On Unstructured Distributed Search over BitTorrent

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

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

March 3, 2014

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 small amounts of bandwidth, that are easily provided by current home broadband capabilities. We also simulate our extension to verify our model and to explore our extension's capabilities in different situations. We demonstrate that our extension can satisfy 99% of queries for popular torrents, as well as discover torrents found on as few as 10 nodes in 5 million after only 8 repeated queries to 100 random nodes.

1 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 3 describes this work. Given the current distribution of torrents, a probabilistic search is unlikely to succeed.

We say that a node is participating in a torrent if it is downloading or uploading the torrent's data. Currently, nodes in a BitTorrent network know only 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 participating in the torrent searched for. Unfortunately, finding a participating node is difficult, as Section 3 reveals. To improve the probability of a successful search, Section 4 proposes a modification to the BitTorrent protocol such that, if a node receives a query for a torrent it is not participating in, 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 substantially improves the probability of a successful search for a torrent in the network. Section 4 provides a detailed analysis.

Section 5 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 6.8KB and 8.8KB and our extension requires only 3.5Kbytes of local storage per hour.

In order to verify our model we run a series of extensive simulations in Section 6. 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.

2 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.

2.1 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)

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

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.

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 uploading or downloading 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 file. 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) [8] 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[8], and 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. By 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 4 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[9] 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.

2.2 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;

[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`

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 repartitioning. It achieves these goals at the expense of accuracy and efficiency[10, 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:

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

In the context of BitTorrent, we are performing a *known item* search, where we are searching for one and only one document, uniquely identified by its infohash. As such, broader definitions of accuracy introduced in [10, 6] are not relevant and Eqn (1) provides the probability of a successful search. In Section 3 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 [11] 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, require the same expected search length, i.e. the average number of nodes that need to be contacted in order to find documents is the same. Square-root replication, where documents are distributed over a number of nodes proportional to the square root of their popularity, performs optimally, i.e. has the lowest expected search length. In [12] 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 large numbers of node failure.

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 [13] the authors consider using random walks over an unstructured networks to replace flooding found in systems such as Gnutella. In [14] the authors introduce Brahms, a system for random peer sampling in unstructured networks that uses another gossip-based protocol. Brahms also pro-

vides security measures for sampling in a Byzantine environment.

3 The Measurement Study

In order to evaluate the applicability of PAC search to BitTorrent 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.

3.1 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.

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.

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

3.2 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 [15, 16]. 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 2 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. For a required probability of finding a document, $P(d_i) = 0.8$ when querying $z = 500$ nodes we need a document to be replicated on $r_i = 17,354$ nodes. We only observed 9 torrents with a replication at or above this level.

[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`

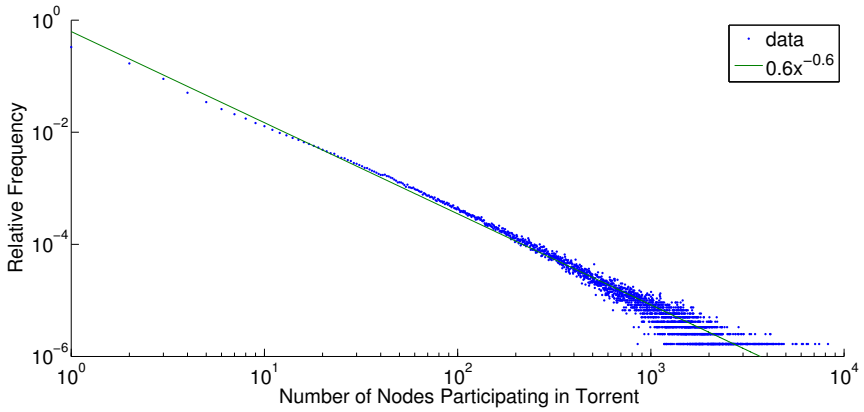

Figure 1: The relative frequency of observations of torrent participation, i.e. the number of nodes that are downloading or uploading the torrent

4 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 identify a 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, and 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 4.1 we first introduce the modification to the BitTorrent protocol that enables peers to index a random, non-disjoint subset of torrent tracking data. Section 4.2 provides a mathematical model of our extension. Section 4.3 then analyses the distribution of torrent tracking data and the associated performance of PAC search.

4.1 Indexing

Our modification is based on the following assumptions, which are discussed shortly. First, we assume that a querying node is able to sample and communicate with z random nodes in the network. This is a key assumption behind the PAC search framework. Second, we assume that a querying node will persist in communicating with nodes until the search is successful, i.e. the querying node identifies a node that is either participating in the torrent or is indexing a node participating in the torrent. Thus, when a node performs a *search* it issues one or more queries for the same 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 *request* consists of a querying node sending the desired torrent's infohash to a random node. The queried node responds with either a list of nodes it believes are participating in the torrent, or an empty list.

When a node receives a query, it updates its index, such that the querying node is now added to the re-

quested torrent's list of participating nodes. If the index does not already contain a record for the torrent, a new record is inserted with the querying node listed as a participating node. In this way, a queried node builds up a local database of tracking data.

In the next section, we analyse the expected distribution of tracking data across nodes and show that this distribution is capable of supporting PAC search.

4.2 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. The number, $r(t)$, of nodes indexing a torrent at a time t depends on: the number, u, of requests made for the torrent at t, and the proportion, c, of nodes that leave the network at t. The change in replication over time can be expressed as:

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

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:

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

The constant, k, is given by the initial condition, $r(0)$. Conceptually, $r(0)$ is the number of nodes that index the torrent before any queries have been made for it. The torrent's author can control $r(0)$ in order to enable early queries to be successful. The authoring node simply makes dummy requests to $r(0)$ nodes in order to push tracking data into the network.

4.3 Discussion

Eqn 3 gives the replication of a torrent's tracking data as a function of time. It depends on a number of constants; c, the network churn rate, u the torrent query rate, z the query size, and n the network size. We see that $r(t)$ approaches a limit of $\frac{un(1+z)}{uz+cn}$ at an exponential rate. After some small t, therefore, we can consider $r(t)$ to be stable, with negligible deviation from the limit. In this steady state condition, the replication is controlled by u, n, z and c. Both n and c are constants defined by the network and so are not controllable. It is possible to control z. In this discussion we assume a value of $z = 100$. This value could be decided globally and apply to all torrents, or perhaps a dynamic value of z could be picked by the torrent author or querying node. The effects and ramifications of when to pick z and who gets to pick it are left for future work. The number of queries performed for the torrent, u, depends on the number of nodes searching for the torrent. It is also possible for participating nodes to issue dummy queries, as the authoring node does at $t = 0$. In this way the query rate can also be controlled.

In the following discussion we assume $n = 5000000$. We set the churn rate c=0.06, i.e. 6% of nodes leave the network every hour and the same number of fresh nodes enter the network. This value is based on [17, 18], where the authors estimate that the average time it takes a node to download a torrent is 8.06 hours and the average time a node spends seeding that torrent is 8.42 hours. If nodes spend an average of 16.48 hours in the network then we expect $\frac{1}{16.48} = 6\%$ of the network to leave every hour. This does not account for nodes that download multiple torrents and so may be an over estimate. If a torrent receives $u = 100$ queries per hour then Eqn 1 tells us that $P(d_i) = 0.96$ when $z = 100$, thus any torrents that are receiving at least 100 queries per hour will have 96% of the queries performed for it succeed if queries go to 100 nodes. Any query that fails can be repeated with a different set of 100 nodes and so in practise it is unlikely that any search will fail. If we decrease $u = 50$ then $P(d_i) = 0.81$, decreasing $z = 50$ instead gives $P(d_i) = 0.57$. We see that the probability of a successful query is much more sensitive to z than u. For this reason it is important that a suitable value for z is picked. Note that even when $P(d_i) = 0.57$ the expected number of reepated

queries required before success is only 1.75. Figure 2 shows the relationship between z and u for three different values of $P(d_i) = 0.5, 0.7, 0.9$. We conclude that even if the desired probability of a successful query were high, e.g. $P(d_i) = 0.9$, that reasonable strategies for z and u can be picked. If the torrent is popular then low values of z will still provide the probability required. If the torrent is not popular, as the majority of the torrents we observed were, then high probabilities can still be achieved by either setting z higher or by artificially increasing the number of queries performed for it. For example, if we fix $z = 100$ and decide on an acceptable probability of successful query, $P(d_i) = 0.9$ then Eqn 1 tells us the replication needed to meet those requirements:

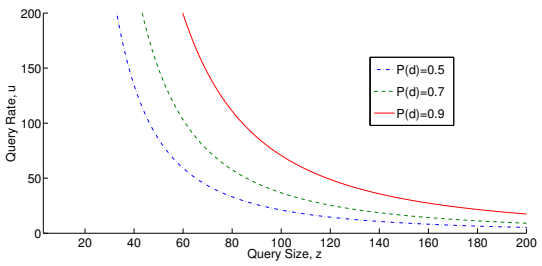

Figure 2: The query rate, u, required to meet the probability of successful query $P(d)$ for query size, z.

$$r(t)_{\text{required}} = n(1 - \exp^{\frac{\log(1-P(d_i))}{z}}) \quad (4)$$

$$= 113814 \quad (5)$$

For the same rate of churn, $c = 0.06$, and number of nodes in the network, $n = 5000000$, we see the replication of a torrent's tracking data, $r(t)$, tending towards at least this replication level when the query rate, $u \geq 69.17$. So torrents with at least 69 queries performed for it per hour will be discoverable 90% of the time if nodes query $z = 100$ nodes at a time. In our measurement study 76% of the torrents that we observed were available on fewer than 10 nodes. It seems unlikely, therefore, that the majority of torrents would have 69 queries being performed for them. But as the torrent spreads, this may become overwhelming. It might be worth emphasising (i) that dummy request only originate from participating nodes rather than nodes that index the torrent data. Also, participating nodes could decrease or stop issuing queries if a dummy query was successfully answered, an indication that the torrent was sufficiently replicated.

The above analysis assumed that the replication had reached a stable point. As noted, this stable point is reached exponentially quickly. For small t however, the replication can be very different from the steady state and therefore the probability of a successful query is also different. The value of $r(0)$ is set by the authoring node and directly controls the probability of finding the torrent in the first hour. If $r(0) > \frac{un(1+z)}{uz+cn}$ then the replication levels will be decreasing towards the limit and the probability of success will always be at or above the steady state. As seen in Eqn 4, in the steady state condition the replication is high, e.g. $r(t) = 113814$. It is unlikely that an authoring node would have the capacity or inclination to issue dummy queries to so many nodes. Instead the replication will be increasing towards the limit. $r(0)$ can therefore set a minimum probability of successful search. For example, if the desired minimum were $P(d_i) = 10\%$, $n = 5000000$, and $z = 100$ then, from Eqn 1, $r(0) = 5265$. As nodes can repeat an unsuccessful query, even if the probability of any individual query is low the expected number of queries required before success can still be reasonable. For instance, with $P(d_i) = 0.1$ we would expect nodes to have to query 10 times before success. After these 10 queries the replication will have increased by at most $10z$ and the probability of successful search will have increased to 12%. Consequently, the next node to search for the torrent is expected to have to query 8.3 nodes before success. The distributing of the bootstrap tracking data can be achieved over time and so should not constitute a significant drain on the resources of the authoring node. A more in depth analysis of the overheads introduced by this extension to BitTorrent follows.

5 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 to other nodes. This requirement increases bandwidth compared with 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 68 bytes for a handshake[10]
4. BitTorrent adds 4 bytes of length prefix per message
5. A torrent infohash is 20 bytes
6. Results can be communicated using 6 bytes per node[11]

5.1 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 applicable nodes each of the queried nodes indexes. The cost, in bytes, to send a query is $z(64 + 68) = 132z$. 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 $r(t)n^{-1}$. There are z trials (requests) made per query and therefore $zr(t)n^{-1}$ successful responses on average. The expected cost of a query is the combination of the upload cost, C_u, to send the requests, and C_d, the download cost to receive the responses:

$$C_u = 132z \tag{6}$$

$$C_d = 68z(1 - r(t)n^{-1}) + (68 + 6a)zr(t)n^{-1} \tag{7}$$

A query will cost a minimum of $132z$ bytes in upload and $68z$ bytes in download. This minimum is seen for unsuccessful queries where every response is empty. In order to estimate the cost of a successful query we need to know $r(t)$. We know that, for constant query rates, Eqn 3 tells us that $r(t)$ approaches an asymptote, and, assuming it increases towards this limit, we can derive an upper bound for the download cost:

$$\max C_d = z(68 + \frac{6au(1 + z)}{cn + uz}) \tag{8}$$

Using an example from Section 4.3, if $z = 100$, $u = 100$, $n = 5000000$ and $c = 0.06$ then we see that the upper bound of the cost to download responses to a query is 6.8Kbytes when $a = 1$ and 8.8Kbytes when $a = 100$. These costs are acceptable. In fact the size of an average webpage (on 2013/04/15), estimated to be 1.411MB[12], far exceeds this.

[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.

[12] `http://httparchive.org/interesting.php`

5.2 Authoring Node Bootstrapping

Given a constant query rate, u, we know that the replication of a torrent's tracking data approaches a limit at an exponential rate. After a sufficient amount of time the amount of replication in the network will be relatively stable; the new nodes querying for the torrent will increase the replication removed by the nodes that are leaving the network. The replication can either decrease towards this limit or increase, depending on the number of dummy queries performed by the authoring node at time $t = 0$. Since the number of replicas in the steady state condition is likely to be too high for a single node to generate we assume the replication to be always increasing towards the steady state limit. The $r(0)$ bootstrap replication then provides a minimum replication level, and so the authoring node can set a minimum probability for successful search. In previous examples we have used $z = 100$ and $n = 5000000$, if we use these values in Eqn 1 we can calculate the required replication for a desired probability of successful query. A minimum probability of $P(d) = 0.1$ requires $r(0) = 5265$, $P(d) = 0.5$ requires $r(0) = 34538$. The latter baseline would cost the author $34538*132$ bytes, or 4.5Mbytes. For comparison purposes we ntoe that the default minimum number of packets sent in a ping flood is 100 per second, at this rate the baseline could be reached in just under 6 minutes at an upload bandwidth requirement of 0.1Mbit/s. At this minimum probability a querying node is only expected to have to perform 10 queries before sucess.

5.3 Single Node Responding to Multiple Queries

In order for searches to be successful, queried nodes must respond to requests. 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. In [15] the authors observe that the average BitTorrent node will perform $q = 1.33$ searches every hour. We can estimate the number of requests generated by the entire network every hour as qzn.These requests will be sent to nodes uniformly at random, each individual node receiving an expected $\frac{1}{n\text{th}}$ of the total. If we use, as in previous examples, $z = 100$, then we would expect each node to receive 133 requests per hour. The cost of responding to these queries depends on the size of the response that can be sent. The longer a node remains in the network the more likely it is to index a requested torrent. If, for simplicity, we assume that for every request received $a = 1, 10$, or 100 results can be returned, we have that the cost of responding to requests is $133(68 + 6a)$ bytes per hour. The cost of receiving requests is $132 * 133$ bytes per hour. We estimate that our extension therefore requires 39 bits/sec in download bandwidth and between 22 and 198 bits/sec in upload bandwidth, amounts easily provided, given current home broadband capabilities.

5.4 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 aware of. If we assume a worst case scenario where 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 requesting 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. As above, we estimate that nodes will receive, on average, 133 requests per hour. At this rate, a node's local index will reach 1GB after 289,184 hours, or 33 years of continuous use.

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, i.e. data that suggests that a node owns a torrent when it does not. We briefly consider three strategies for removing incorrect data. Using a Least Recently Used algorithm, nodes could remove old index data to make room for new. Given the churn rate of BitTorrent networks, newer data is more likely to be correct. One of the current BitTorrent DHT implementations pe-

riodically sends a ping to indexed nodes. If the ping times out then a negative mark is given that node, too many negative marks and the node is removed. A simpler method would be to remove data from the index as soon as it reaches a certain age. 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.

6 The Simulations

In this section we discuss the simulations we ran in order to verify our extension's ability to enable PAC search over BitTorrent. In the simulations we model complex node behaviour that is not accounted for in our models and analysis so far. Each of our simulations creates a BitTorrent network of 5 million nodes, each node operating independently of each other. We simulate a single torrent in the network and analyse the queries, replication and participation over time. Each simulation is comprised of multiple trials, trials are repeated until the minimum number have been run or the confidence level of the recorded statistics is 5%, which ever is greater. For each simulation we apply a PAC search using a query size of $z = 100$. Until now we have assumed that network churn removes a constant proportion of nodes from the network every hour. In [19, 17] a fluid model of BitTorrent networks is created that instead describes churn as a function of each individual node's time spent in the network, a combination of time spent downloading the torrent and time spent seeding it. In this model the amount of time a node is willing to seed a torrent for is assumed to be exponentially distributed with mean $\frac{1}{\gamma}$. In order to implement this, we have each node sample from an exponential distribution with $\gamma = 0.01\dot{6}$ in order to determine how long to seed the torrent for. The amount of time spent downloading the torrent is more complicated; the maximum amount of time a node is willing to wait for a download to complete is exponentially distributed with mean $\frac{1}{\theta}$. The actual time spent downloading is determined by the node's available download bandwidth and the amount of upload bandwidth available from participating nodes. We implement this by assuming that each node's bandwidth allows for a maximum of 10% of the torrent file to be downloaded every hour and 1% of the torrent file to be uploaded. We then have each node sample from an exponential distribution with mean $\theta = 0.025$ in order to determine how long they are willing to wait. If there are enough participating nodes that a node can complete a download before aborting then the node seeds the torrent for the randomly sampled time period described above. If a node aborts a download or finishes seeding, then the node leaves the network. In order to keep network size a constant 5 million, when a node leaves, we add a fresh node to the network to compensate. This more complex model of churn allows us to verify our extension under a more realistic setting. We model and observe a single torrent in the network for a maximum of 450 hours. Not every node participates in the observed torrent. Nodes that do not participate still exhibit the behaviours outlined above but they make no PAC queries. Instead they simply progress through their downloading and seeding phases, responding to any PAC queries made of them, before leaving the network.

Using this model for churn we ran simulations of torrents with constant numbers of participating nodes, i.e. whenever a participating node left the network it was immediately replaced by another, new, participating node. We ran three such simulations: one for a torrent with a constant 1000 node participation; one with 100; and one with 10. These numbers cover the range of constant participation observed by the authors in [20], who noted that a significant number of torrents over 40 weeks old displayed a constant level of participation. With constant participation the amount of time it took each node to download the torrent was almost equal across all nodes. This meant that we observed a close to constant query rate. Figure 3 shows the probability of a successful query over time for the three simulations. As expected from our previous analysis, with a constant query rate we observe a steady probability. When the participation is set to 10 nodes we observe an average probabil-

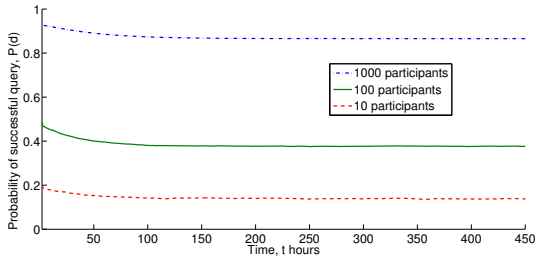

Figure 3: The probability, $P(d_i)$, of a successful query over time for three different levels of constant torrent participation.

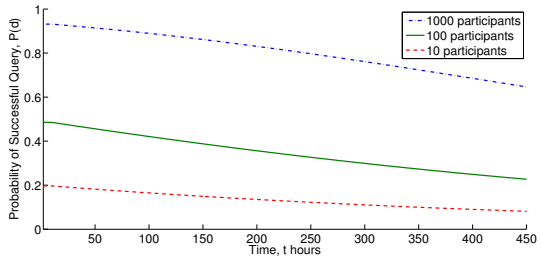

Figure 4: The probability, $P(d_i)$, of a successful query over time for three different levels of fixed torrent participation.

ity of a successful query of 14.26%, this might seem low, but each joining node is running an average of only 8 queries before success. This is in-line with our previous analysis and shows that even if only 10 out of 5 million nodes own a torrent our extension enables that torrent's discovery. For greater levels of participation the probability of successful query is higher and therefore the torrent is even easier to find. When 1000 nodes participate, the average probability of finding the torrent after a single query to 100 nodes is 87%.

Our next set of simulations explore what happens if a fixed number of nodes participate in a torrent but no additional queries are made for it. In this situation we define a fixed number of nodes that remain in the seeding state for the duration of the simulation. No nodes search for the torrent and so the replication decreases with network churn. Figure 4 shows the probability of a successful query for torrents with fixed participation levels of 1000, 100, and 10 nodes. We observe that the probability declines at an almost linear rate, dropping between 11.55 and 28.51 percentage points over the 450 hours of the simulation. We conclude that if a torrent's participating nodes remain in the network then our extension is reasonably resilient to churn before any steps have been made to mitigate it. This is because each participating node will have replicated tracking data that will never become incorrect. If these nodes wished to stabilise the probability of discovering their torrents then they could make dummy requests to mitigate the loss of replication due to network churn. For participation levels of 1000, 100, and 10 on average as few as 181, 46, and 15 dummy requests per hour would have to be made respectively.

Our next simulations use the fluid model from [19, 17] to model not only the network churn but also query rate over time. Our analysis so far has assumed a query rate at or near a constant value but we cannot expect that torrents will always exhibit such constant popularity. In the fluid model, query rate is determined using the node arrival time; the amount of time that passes before a node first starts to participate in the torrent. Node arrival time is exponentially distributed with mean $\frac{1}{\lambda}$. In our simulations a number of nodes are chosen to participate in the torrent, each node samples form an exponential distribution with mean $\lambda = 0.0\dot{3}$. This tells the node how many hours to wait before initiating a PAC search. We simulate three types of torrent; torrents whose participation peaks at 10,000 simultaneous nodes; torrents with a peak at 1000 nodes; and torrents with peaks at 100 nodes. These numbers broadly cover the range of participation we observed in our measurement study. We omit torrents with a participation peak of one node because there is no query rate to simulate over time. Such a torrent, and any torrents with similarly small participation, will *h*ave to have nodes issues dummy requests in order to make the torrent discoverable, as discussed in Section 4. Figure 5 shows the probability of a successful query over time during these three simulations. As

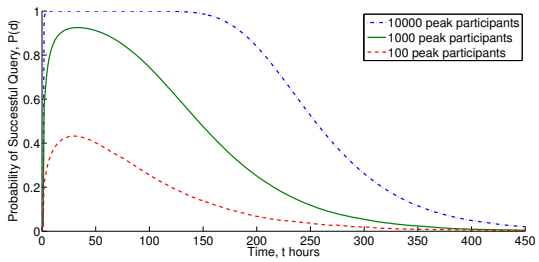

Figure 5: The probability, $P(d_i)$, of a successful query over time for three different levels of peak participation under fluid modelled query rates.

expected, if a torrent has a higher participation then it's tracking data will have higher replication and so the probability of success will be greater. Interestingly, under this model of query rate the probability of successful query peaks with the participation and so is greatest when the most queries are being performed. For this reason the average probability of a successful query is 99.06% for torrents that peak at 10,000 nodes, 78.61% for torrents that peak at 1000, and 25.68% for torrents that peak at 100. This is despite seemingly long tail of lower probabilities for each curve. The average number of queries required before success was 6 for the worst performing curve, meaning that even if a node is searching for a torrent whose participation never exceeds 100 nodes in 5 million, only 6 queries are expected to be required before success.

7 Conclusions and Future Work

The security weaknesses of the BitTorrent protocol are well known. Improvements to the protocol have been made to alleviate this issue. However, even the DHT-based extensions have proven susceptible to attack. Since unstructured networks are usually more resistant to attack, this paper investigated the feasibility of a probabilistic (PAC) search to discover torrents.

The performance of PAC search is strongly dependent of the number of nodes queried and the distribution of torrents in the network. A two month study of the distribution of torrents across nodes showed a power law distribution that is not amenable to PAC search. To address this issue we proposed a modification of the BitTorrent protocol such that each node in the network now indexes a random subset of tracking data. Each node's local database is independently constructed by recording the torrent ID, i.e. infohash, together with the IP address of the querying node. A subsequent analysis of the distribution of tracking data revealed that the tracking data is replicated sufficiently to support a PAC search. Moreover, the communication and storage overheads associated with the modified protocol were shown to be small. Thus, no degradation in performance of BitTorrent is expected.

Simulations were performed on a network of 5 million nodes under a variety of torrent query rates and churn rates. The simulation results support our theoretical analysis.

We envision that PAC search could be used to complement rather than replace existing torrent discovery mechanisms. There are a number of directions for future work. These include (i) developing a mechanism to adaptively select the number of nodes queried based on the popularity of the queried torrent, and (ii) developing a mechanism for nodes participating in a torrent to adaptively issue dummy queries so that a torrent's tracking data is sufficiently replicated to guarantee that the probability of a successful search is high.

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 Inter-*

national 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] 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.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] M. Izal, G. Urvoy-Keller, and E. Biersack, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime," in *Passive and Active Network Measurement*, 2004, pp. 1–11.

[19] 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.

[20] 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.