Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT

Liang Wang and Jussi Kangasharju
Department of Computer Science
University of Helsinki, Finland

***Abstract*—Peer-to-peer networks have been quite thoroughly measured over the past years, however it is interesting to note that the BitTorrent Mainline DHT has received very little attention even though it is by far the largest of currently active overlay systems, as our results show. As Mainline DHT differs from other systems, existing measurement methodologies are not appropriate for studying it. In this paper we present an efficient methodology for estimating the number of active users in the network. We have identified an omission in previous methodologies used to measure the size of the network and our methodology corrects this. Our method is based on modeling crawling inaccuracies as a Bernoulli process. It guarantees a very accurate estimation and is able to provide the estimate in about 5 seconds. Through experiments in controlled situations, we demonstrate the accuracy of our method and show the causes of the inaccuracies in previous work, by reproducing the incorrect results. Besides accurate network size estimates, our methodology can be used to detect network anomalies, in particular Sybil attacks in the network. We also report on the results from our measurements which have been going on for almost 2.5 years and are the first long-term study of Mainline DHT.**

I. Introduction

There have been many measurements done on peer-to-peer networks in general and BitTorrent in particular (see Section VI for a detailed comparison) over the past years. However, most of the recent studies, e.g., [1]–[3], have focused on smaller networks like KAD and Vuze and we are aware of only two other studies on Mainline DHT [4], [5], but as we discuss in Section VI, their estimation methods omit a crucial parameter and thus yield highly inaccurate results.

Because of this, we developed a more accurate method for measuring the number of nodes in Mainline DHT. Although we use Mainline DHT as our test case, our methodology equally applies to any measurement of a large-scale system based on sampling a part of the system and scaling up to obtain the number of nodes in the system. Existing studies incorrectly assumed that combining the results of a small number of samples will yield a good estimate of the network size. Our work shows that although the estimate from combined samples is better than an estimate from a single sample, errors on the order of tens of percents in the size of the network still persist. Our methodology fixes this error and yields much more accurate estimates.

In order to demonstrate the higher accuracy of our method, we perform extensive validation experiments in a controlled environment and show that previously presented methods yield incorrect results. Through an iterative tweaking of previous methods, we show what the causes of the errors are and that by fixing those issues, correct results can be obtained.

Additionally, we obtain a comprehensive picture of users of Mainline DHT, which is the largest DHT network in the Internet. Although our focus is not on presenting the actual measurement data from Mainline DHT, we will briefly present the main findings regarding the number of nodes and the churn patterns, since previous reports on these have been inaccurate.

The contributions of this paper are as follows:

1) We identify a systematic error in previous works measuring the number of nodes in DHT-based BitTorrent networks (e.g., Mainline DHT, KAD, Vuze) and present the cause behind it.
2) We develop an efficient and accurate methodology called *Correction Factor* for measuring the size of Mainline DHT. Our methodology gives an accurate estimate of the system size in less than 5 seconds. The methodology is based on modeling the inaccuracies of the crawling as a Bernoulli process.
3) We validate our methodology and justify our claims about the inaccuracies of previous works by performing extensive comparison and validation in a controlled environment, confirming our claims.
4) Applying the methodology to Mainline DHT over a period of more than 2 years, we discover that the number of users varies between 15 and 27 million in a day, with a clear and pronounced daily churn pattern. There was an increase of about 10% in the number of users from 2011 to 2012, but since then the number of users has remained stable.

Many of our findings are very different from most of the results previously presented about BitTorrent-like systems. As we later show, these differences are mainly due to two factors. First, previous studies either used different methodologies or studied different systems; these will naturally produce different results. Second, some of the previous studies have used inaccurate methods leading to incorrect results. Our methodology fixes the inaccuracies and therefore gives more accurate results.

We would like to point out that our methodology aims at estimating the size of the network, although as we show in Section V it has other applications as well. The inaccuracies in previous work relate only to their estimates of the size of the network; other parameters such as session lengths, shared content, etc., are unaffected by the omission in their measurement methodology.

978-1-4799-0521-8/13/$31.00

This paper is structured as follows. Section II discusses the merits of different measurement methodologies. In Section III, we show our crawler design and give details on our measurement methodology. In Section IV, we show how we set up our monitor system, and discuss how crawler performance affects results. In Section V we show how our measurement methodology can also be used to discover Sybil-attacks in the system. We discuss related work in Section VI, and conclude our paper in Section VII.

II. Systems and Measurements

Measuring peer-to-peer (P2P) networks and in particular BitTorrent has been very popular in the networking community over the last decade. Measurement methods can be divided into different categories either based on the system or methodology used. In this section, we first present an overview of Mainline DHT and compare it with other DHT-based systems. We then give an overview of measurement methodologies and discuss their pros and cons.

A. Mainline DHT

Our focus in measuring Mainline DHT (MLDHT) is on *obtaining a system level view* of the network. MLDHT is Kademlia-based protocol, which denotes the distance between two nodes as the XOR of their IDs. Node ID in MLDHT is 160-bit long and is not persistent, i.e., every time a node joins the system, it will generate a random ID on the fly. Thus, it is impossible to measure some metrics such as inter-session time since it is not possible to correlate users across sessions.

MLDHT is the largest P2P system today with 15–27 million concurrent users online. Because of its popularity in the real-world, many modern P2P softwares support the MLDHT protocol and it has in fact evolved into an ecosystem [6]. This evolution means that it is not restricted to any particular type of content or application, and therefore obtaining a system level view is paramount to a better understanding of this ecosystem.

There is another popular DHT implementation in the BitTorrent world, namely Vuze DHT [7]. Even though both are based on Kademlia [8], they are fundamentally incompatible as protocols. Vuze has been estimated to have about 1 million users [9], [10], but as we show, MLDHT has 10 to 20 times as many users, making it a more important network.

B. Methodologies

We classify existing BitTorrent measurement methodologies in two high-level categories: tracker- and DHT-based. These can be further refined into sub-categories as described below. Table I shows an overview of the sub-categories and respective advantages and disadvantages. In this section, we focus on the differences in methodologies and return to contrasting our results with related work more closely in Section VI.

Tracker-based

Tracker-based measurements can be divided into three sub-categories:

- Instrumenting a client
- Monitoring a swarm
- Using tracker logs

Research with instrumented clients, e.g., [6], [11]–[13], allows collecting of data directly from the users and permits looking at system performance as users would perceive it. Because users join swarms and data can only be collected based on what the client sees, instrumented clients are actually also swarm-based measurements. A major issue in using instrumented clients is the risk of obtaining a biased measurement, since only data from users who have specifically installed the instrumented client is obtained. Existing studies typically do not address this issue of possible bias.

Swarm-based measurement in general focuses on a single swarm or a set of swarms and monitors the behavior of peers in that swarm. Monitoring can happen either with instrumented clients (who need to be a part of the swarm) or by joining the swarm and logging all the information that the measurement client sees. Swarm-based measurement is appropriate when investigating that particular swarm (or similar swarms), but is inappropriate for investigating the complete system. For example, client behavior in a swarm for a popular movie is going to be very different from clients in an unpopular swarm for an electronic book. Measuring session lengths for the whole system is also impossible with swarm-based measurements.

In both swarm-based measurement and instrumented client measurements, there is also the risk that the measurement is biased by the measurement clients. This is because BitTorrent clients cluster based on their upload bandwidths [14], [15] and the measurement client might not get even a good picture of the swarm due to its limited upload bandwidth.

Besides swarm-based research, there are also some work based on analyzing tracker logs [6], [16]–[18]. Tracker-based measurement gives a broader view than swarm-based measurement, since a tracker typically hosts many swarms. However, even popular trackers have a biased view of the system as a whole. For example, Chinese users represent a large fraction of BitTorrent users, but mainly use trackers inside China which are not covered by popular international trackers. As Zhang et al. [18] show, there are a lot of private trackers in use, making it impossible to obtain a good system-level view from tracker-based analysis.

DHT-based

There are three popular DHT-based systems, KAD, Vuze, and MLDHT. They are all based on the Kademlia DHT, however MLDHT is different from the other two, requiring a slightly adapted measurement methodology. Measurements of KAD and Vuze have been very popular recently [1]–[3], [9], [10], [17], [19]–[21]. The main difference between the systems is that KAD has permanent IDs and Vuze assigns IDs based on IP address to clients. In MLDHT a client chooses a new ID for every session.

Steiner et al. [2], [19] state that they were able to crawl an 8-bit zone in KAD network within 2.5 seconds, and a full crawl in 8 minutes. This is possible because of the persistent IDs which means that any IDs collected during previous crawls can be reused in subsequent crawls. In MLDHT with dynamic IDs, this is not possible and thus MLDHT requires more time to crawl a similar zone.

Method	Advantages	Disadvantages	Related
Instrumented client	Direct access to user behavior	Data comes only from instrumented clients, not all clients Biased by client selection	[6], [11]–[13]
Swarm monitoring	Focuses on content	Results only applicable to that swarm Biased by swarm selection	[16], [17]
Tracker logs	Focuses on content Better coverage than single swarms	Biased by tracker selection	[6], [16]–[18]
Persistent DHT	Fast comprehensive crawls possible System-level view	Studying session times difficult Content monitoring difficult	[1]–[3], [9], [10], [17], [19]–[21]
Dynamic DHT	Fast crawls possible System-level view	Content monitoring difficult	[4]

TABLE I: Classification of measurement methodologies for BitTorrent-like systems

In general, the methods for persistent and dynamic DHT IDs are similar, i.e., crawling a specific zone or injecting sybils into the system. General rules of thumb dictate that the duration of a crawl should be as short as possible to obtain a good snapshot and duplicates should be filtered out from the results. However, previous research work on MLDHT, KAD, and Vuze all contain a methodological omission. They fail to take into account the "missing node" problem (see Section III-D) which means that their results are incorrect. By failing to account for the nodes missed in a crawl, scaling up the node density from the observed crawl will lead to an incorrect estimate of the overall system size. As our results show, for MLDHT this error would be around several tens of percents.

C. Background on MLDHT

We now present a short overview of how MLDHT operates, as this is fundamental to the design of our measurement system. In the BitTorrent system, to join a swarm, a peer needs to get meta information first. In standard BitTorrent, the meta information can be obtained from the torrent file, which also contains a list of centralized trackers to help a peer get the initial peer set to bootstrap the download.

Partly due to legal issues, but also based on improving the service availability and system robustness, distributed trackers have been developed. BitTorrent has two independent, incompatible distributed tracker implementations, even though both are based on the Kademlia DHT [8]. One is VUZE [7] and the other is MLDHT.

MLDHT implements the minimum functionality of Kademlia. In MLDHT, both peers and content each have a 160-bit string as its ID. Content IDs are also known as infohashes. A peer uses this infohash to obtain the meta information and initial peer set. MLDHT supports four control messages:

1) `PING`: probe a node's availability. If the node fails to respond for some period of time, it will be purged out of the routing table.
2) `FIND_NODE`: given a target ID, this message is used to find the k closest neighbors of the ID.
3) `GET_PEERS`: given an infohash, get the initial peer set.
4) `ANNOUNCE_PEER`: a peer announces it belongs to a swarm.

Figure 1 illustrates normal operation in MLDHT. Suppose we have 3 nodes A, B and C. A holds a file with infohash $x = 59$. Assume B is responsible for storing x its peer set. Node C wants to download the file.

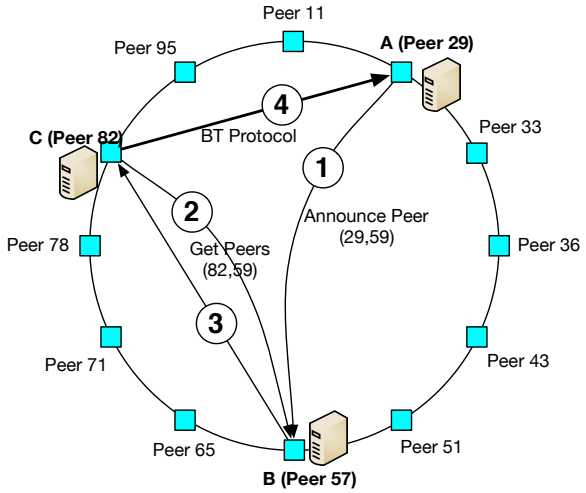


Fig. 1: Normal operation of MLDHT

First, A publishes the file by storing x at B. A calls `GET_PEERS` iteratively to get closer and closer to B, and finally reaches it. Then, A uses `ANNOUNCE_PEER` to tell B he is sharing a file with infohash x. B stores A's contact information in the corresponding peer set for x. Since A is the publisher, it is the only one in the peer set at the moment.

When A sends the `GET_PEERS` messages, two possibilities emerge. If the queried node knows this infohash already and stores some peers in the corresponding peer set, it will respond with the peer set. If it does not know the infohash, it will respond with the k closest nodes to the infohash in *its* routing table. In such a way, A will get closer and closer to B, and finally reach B and the search finishes. `FIND_NODE`, which we use in our work, behaves the same way, except that x in this case represents a node instead of content.

For C to download the file, it should get x first. It will proceed exactly the same as A did before by using `GET_PEERS` to approach B. Since B already saved the peer set for x, C can obtain the initial peer set from B. C joins the swarm, sets up connections to the peers in peer set, and gets metadata (torrent-file) from other peers using BitTorrent extension protocols [22], [23]. Then the download process starts. In our work, we do not consider the download process.

III. Methodology

The known crawling methods are based on starting from an n-bit zone, which means that all the nodes in that zone share a prefix of n bits in their IDs. We have decided to scale up the node density from a suitably selected zone and use that result as an estimate of the network size. We will first focus on evaluating different zone sizes and then derive a bound for the error in our method. The difference between our work and the previous ones is we have well-defined methodology to validate our experiment design and the results.

A. Assumptions

This kind of measurement method is based on the assumption that nodes are uniformly distributed in the ID space. In principle, the MLDHT protocol should guarantee a uniform distribution of node's ID. However, we cannot take this for granted since not all the implementations conform to the protocol, and the abuse of certain IDs has already been observed in other P2P networks as Steiner et al. reported in [2].

We carefully examined a large set of samples (over 32000) crawled from the different parts of MLDHT, and found the node IDs follow a uniform distribution. We did observe some abused IDs such as ID 0, but they only contribute a trivial amount of nodes to the whole MLDHT, and can be safely neglected in our case.

B. Choosing a Zone

We pick a random ID as our node ID, then we try to collect all the nearby nodes by exploiting the `FIND_NODE` operation. To bootstrap this process, we maintain a set of active nodes of several thousands, and randomly choose 100 to put them in a FIFO queue. Then we perform a BFS iteratively to get enough nearby nodes.

Since the node IDs are assumed to be distributed uniformly in the ID space, there should be no correlation among session length, content and other factors. We have extensively sampled many different parts of the ID space and have observed the assumption about uniformly distributed IDs to hold.

Figure 2 shows the number of nodes in different n-bit zones in one sample. Recall that nodes in an n-bit zone share an n-bit prefix. We can see the curve exhibits stable behavior between 5 bit-zone and 25-bit zone.

Then we have to answer the first question: which zone we should use to scale up? For the first question, although using a large zone (small n) would seem attractive, this method is expensive in practice. The reason is as the zone size increases, the crawling time and traffic overheads increase exponentially instead of linearly, which is also revealed by Memon et al. in [3] when they were crawling KAD. At the same time, lots of nodes are missed due to the crawler's performance limit. Furthermore, because crawling a large zone takes a long time, normal system churn can affect the results during the crawl, leading to hard-to-estimate errors. In this paper, our aim is to obtain an estimate quickly and accurately with low overheads, hence a large zone is not appropriate.

However, a very small zone also has its own problems. Due to the missing node issue, which will be discussed below, it is inevitable that some nodes will be missed in each crawl. If we choose a very small zone (large n), the slight fluctuation caused by the missing nodes will be magnified after scaling up, leading to a significant (and random) difference in the estimate. The huge fluctuation can be amortized over a large set of simultaneous samples. However, this method is also expensive for a continuous monitoring system.

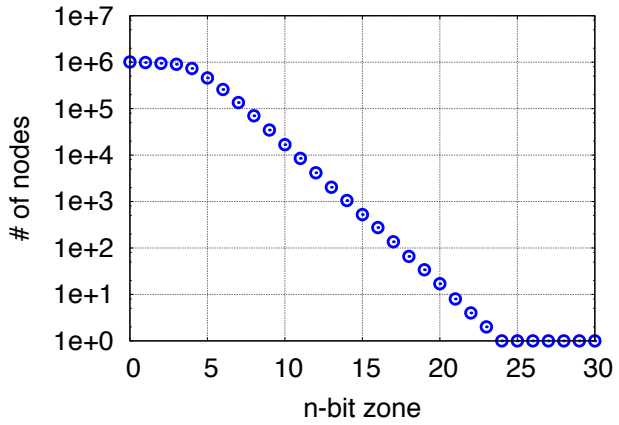


Fig. 2: Number of nodes discovered by our crawler in different n-bit zones. There were about 20 million nodes in the system when the experiment was performed. With 5-bit zone, the crawler reached its performance limit and many nodes were missed. Beyond a 24-bit zone, the node density is so sparse that the crawler cannot find any other nodes except itself.

By testing different zone sizes, we decided to use the 12-bit zone to derive the estimate, since it achieves relatively good trade-off between the overheads and accuracy. We can finish a crawl in about 5 seconds and the sample variation is small. We must point out 12-bit zone is not the only choice; any zone between 9-bit to 14-bit zones works well in our experiments. The only difference between different zone sizes is that they have different correction factors (see Section III-D).

C. Scaling Up

The second question we need to answer is: can we safely scale up the node density of a given zone to derive the network size? The answer is in fact NO, simply because a crawler cannot collect all the nodes in a zone and some of them will be missed. *This question is overlooked by most of the previous work*. Most related work simply scales up the number of nodes, but as we show in this paper, this leads to an incorrect result. It is because of this incorrect scaling that we know the previous methodologies to contain an error.

In [2], Steiner et al. set up two crawlers at different vantage points and combine the views in an attempt to get a better estimate. They mentioned that one crawler sometimes saw fewer nodes than the other, and they blamed this on network connectivity. In such cases, they just merged two node sets. However, there is a subtlety hidden here, since they did not explicitly mention whether one node set is a proper subset of the other or whether "fewer" simply means "less in number" but does not take a stand on the overlap between the sets. Although such a combination of views is legitimate, two views

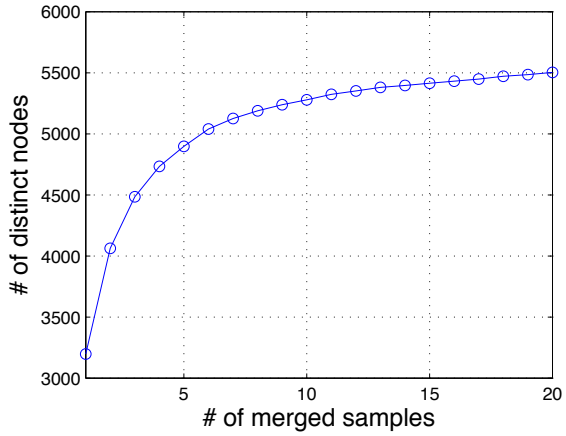


Fig. 3: Number of nodes in merged samples

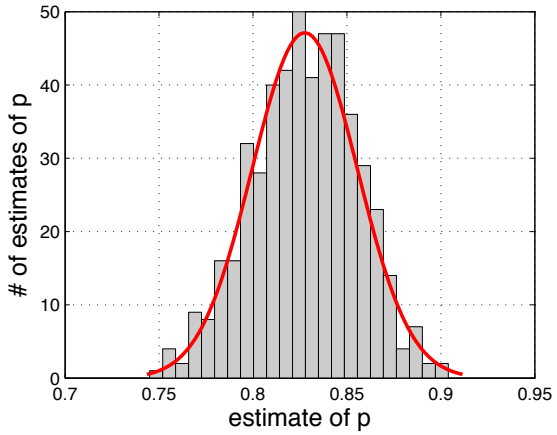


Fig. 4: The distribution of 500 estimations for p

is not enough to get a correct estimate, which is well illustrated by Figure 3.

In our study, we consider the missing node issue as a systematic measurement error, and model the whole process as a Bernoulli process. We present how we tackle this problem in Section III-D. In order to justify the need for this approach, we performed an experiment in which *20 simultaneous* samples are crawled within the same 12-bit zone. Each sample's 12-bit zone contains about 4000 nodes. Figure 3 plots how the number of *distinct* nodes increases as we merge the samples one by one. As more samples are merged, the number of distinct nodes increases logarithmically. The experiment tells us that each sample only covers a part of the 12-bit zone, thus it shows that the missing node problem exists. As the samples are collected simultaneously, we can ignore the effects of churn on the result. As an example, considering the method from [2] where they combine two samples, they would extract about 4000 distinct nodes, when we know from combining 20 samples that there are at least 5500 distinct nodes in the zone. Using these numbers and scaling them up to get an estimate of the whole network size would yield an error of 37.5%.

D. Correction Factor

The missing node issue degrades the accuracy of the estimate. The reason can be multifold, e.g., firewalls, network connectivity, congestion, different implementations, peer's abnormal behavior etc. Previous works typically attempt to prove the accuracy of their measurement by showing the small variations of samples without considering this issue. However, small variation cannot guarantee the accuracy if the measurement method is not designed to handle missing nodes.

To get around this issue, we assume that our crawler *will always miss some IDs* when it is crawling a 12-bit zone. Then we can model our sampling process as a Bernoulli process. For each ID in the zone, we have two options – "being selected" or "being missed". If the probability of "being selected" is p (then the missing rate is $1-p$), all we need for an accurate estimate of the number of nodes in the zone is an estimate of p.

To this end, we devised an experiment. First, we insert a special ID, say x, into the zone to be crawled. Second, we let the crawler keep crawling this zone repeatedly to obtain a series of samples. By counting in how many samples x appears, we can estimate the value of p.

It is obvious that the more simultaneous samples we have, the more accurate the estimation of p will be. So in practice, we insert 50 IDs into a specific 12-bit zone, and start multiple crawlers simultaneously. In such a way, the measurement efficiency can be improved significantly, because we can get 50 estimations of p in a single experiment. We ignored the influence of inserting multiple IDs, since their percentage is always well below 1% of the actual IDs in this zone. Then we generate another 50 random IDs and repeat the experiment again. In total, we carried out 10 experiments and obtained 500 estimations of p. We ran Jarque–Bera[1] test on our estimations, the *null* hypothesis was accepted with a significance value $\alpha = 0.05$. Figure 4 shows the distribution of these estimations with normal distribution fit.

In our experiments, the average of p is 0.8276, sample deviation is 0.0280, so the 95% confidence interval is $[0.7716, 0.8836]$. In other words, let M be the actual population size of MLDHT, and m be our estimation from 12-bit zone. The true M will fall in $[1.132 * m, 1.296 * m]$ with 95% confidence. We call this multiplier *Correction Factor (CF)* and it is given by $CF = \frac{1}{p}$.

We also investigated the *coverage* in 13–16 bit zones, for example in a 13-bit zone, the average p is 0.8343, and sample deviation is 0.0266. The results derived from different zones with the corresponding correction factor are similar.

To verify our result for p, we ran two parallel experiments. We collected several samples in a 12-bit zone and combined the samples. After dropping duplicates, the number of unique nodes increases as shown in Figure 3. Because the result converges, we can use this to estimate p. We matched this estimate with the estimate of p given by the Bernoulli process described above and found out that they match. This serves as validation of our estimate of p.

[1]Jarque–Bera test is a test of normality. The *null* hypothesis is "the data is drawn from normal distribution(skewness and excess kurtosis are both zero)".

We must point out that p is subject to many factors like network connectivity and congestion, and may change over time. Thus the value of p should be calibrated periodically in order to keep the accuracy at the required level. However, for a continuous monitoring system, performing such experiments for each crawl is extremely expensive. We will show how we tackle this problem in designing our monitor system in Section IV-A.

E. Validation of Methodology

We validated each step of our methodology from the assumptions to the final outcome. Our repeated samplings confirm that the assumption about IDs being uniformly distributed is justified; results in Section III-A demonstrate this.

In order to model the sampling as a Bernoulli process, we assumed that the crawler will always miss some nodes in a crawl. This assumption is verified by the results in Sections III-B and III-D and Figure 4. The same results also verify the accuracy of our estimate for p. Furthermore, we performed a controlled emulation experiment described below on our cluster to obtain a ground truth against which we can compare our results.

The best way to demonstrate the validity of our methodology is recreating a realistic scenario in a closed environment. To this end, we designed and performed an emulation using one million BitTorrent clients in a large-scale experiment on our department cluster. The cluster consists of 240 Dell PowerEdge M610 nodes and each node is equipped with 2 quad-core CPUs, 32GB memory, and connected to 10-Gbit network. All the nodes run Ubuntu SMP with 2.6.32 kernel.

To make the experiment setting more in line with the real world, we used three different MLDHT client implementations in our experiments: *Mainline BT*, *Aria* and *libtorrent*. Each of these is a popular client used currently on MLDHT, and because they use the same DHT, they are compatible at the protocol level. We also tested a heterogeneous situation by mixing different fractions of each clients together. In each experiment, we deployed one million MLDHT nodes on 200 machines, i.e., 5000 instances per machine. Figure 5 shows the value of p obtained by our crawler for four different traffic mixes using either 10-bit or 12-bit zones.

For each p value, experiments were repeated 50 times and arithmetic mean is used. Table II shows a more detailed view of the parameters, including p, its standard deviation, and the 95% confidence interval for the correction factor. As we show later, running a large number of parallel samples will allow to reduce the confidence intervals to arbitrarily small.

As we can see from the results, there is no significant difference in p value in different mixtures, even though a network with only Mainline BT clients always has a slightly higher value. Given the consistency of the results, we can consider that all three implementations follow faithfully the MLDHT's official specifications. A 10-bit zone's p value is always smaller than a 12-bit zone's, but corresponding *stdev* is also smaller. This result is also consistent with our previous discussion that a larger zone provides a more stable estimation but suffers from lower coverage.

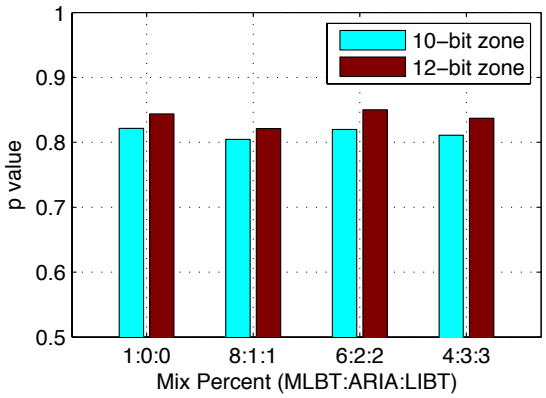


Fig. 5: 10-bit and 12-bit zone's p value as a function of different mix percents of three applications. *stdev* is small therefore omitted from the figure. 10-bit zone's *stdev* is consistently smaller than 12-bit zone's. (MLBT: Mainline BitTorrent; ARIA: Aria; LIBT: libtorrent)

We also evaluated to what extent firewalled nodes can affect accuracy of our measurement. (As discussed in more detail in Section IV-D and in [12], firewalled nodes can represent a significant fraction of the nodes in the network.) According to MLDHT protocol, firewalled nodes and stale routing information should be purged out of routing tables within 15 minutes. We are not so concerned with stale routing information, since it is easily handled by MLDHT clients. However, firewalled nodes can still enter and stay in other nodes' routing tables via different ways. Because various BitTorrent (protocol) level operations can trigger the nodes being inserted into the routing table. For example, if a node behind a firewall initiates a connection to a node outside the firewall, the second node will keep the firewalled node's information in its routing table, even though it is not able to initiate connections reversely. We simulated such a situation by mixing different fractions ($\leq$ 30%) of firewalled nodes into the system and adding some synthetic BT-level activities. The measured p value still remains at the same level, giving us confidence that our methodology does not suffer from the presence of a large fraction of firewalled nodes.

F. Implications

The above experiment has several implications regarding our measurement methodology and large-scale system measurements in general. First, the results of the controlled experiment show that our methodology, in particular the correction factor, are indeed correct and a vital component of a measurement framework. Our crawler is shown to work efficiently in realistic scenarios and to provide accurate estimates of the network size. Second, in a stable network with little churn, values of p are rather stable. This implies that the inaccuracies in measurements are due to MLDHT's inherent design properties. In practice, the MLDHT protocol cannot guarantee returning the actual k closest neighbors, which results in the need to have the correction factor. Third, the differences in p values for differently sized zones are smaller in a stable network. We observed differences of 2–4% in value of p between 10-bit and

App Percent (%)			10-bit			12-bit		
MLBT	ARIA	LIBT	p	stdev	Corr. Factor	p	stdev	Corr. Factor
100	0	0	0.8215	0.0173	(1.1681,1.2708)	0.8437	0.0263	(1.1157,1.2641)
80	10	10	0.8045	0.0164	(1.1943,1.2958)	0.8213	0.0291	(1.1370,1.3104)
60	20	20	0.8198	0.0211	(1.1601,1.2860)	0.8501	0.0243	(1.1127,1.2477)
40	30	30	0.8109	0.0190	(1.1780,1.2938)	0.8372	0.0257	(1.1254,1.2726)

TABLE II: 10-bit and 12-bit zone's p value with 95% confidence interval as a function of different mix percents of three applications. (MLBT: Mainline BitTorrent; ARIA: Aria; LIBT: libtorrent)

Simultaneous samples	**1**	**2**	**4**	**8**	**16**	**20**
Without CF	13,021,184	16,642,048	19,390,464	21,254,144	22,245,376	22,540,288
With CF	21,951,659	22,077,793	22,187,636	22,328,867	22,915,635	23,195,538
Error	40.68%	24.62%	12.61%	4.81%	2.92%	2.82%

TABLE III: Estimation with and without correction factor. The numbers report the estimated size of the network when running n simultaneous samples. (CF: Correction Factor)

12-bit zones in our controlled test, whereas we have observed differences of 6–9% in the real MLDHT. As the zone gets larger, p decreases which makes the accuracy of the estimate lower, necessitating the use of the correction factor. Scaling the node density without considering the correction factor is guaranteed to underestimate the network size, but there is no way to estimate how much lower it is. Our methodology eliminates this problem.

The correction factor is independent of the underlying crawling method and can be applied to any existing crawler. Using a very large number of simultaneous samplers in parallel obviates the need for the correction factor, but requires a considerable expenditure in resources for crawling. The correction factor strikes a trade-off, by allowing a much lower use of measurement resources and still obtaining the same level of accuracy as a large number of simultaneous samples would provide. Table III shows the network size estimated by using a different number of simultaneous parallel samplers with and without the correction factor. As we see, using only 1 or 2 samplers, as seems common in literature, will yield an error on the order of 20–40%. Naturally, as shown in Table III multiple parallel samplers improve the accuracy of our method as well, although the error between 1 and 20 parallel samplers is only on the order of 5%. However, correction factor does not give an answer for exact causes of missing node issue which can be manifold. Finding those causes will be our future work.

IV. Experiments

We now present the implementation of our crawler and discuss practical aspects related to data collection.

A. System Architecture

Figure 6 shows the four principal components of our system. An efficient *Crawler* lies at the core of the whole system. It can finish a crawl within 5 seconds, trying to gather as many nodes as possible in a target zone. Beneath the crawler, the *Maintainer* component maintains a set of over 3000 active nodes, and randomly provides 100 nodes among these long-lived nodes to bootstrap the crawler. The *Injector* component is responsible for injecting *controlled nodes* into the monitored target zone. Then the sample will be sent to

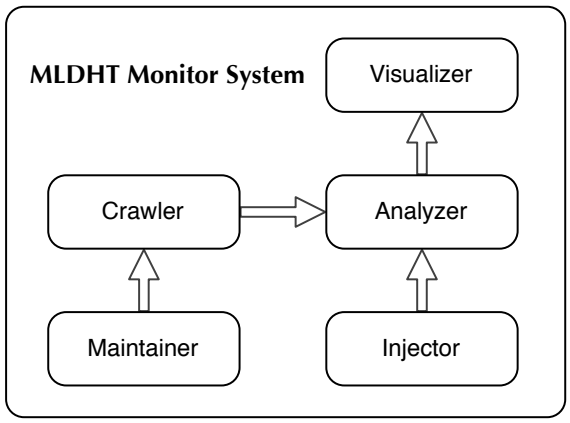


Fig. 6: Principal components of monitoring system

Analyzer component, where the p value will be calculated by checking the occurrence of controlled nodes. Finally, the estimate and other relevant information will be stored in the database and visualized by *Visualizer* component.

As mentioned before, for each crawl, obtaining several hundreds of simultaneous samples to calculate p is expensive. So in practice, p is the moving average of all available measurements and is calculated and refreshed in each crawl by putting more weight on the new value. The formula we use is $p = 0.2 * p_{old} + 0.8 * p_{new}$. It turns out p value in our system is rather stable, always around 82%.

B. Deployment

We use two nodes for sampling, both Dual Intel Xeon E5440 @ 2.83GHz with quad cores, 32 GB memory and Gigabit connection to the Internet. The operating system is Debian SMP with Linux 2.6 kernel. On each node, we set up a crawler with its own sampling policy. One is called *Fixpoint Crawler*, which always uses the same target ID when sampling. The other is called *Randompoint Crawler*, which generates a new random target ID whenever it takes a sample. The sampling frequency is twice per hour, and there is 15-minute difference between the two crawlers.

There are two reasons for setting up a pair of crawlers. The first reason is to prevent the sample gaps due to the

application failure. The second is to assess the accuracy of the captured data by cross-correlating the samples from two parallel measurements.

We have been collecting data since December 17, 2010 and the monitoring is on-going. We have had only a few small gaps in the collection process until now. The duration for capturing a snapshot varies within 5 seconds, depends on the network size at that time. On average, each sample contains about 20,000 distinct IDs from different zones. The data we have collected and the code for our crawler are available for other researchers. Please see http://www.cs.helsinki.fi/u/jakangas/MLDHT/ for more details.

C. Duplicated IDs

The crawler records ID, IP and port of a node in a crawl. Basically, we only need to handle two types of duplicates. The first is the case of same ID and IP but different port. We count such multiple records as one node since it is typically the result of a client listening on multiple ports. The other one is the case of same ID but different IP. There are two possibilities for this kind of duplicates. The first is pure collision, which is very rare and the second is due to modified clients. In both cases, we count them as one node by selecting one of the IPs randomly. We consider this acceptable, since such cases contribute a negligible part in a sample.

D. Non-responding Nodes

Non-responding nodes refer to the nodes who fail to answer our `FIND_NODE` queries. One possibility is the node is behind a firewall, another possibility is the node has already left the network and the routing information is stale. Unfortunately there is no reliable means of distinguishing between these two cases. On average, non-responding nodes constitute about 30% of our nodes, and this percentage remains quite stable in all the samples. Other studies, e.g., [12], have found similar numbers for firewalled nodes. Stale routing information should be purged in about 15 minutes on average, but this depends on the actual implementation of the client. We have seen that a large fraction of those non-responding nodes are present in several consecutive samples, leading towards the conclusion that they are behind firewalls. However, a more thorough study would be needed to validate this.

As presented in Section III-E, we tested the performance of our crawler in a controlled environment with around one third of the nodes being behind a firewall. As our results showed, our methodology is robust against firewalled nodes and we do not need to consider any special procedures for them.

E. Crawler Performance Issues

As [17], [24] pointed out before, a crawler must be well designed and carefully tuned. This is not trivial but is critical for measurement accuracy. To save time and reduce development complexity, some previous work, e.g., [4], developed their crawler from the third party library, and used it in the experiments. However, those libraries are usually intended for general use as parts of normal clients and are not specialized for measurements.

In order to show that general purpose libraries are not suitable for system measurement, we developed a crawler based on the *libtorrent* library. *libtorrent* is a popular open source library for BitTorrent protocol. Several popular BitTorrent clients (e.g. Deluge, LimeWire, rTorrent) are developed with this library, and it is also used in some research work.

In the first version, we only made marginal modifications to *libtorrent* and kept most of the default settings. We also avoided unnecessary tweaks. The crawler suffered from low efficiency and high inaccuracy. The estimated network size reported by the crawler is only 1/6 of the actual number obtained by our special-purpose crawler. Furthermore, the difference in the estimate of the network size between [4] and our work is about a factor of 6, leading us to conjecture that the difference in the results is largely due to the implementation of the crawler. In the second and third version, we tuned some parameters and also tweaked the code in *libtorrent* to improve the crawling efficiency. Even though the tuned version can crawl faster and discover more nodes within a zone, the reported value is still only 1/4 to 1/3 of the actual value, which is far from accurate.

We therefore looked into *libtorrent* code and carefully examined its design. The reason for inefficient performance is the results of many factors; some commonly used mechanisms like banning suspicious nodes, abandoning malformed messages can severely degrade crawler's performance and accuracy. Queue size, bucket size, complicated routing table operations, and callback functions further make the crawler's efficiency deteriorate rapidly. For a crawler, the most used method is obtaining a node set from a specific zone, so speed of the crawler is the most important factor considering the convergence speed. In a distributed system with high churn, crawling speed effectively determines the quality of a *snapshot*.

We then further modified our *libtorrent* test case and added a correction module into the libtorrent crawler. This correction module consisted of the *Injector* and *Analyzer* components from our own crawler. Based on these, we were able to determine that the libtorrent crawler can discover only about 32% of the actual nodes within the zone. After we applied our correction factor (Section III-D), the results were consistent with our own MLDHT crawler. This test confirms that a correct and efficient implementation of a crawler is vital to getting accurate measurements. A slow crawler, like used in [4], will yield inaccurate results.

F. MLDHT Evolution

Figure 7 shows the number of nodes in MLDHT during one week in March 2011, March 2012, and April 2013. The number of users shown, 24 to 27 million at peak, is typical and the roughly 10 million users of daily churn is also typical. The churn is mainly generated by European, in particular East European users; details are not shown due to space constraints. The weeks shown are typical for the years they depict and the roughly 10% growth from 2011 to 2012 happened mostly gradually, although there was a marked increase in Fall of 2011. From 2012 to 2013, the size of the network has remained roughly stable.

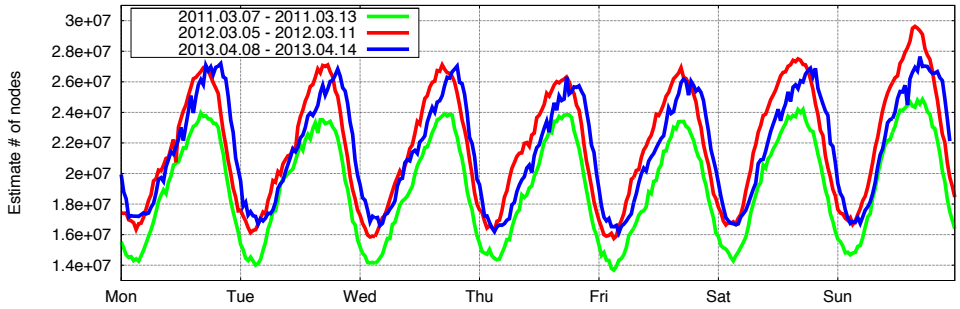
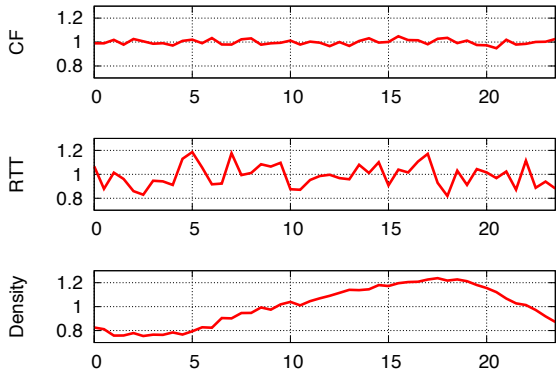
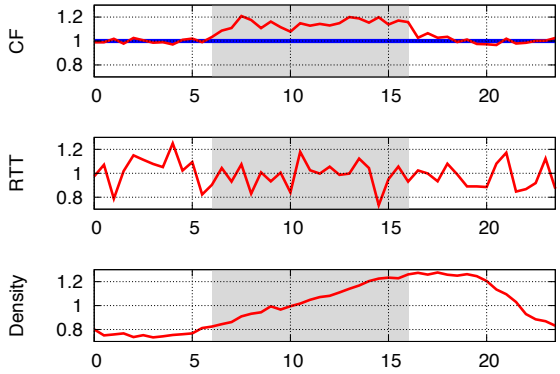


Fig. 7: One week 2011, 2012, and 2013. The weeks show the typical daily churn, mainly caused by East European users. The typical number of users increased by about 10% from 2011 to 2012, but has remained stable since then.



(a) In a normal day



(b) During a Sybil-Attack

Fig. 8: Evolution of three system metrics in a day, from top Correction Factor, RTT, and node density. Values have been normalized to the average value over that day.

V. Correction Factor and Anomaly Detection

Despite the large churn in the system, the correction factor is a rather stable system value. Therefore it can also be used as system health indicator and detect system anomalies. Figure 8a shows the evolution of three system metrics: correction factor, average RTT (Round-Trip Time), and node density over 24 hours. For the ease of comparison, the values have been normalized to the average value of that day. From the figure, we can easily see that correction factor is the only metric which remains stable throughout the whole day.

However, system anomalies, especially Sybil-attacks, can significantly influence the correction factor. In our previous work [27], we reported a real-world Sybil-attack in MLDHT on Jan. 6, 2011. The attack was from two virtual machines of Amazon EC2, and started from 6:00 am. Figure 8 presents evolution of three metrics captured by our monitor system during the attack. From Figure 8b, we can see the correction factor increased by about 20% when the attack was launched and it recovered to the normal level after the attack ceased. At the same time, there is no noticeable changes in the other two metrics. For detailed analysis of these attacks in MLDHT, please refer to [27].

The reason for this increase is that in a Sybil-attack like this, the attacker inserts a large number of sybils in the network (or part of it). Our sampling process is affected by this and the probability p that a node is present in a sample decreases (see Section III-D). Correspondingly, the correction factor increases. This is expected, since the attack increases the number of nodes in the network and the correction factor captures and corrects inaccuracies in the sampling process; the increase is necessary to obtain the correct estimate of the size of the network.

VI. Related Work

There have been a lot of measurement work on different P2P networks, such as [1]–[3], [9], [10], [17], [19]–[21], [24], [27], but most of them studied KAD or Vuze DHT, and only [4], [5], [27] are MLDHT related. In [27], Wang et al. studied two major Sybil-attacks in MLDHT and reported large-scale anomalies in the real-world system by their honeypot design.

In [25], Kostoulas et al. gave a thorough and general survey on various techniques for group size estimation of large-scale distributed systems. In [3], Memon et al. monitored 32000 peers in KAD using a single PC. They intercepted most of the traffic to the monitored nodes. Their crawler *Montra* is introduced in [24] where they also extensively discuss practical issues related to crawling P2P networks. In [17], Stutzbach et al. also point out several pitfalls that can cause significant bias in the sample.

In [1], [2], [19], Steiner et al. used crawler *Blizzard* to study KAD. Their work showed China is the biggest country in KAD, and they also showed the popularity of KAD in Europe. Our findings mostly concur with these results. In [10], Steiner et al. crawled Azureus DHT by exploiting `REQUEST_STATS`

and REPLY_STATS messages in Azureus DHT protocol. They found out that there are 3–4.5 million users in KAD, followed by Azureus with about 1 million users. According to our research, there are over 27 million users in MLDHT (at peak time). We adapted our sampling method from their work to be able to handle the much larger MLDHT network. Their original sampling method suffers from the missing node issue (Section III-D).

[11], [21] focused on content and publishing activities on popular P2P networks. Zhang et al. carried out a thorough study on BitTorrent ecosystem in [6]. Their method is monitoring large amount of swarms from popular BitTorrent portals. As mentioned in [6], this can cause bias in the measurement since some countries, e.g., China and India, are underrepresented. In [26], Iosup et al. tried to produce an accurate geo-snapshot for BitTorrent network by using trace files from Supernova.org. Their work, besides being rather old, suffers from the same limitations as [6] since it ignores all users not using those particular sites.

In [4], Junemann et al. did very similar work as ours. However, there is big difference in the estimate of the network size. Their estimate is about 5 to 7 million, whereas ours can be over 27 million. We suspect the reason is that they adopted a third-party plugin, *libtorrent*, in their implementation for accessing the DHT. Since *libtorrent* is not designed for sampling the network, it ignores convergence speed, missing nodes, and other aspects. As a result, the node density is severely underestimated. Our method remedies this problem and in addition also allows us to tell the exact estimation error. A similar problem also exists in [5] where they estimate MLDHT size based on a modified plugin from Vuze. The key reason behind the difference in the results is that both of these works ignore the missing node issue. As we showed in Section IV-E, crawler performance is one of the key factors in getting a right estimate and plain *libtorrent* is not sufficient for actual measurement work.

VII. Conclusion

In this paper, we have developed a fast and accurate method for estimating the number of nodes in the BitTorrent Mainline DHT network. We have identified the missing node problem as a key omission in previous work and show how to fix this via modeling the crawling as a Bernoulli process. Our method provides much more accurate results and is able to run in about 5 seconds. Our correction factor can also be used to identify Sybil-attacks in the system.

We have validated our methodology by taking previously developed measurement methodologies and shown in a controlled environment that they lead to an incorrect estimate in the number of nodes. We also show that practical crawler implementation issues can easily lead to large errors.

Concerning the actual number of nodes in the system, our results show that the number varies between 15 and 27 million per day with a very clear daily churn pattern. European users dominate both in terms of number of users and over the course of the past 30 months of our study we have seen that the number of users increased by about 10% from 2011 to 2012, but has remained stable since then.

References

[1] M. Steiner, T. En-Najjary, and E. Biersack, "Long term study of peer behavior in the KAD DHT," *IEEE/ACM Transactions on Networking*, vol. 17, no. 5, pp. 1371–1384, 2009.

[2] M. Steiner, E. Biersack, and T. Ennajjary, "Actively monitoring peers in KAD," in *Proceedings of IPTPS*, 2007.

[3] G. Memon, R. Rejaie, Y. Guo, and D. Stutzbach, "Large-scale monitoring of DHT traffic," in *Proceedings of international conference on Peer-to-peer systems*, 2009.

[4] K. Junemann, P. Andelfinger, J. Dinger, and H. Hartenstein, "Bitmon: A tool for automated monitoring of the bittorrent dht," in *Conference on Peer-to-Peer Computing*, Aug. 2010, pp. 1–2.

[5] J. Timpanaro, T. Cholez, I. Chrisment, and O. Festor, "Bittorrent's mainline dht security assessment," in *IFIP International Conference on NTMS*, feb. 2011, pp. 1 –5.

[6] C. Zhang, P. Dhungel, D. Wu, and K. Ross, "Unraveling the bittorrent ecosystem," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, p. 1, 2011.

[7] Vuze, "Distributed hash table," http://wiki.vuze.com/w/DHT.

[8] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proceedings of IPTPS*, Cambridge, MA, Mar. 7–8, 2002.

[9] J. Falkner, M. Piatek, J. John, A. Krishnamurthy, and T. Anderson, "Profiling a million user DHT," in *Proceedings of ACM conference on Internet measurement*, 2007.

[10] M. Steiner and E. W. Biersack, "Crawling azureus," Institut Eurecom, France, Tech. Rep. EURECOM+2495, 06 2008.

[11] R. Cuevas, M. Kryczka, A. Cuevas, S. Kaune, C. Guerrero, and R. Rejaie, "Is content publishing in bittorrent altruistic or profit-driven?" in *Proceedings of ACM CoNext*, dec 2010.

[12] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," *Peer-to-Peer Systems IV*, pp. 205–216, 2005.

[13] J. S. Otto, M. A. Sánchez, D. R. Choffnes, F. E. Bustamante, and G. Siganos, "On blind mice and the elephant: understanding the network impact of a large distributed system," in *Proceedings of ACM SIGCOMM*, 2011.

[14] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in bittorrent systems," in *Proceedings of ACM SIGMETRICS*, 2007.

[15] M. Meulpolder, J. Pouwelse, D. Epema, and H. Sips, "Modeling and analysis of bandwidth-inhomogeneous swarms in bittorrent," in *Conference on Peer-to-Peer Computing*, sep. 2009, pp. 232 –241.

[16] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. Al Hamra, and L. Garcés-Erice, "Dissecting BitTorrent: Five months in a torrent's lifetime," in *Proceedings of Passive and Active Measurements*, Apr. 2004.

[17] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of ACM conference on Internet measurement*. ACM, 2006, pp. 189–202.

[18] C. Zhang, P. Dhungel, Z. Liu, and K. Ross, "Bittorrent darknets," in *IEEE INFOCOM*. IEEE, 2010, pp. 1–9.

[19] M. Steiner, T. En-Najjary, and E. Biersack, "A global view of kad," *Proceedings of ACM conference on Internet measurement*, Oct 2007.

[20] M. Steiner, T. En-Najjary, and E. Biersack, "Exploiting KAD: possible uses and misuses," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 5, pp. 65–70, 2007.

[21] S. Wolchok and J. Halderman, "Crawling BitTorrent DHTs for Fun and Profit," in *Proc. 4th USENIX Workshop on Offensive Technologies*, 2010.

[22] A. N. Greg Hazel, "Bep 9: Extension for peers to send metadata files," http://www.bittorrent.org/beps/bep_0009.html, 2008.

[23] G. H. Arvid Norberg, Ludvig Strigeus, "Bep 10: Extension protocol," http://www.bittorrent.org/beps/bep_0010.html, 2008.

[24] D. Stutzbach and R. Rejaie, "Capturing accurate snapshots of the gnutella network," in *IEEE INFOCOM*, 2005.

[25] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, "Active and passive techniques for group size estimation in large-scale and dynamic distributed systems," *Journal of Systems and Software*, vol. 80, no. 10, pp. 1639–1658, 2007.

[26] A. Iosup, P. Garbacki, J. Pouwelse, and D. Epema, "A scalable method for taking detailed and accurate geo* snapshots of large P2P networks," TU Delft, PDS-2005-002, ISSN 1387-2109, Tech. Rep., 2005.

[27] L. Wang and J. Kangasharju, "Real-world sybil attacks in BitTorrent Mainline DHT," in *Proceedings of IEEE Globecom*, Dec. 2012.