring node of a torrent is unlikely to be aware of the torrent's popularity before releasing it. It is therefore difficult for the authoring node to determine the optimal starting replication, $r(1)$. It also difficult to know what value of z is best suited for a torrent with an unknown popularity. Currently, our system will continue to increase the replication of a torrent's tracking data as the torrent gains popularity. A dynamic z value could be used to ensure that the replication remains as close to optimal as possible. This is left as future work.

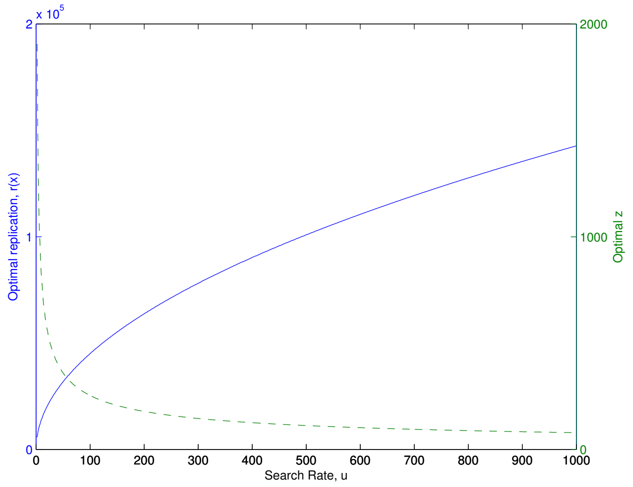

Fig. 2. The optimal bootstrap replication, $r(1)$, and z values required to minimise the communication cost for a torrent with u concurrent searches being performed for it.

C. *Single Node Responding to Multiple Queries*

In order for searches to be successful, queried nodes must respond to requests made of them. In order to estimate the communication cost of providing responses we need to know the number of times a node will be contacted and how large each response will be. When searching for a node, the expected number of requests made is $zP(d_i)^{-1}$. In [16] the authors observe that the average BitTorrent node will perform $q = 1.33$ searches every hour. This is supported by Internet usage reports from Cisco[12]. We can now estimate the number of requests generated by the entire network every hour:

$$qz\sum_{j=0}^{n} P(d_{n_j})^{-1} \tag{16}$$

Where d_{n_j} is the document that node j is searching for. These requests will be split uniformly over all of the nodes, each receiving an expected $\frac{1}{n\text{th}}$ of the total. To derive an estimate for the number of requests each node is likely to received every hour we can use the mean popularity of the torrents we observed in Section III, 27. Table II shows the expected number of requests received by a node per hour for different values of z and $r(1)$. Each of the received requests requires a response, the size of the response depends on the node's awareness of the torrent requested. A node's awareness of torrents will increase the longer it remains in the network.

[12]Cisco's Visual Networking Index (VNI) puts the total peer-to-peer file sharing bandwidth consumption at 4,656 PB per month in 2011. Assuming that files downloaded over BitTorrent are, on average, 1GB in size. This tells us that $4656 \times 10^{15} \times 10^{-9}$ files are downloaded every month and that $4656 \times 10^{6} \times (24 \cdot 7 \cdot \frac{52}{12})^{-1} = 6,395,604$ downloads complete every hour. Assuming bandwidth speeds and request rates are constant, we have that BitTorrent users request nearly 6.5 million torrents every hour. We observed 5 million nodes and therefore conclude that each node generates $q = 1.28$ requests per hour

TABLE II. THE EXPECTED BANDWIDTH (BYTES/SEC) USED BY A NODE FOR RESPONDING TO QUERIES.

Strategy			Multi-request Response Cost (bytes/sec)		
z	r(1)	#requests/hour	$a = 1$	$a = 10$	$a = 100$
100	1000	438	9	16	81
500	1000	736	15	26	137
500	5000	736	15	26	137
1000	5000	1332	27	47	247

Again, for simplicity, we shall assume that the node will always respond with the details of $a = 1, 10,$ or 100 nodes. This simplification gives us the upper bound on the cost of responding, see Table II:

$$\begin{aligned} R &= \text{requests recevied} && (17)\\ &= \frac{1}{n}qz\sum_{j=0}^{n}(1-(1-\frac{r(27)}{n})^z)^{-1} && (18)\\ &= \frac{qz}{1-(1-\frac{r(27)}{n})^z} && (19)\\ \text{total cost} &= (68+6a)R && (20)\end{aligned}$$

In the worse case scenario ($z = 1000$ and $r(1) = 5000$), we observe that queried nodes would expend no more than 247 bytes/sec, an amount easily provided, given current home broadband capabilities.

D. *Global Bandwidth*

The total traffic generated by a PAC Search system over BitTorrent can be estimated by taking the product of the number of requests generated by each node, the number of nodes and the average cost of a request and response. As in the previous section we use the popularity of the average torrent for simplicity:

$$\frac{nqz(183+\frac{6ar(27)}{n})}{1-(1-\frac{r(27)}{n})^z} \tag{21}$$

where $q = 1.33$ is the query rate and $n = 5,000,000$ is the number of nodes in the network. The expected global communication costs are given in Table III. For low z and $r(1)$ values we see a very small increase in global traffic, less than a quarter of 1%. For larger values we see a much larger increase, just over 7% in the worst cases. The cost of each strategy represents the expected costs incurred for successful searches, so we do not increase the failure rate by decreasing z and $r(1)$. These data would suggest that low valued strategies are best if one wishes to reduce overall bandwidth consumption. These costs may be further reduced by applying a dynamic z strategy, as discussed in previous sections.

E. *Storing the Index*

In addition to using bandwidth to query and respond, each node must keep a local index of the torrents and nodes it is

TABLE III. Global bandwidth estimates in gigabytes per second for four different strategies of z and $r(1)$ and three choices of a. The percentage increase over current global P2P traffic is also given.

Strategy		Global Bandwidth Cost (GB/sec)		
z	r(1)	$a = 1$	$a = 10$	$a = 100$
100	1000	4 (+0.24%)	4 (+0.24%)	4 (+0.24%)
500	1000	35 (+1.96%)	35 (+1.97%)	36 (+2.03%)
500	5000	35 (+1.96%)	35 (+1.97%)	36 (+2.03%)
1000	5000	126 (+7.09%)	126 (+7.11%)	130 (+7.32%)

aware of. If we assume that each received request is for a distinct torrent then each item in the index will require 26 bytes, 20 for the infohash and 6 for the node's IP and port details. For a received-request rate, s per hour, and total hours of operation, h, the local index size has an upper bound of $26sh$. The local index increases in size at a constant rate. In practise, requests will be received for torrents that are already in the index and so will only require an additional 6 bytes per request. Therefore we show here the worst-case scenario.

Using the values in Table II for the expected number of requests received per hour we can build an estimate for the upper bound of the local index. The highest received-request rate in Table II is 1332 per hour. At this rate a node's local index would reach 1GB after 28,876 hours, or 3.3 years of continuous use. For the lowest received-request rate we would see an index reach 1GB after 10 years.

We see that the index size remains comfortably small even after extended usage. We consider then, that the most pressing reason for removing data from the index is to remove incorrect data. Data that suggests that a node owns a torrent when it does not. We offer three strategies for removing incorrect data:

Least Recently Used The local index is restricted to a small number of entries. When this entry limit is reached the oldest record is removed in order to make room for new ones. This method guarantees an upper bound to index size.. Unfortunately, it may remove perfectly good records from the index just because they happen to be the oldest. It may also do this when there exists incorrect data elsewhere in the index.

Periodic Ping This is the method employed by the BitTorrent DHT to ensure that records are always relatively up to date. When a record is added to the index a timer is set up that regularly pings the node to check if it is still online. If it is not online then it is marked as potentially incorrect. If a node receives several "incorrect" marks in a row then it is removed. This method will rarely remove good data from the index and potentially bad data is marked as such. Unfortunately, it requires additional bandwidth as nodes have to be periodically pinged.

Timed This method involves simply removing a record from the index when it has been there for a certain amount of time. As with least recently used, this method may remove good data.

We imagine that a combination of either timed or least recently used with periodic ping will provide the best solution. When a record is considered for deletion, the node is pinged to check for correctness. If the record was in fact correct then it is left in the index. If the record is incorrect it is removed. We leave the analysis of these (and other) solutions as future work.

VI. The Simulations

In order to assess our extension in a variety of different situations we ran several simulations. We looked to replicate different types of node behaviour and then assess different PAC strategies on each. Importantly, we introduced churn into the simulated networks. Our analysis so far has not accounted for churn, where nodes leave the network and so decrease awareness. These simulations, therefore, demonstrate our extension's effectiveness in a more true-to-life environment.

A. The Design

The simulation software was written in Java and is publicly hosted on GitHub[13]. The software simulates nodes in a peer to peer network, each making independent decisions about their search, download and upload behaviours. A round of the simulation consists of every node making a single action, these actions could be to start searching for or downloading a torrent, to leave the network, or simply to remain in the network. Each round is considered to take unit time. After each round, each node is quizzed regarding its state and the responses are stored. Rounds continue until there is no possibility of a successful download, this could be because all participants in the torrent have left, or perhaps because there is no awareness of the torrent in the network. A single trial of the simulation consists of a complete set of rounds. At the end of a trial the data gathered from each of the nodes in each of the rounds is collected and an average value for each statistic at each round is stored. Trials continue until either the minimum number has been run or we have obtained a 5% confidence interval for each of the statistics calculated, whichever is larger.

For our simulations we implemented two distinct patterns of node behaviour. One is based on the fluid model of peer to peer nodes developed in [18], [19]. A node's behaviour is decided according to which of four states the node is in; passive, downloading, seeding and dead. A passive node does not interact with a torrent directly but does respond to PAC search requests by building and maintaining a local index. All nodes are initialised as passive nodes, passive nodes can either remain passive for their lifetime or can instead be chosen to initiate a search. If a passive node has been chosen to become a downloading node then when it is initialised it decides a waiting time by sampling from an exponential distribution that models node arrival time. The parameters to the distribution can be tailored to suit the simulation. At each round the passive node decreases the wait time, when the time reaches zero the node performs a PAC search and becomes a downloading node. Downloading nodes remain available for PAC requests but at each round there are now two state checks to perform. The first is the download abort check, this is a waiting time, much like the passive node's arrival

[13]https://github.com/WilliamMayor/SimPact

time, that is sampled from a different exponential distribution. If the abort time reaches zero then the node moves into the dead state. The second check is to see if the download has completed, this check involves increasing the download complete percentage. The percentage increase at each round depends on the node's available download bandwidth and the collective upload bandwidth available from other, participating, nodes. If the download complete percentage reaches 100 before the abort time reaches zero then the node becomes a seeding node. Seeding nodes remain available for PAC search requests and contribute to the total available upload bandwidth in the network. At each round they decrease their seeding wait time, sampled from a third exponential distribution. If the seeding time reaches zero then the node becomes a dead node. A dead node is one that has left the network and is no longer available for requests. When a node leaves the network it is replaced by a fresh node in order to keep the network size constant. The fresh node shares no data with the departing node and they have differing network addresses. Passive nodes enter and leave the network according to the same behaviours as described above. The difference is that their status is not recorded. This is done in order to simulate network churn.

The second pattern of node behaviour is one described in [20]. The authors observe that a proportion of torrents display a constant popularity. We model this behaviour in two ways, the first is to keep a static number of nodes in the seeding state, the second is to churn the torrent's participants by adding a new node to the torrent every time an older one leaves. For both constant popularity models the passive nodes exhibit the same churn behaviour as before.

We ran 72 simulations, each for a minimum of 500 trials. We tuned the simulations to produce 18 different types of node behaviour and for each type we tested four different strategies for PAC search. In each simulation we fixed the network size to 5 million nodes and we simulated and observed a single torrent. A limit was set on the total lifetime torrent popularity, the maximum number of nodes that ever participate in the torrent. This was done to limit the runtime of the simulations and to help shape the popularity over time. For fluid model simulations we considered four different popularity shapes; large, medium, small and extra small. These were defined by the peak popularity over the torrent's lifetime. Large torrents peaked at 10,000 concurrently participating nodes, medium at 5,000, small at 1,000 and extra at only 100, see Figure 3 for an example. Additionally, fluid model simulations could exhibit standard, flash crowd or long tail node behaviours. In flash crowd simulations nodes seeded the torrent for less time than standard simulations. In long tail simulations nodes seeded for much longer, see Figure 4 for an example. Constant model simulations were either; large, with 1,000 participating nodes; medium, with 100; or small with 10. For each type of simulation we then considered the four different PAC strategies seen previously: $z = 100, r(1) = 1000$, $z = 500, r(1) = 1000$, $z = 500, r(1) = 5000$, and $z = 1000, r(1) = 5000$.

B. The Results

Figure 5 shows the probability of a successful query over time for a small sized fluid model simulation. The results

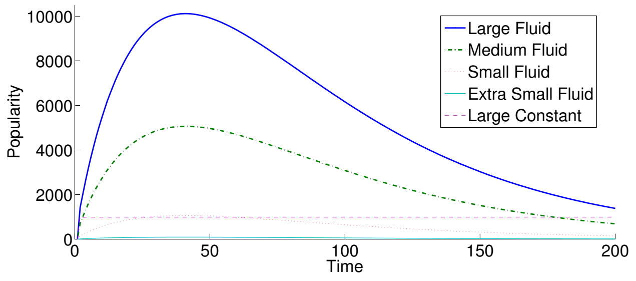

Fig. 3. The popularity over time of torrent in five different simulation types.

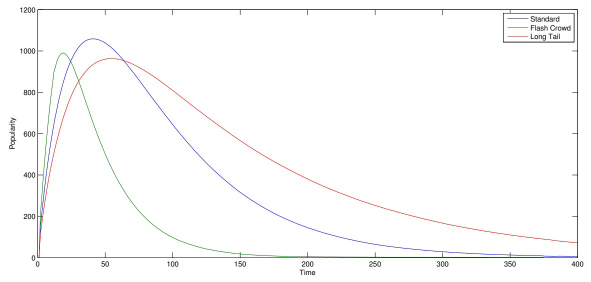

Fig. 4. The popularity over time of torrents with different peak popularity.

displayed use a PAC strategy of $z = 100$ and $r(1) = 1000$. Simulations of torrents with higher popularity display similar curves with a higher peak, lower popularity torrents display a smaller peak. Likewise when the PAC strategy uses higher values of z or $r(1)$. The probability of a successful query increases and decreases with popularity. This means that, taking into account *when* a query is likely to be performed, we see average probabilities of success over 80% for every simulation type. There is always a strategy of PAC search that provides a high probability of success, no matter how low the popularity of the torrent. Torrents that display a long-tailed popularity behave as expected; the probability reaches a similar peak at a similar time but remains higher for longer. Long-tailed torrents have a higher average probability of successful search. Torrents that demonstrate a flash-crowd popularity show interesting behaviour towards the end of the simulation; the probability of successful search starts to fluctuate. This is due to nodes searching for the torrent in a round where the awareness of the torrent is not high enough to easily support it. Those nodes are having to perform several queries repeatedly in order to discover the torrent and so are having a huge impact on the awareness of the torrent. When those late nodes leave the network, the awareness of the torrent subsequently drops by a large amount. Despite this, the average probability of successful search only drops below 70% in one case out of 16; where torrent popularity peaks at 325, $z = 100$ and $r(1) = 1000$, in this case the average is 27%.

For torrents that exhibit constant popularity we observe some interesting properties that can be exploited. For torrents with static participation the awareness of the torrent decreases only with the churn of the network. As aware nodes leave

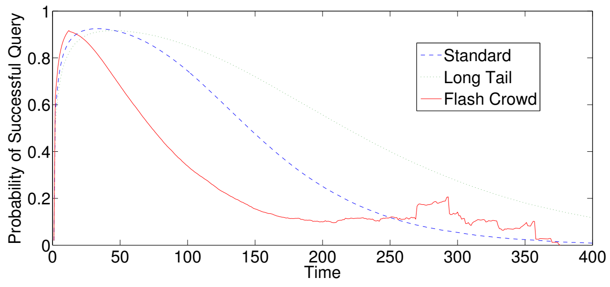

Fig. 5. The probability of successful search over time for fluid model popularities. Standard, long-tail and flash crowd torrents are show, each using a PAC strategy of $z = 100$ and $r(1) = 1000$.

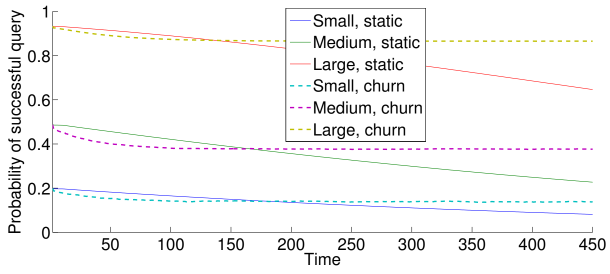

Fig. 6. The probability of succesful search over time for constant model popularities. Both static and churned popularities are shown, for large, medium and small sized torrents.

the network the probability of finding the torrent falls. To counter this decrease, the participating nodes can easily perform additional dummy queries, as when bootstrapping the torrent. If network churn is accurately modelled then an exact figure for the amount of dummy queries required can be calculated and the probability of a successful query can be easily maintained. For torrents with constant, but churning, participation we observe that awareness of the torrent remains as static as the popularity. This is because the steady stream of queries generating awareness offsets the network churn that decreases it. Figure 6 shows the probability of successful search over time for constant popularity torrents with churning and static participation of large, medium and small sizes.

VII. Conclusions and Future Work

In this paper we investigated the application of PAC search to BitTorrent tracking data. The study was motivated by an increasing awareness of the security issues inherent to the currently used BitTorrent protocols. PAC search offered a solution to the security issues but introduced an unknown drop in efficiency. PAC search's unstructured network design solves problems of censorship and information deletion. It does so at the expense of bandwidth costs, storage costs and information retrieval accuracies. This paper set out to determine the extent to which these negative factors would prohibit a PAC search solution over BitTorrent. We conclude that PAC search over BitTorrent tracking data is, generally, feasible. The models and strategies presented here outline a PAC solution capable of retrieving 70% of all observed torrents after a single search. Under normal working conditions current BitTorrent protocols can offer 100% retrieval accuracy. Under adversarial working conditions these same, existing, protocols can be reduced to 0% accuracy. The PAC search protocol offers a practical solution resistant to failures of this magnitude.

This paper does not look at the effect of node drop out on the performance of the system. Future work needs to be done to account for the likely lowering of search accuracy under network churn. Furthermore, some interesting work could be done looking at the effect of dynamically changing z. There would seem to be some efficiency increases possible if nodes start querying a small number of nodes and then increase z after an unsuccessful search.

Acknowledgments

Grant no: EP/G037264/1 as part of UCL's Security Science Doctoral Training Centre.

References

[1] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The BitTorrent P2P File-Sharing System: Measurements and Analysis," in *4th International Workshop on Peer-to-Peer Systems*, no. Oct, 2005, pp. 205–216.

[2] J. P. Timpanaro, T. Cholez, I. Chrisment, and O. Festor, "BitTorrent's Mainline DHT Security Assessment," in *NTMS - 4th IFIP International Conference on New Technologies, Mobility and Security*, 2011, pp. 1–5.

[3] E. Sit and R. Morris, "Security Considerations for Peer-to-Peer Distributed Hash Tables," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, vol. 2429, 2002, pp. 261–269.

[4] E. K. L. E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[5] H. Asthana and I. J. Cox, "PAC'n'Post : A Framework for a Micro-Blogging Social Network in an Unstructured P2P Network," in *Proceedings of the 21st international conference companion on World Wide Web*, 2012, pp. 455–456.

[6] I. Cox, J. Zhu, R. Fu, and L. K. Hansen, "Improving Query Correctness Using Centralized Probably Approximately Correct (PAC) Search," in *Proceedings of the 32nd European conference on Advances in Information Retrieval*, no. i, 2010, pp. 265—-280.

[7] I. J. Cox, R. Fu, and L. K. Hansen, "Probably Approximately Correct Search," in *Advances in Information Retrieval Theory*, 2009, pp. 2–16.

[8] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," in *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*, 2003, pp. 314—-329.

[9] D. Wu, P. Dhungel, X. Hei, C. Zhang, and K. W. Ross, "Understanding Peer Exchange in BitTorrent Systems," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, 2010, pp. 1—-8.

[10] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pp. 53–65, 2002.

[11] H. Asthana, R. Fu, and I. J. Cox, "On the Feasibility of Unstructured Peer-to-Peer Information Retrieval," in *Proceedings of the Third international conference on Advances in information retrieval theory*, 2011, pp. 125—-138.

[12] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks," in *ACM SIGCOMM Computer Communication*, 2002, pp. 177—-190.

[13] W. Terpstra and J. Kangasharju, “BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search,” *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 49—-60, 2007.

[14] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, “Making gnutella-like P2P systems scalable,” in *SIGCOMM ’03*, 2003, pp. 407—-418.

[15] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, “Brahms: Byzantine resilient random membership sampling,” *Computer Networks*, vol. 53, no. 13, pp. 2340–2359, 2009.

[16] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, “Measurements, Analysis, and Modeling of BitTorrent-Like Systems,” in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005, pp. 35—-48.

[17] G. Dán and N. Carlsson, “Power-Law Revisited: Large Scale Measurement Study of P2P Content Popularity,” in *Proceedings of the 9th international conference on Peer-to-peer systems*, 2010, p. 12.

[18] D. Qiu and R. Srikant, “Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 367—-378, 2004.

[19] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, “A Performance Study of BitTorrent-Like Peer-to-Peer Systems,” *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 155–169, 2007.

[20] C. Zhang, P. Dhungel, D. Wu, and K. W. Ross, “Unraveling the BitTorrent Ecosystem,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, pp. 1164–1177, 2011.