Splider: A Split-based Crawler of the BT-DHT Network and its Applications

Bingshuang Liu*†, Shidong Wu*†, Tao Wei*†‡, Chao Zhang*†‡, Jun Li§, Jianyu Zhang*†¶, Yu Chen*† and Chen Li*†

*Institute of Computer Science and Technology, Peking University, Beijing 100871, China
†Beijing Key Laboratory of Internet Security Technology, Peking University
Email: {liubingshuang, wusd, wei_tao, chao.zhang, zhangjianyu, chen_yu, icst-lichen}@pku.edu.cn
‡University of California, Berkeley, CA 94720, USA
§Department of Computer and Information Science, University of Oregon, Eugene, OR 97403, USA
Email: lijun@cs.uoregon.edu

***Abstract*—Capturing accurate snapshots of peer-to-peer (P2P) networks, especially those with millions of users, is essential to many P2P-based applications, including those monitoring and analyzing P2P networks. The large scale and dynamic nature of P2P networks, however, make this task very challenging. Existent crawlers of P2P networks, for example, often miss a substantial portion of the ID space while unnecessarily crawling numerous nodes repeatedly. In this paper, we design and evaluate a new crawler called *Splider*. Unlike traditional crawling algorithms that adopt an iterative approach, Splider recursively splits the ID space of P2P nodes to crawl even tiny corners of the ID space, while avoiding crawling repeated nodes. We further implement a Splider prototype for BT-DHT, a Kademlia-based distributed hash table (DHT) P2P network, that exploits the structure of routing tables at BT-DHT nodes. Experiments show that Splider is able to gather more than 16 million nodes with a 100% recall ratio, whereas a traditional iterative crawler can at best capture only about 8 million nodes with a 66% recall ratio while its traffic-cost effectiveness is 50% less than Splider. Splider can further support distributed deployment; without any synchronization overhead, it reduces the time of capturing a full snapshot to be only about 3 minutes. We finally report and analyze the captured BT-DHT snapshots, including the spatial and temporal distribution of BT-DHT nodes and the existence of sybil and eclipse attacks in BT-DHT.**

I. Introduction

Snapshots of peer-to-peer (P2P) networks are often fundamental for studying and optimizing P2P networks. They can show characteristics of a P2P network, such as its size and the spatial and temporal distribution of its nodes, or even help discover stealthy attacks, e.g., sybil attacks [1] and eclipse attacks [2].

A common approach to capturing the snapshot of a P2P network is to use a crawler to query nodes in the network and collect their relevant information in order to build accurate snapshots of the target network. The crawler faces key challenges, however. First, the snapshot needs to be accurate; i.e., it should cover as many nodes as possible that are active when capturing the snapshot. Also, since the goal is to obtain a snapshot, but a P2P network is dynamic and nodes join and leave the network all the time, it must be done as quickly as possible. Furthermore, the crawler should be efficient and consume as little time and bandwidth as possible, such as by not crawling nodes repeatedly.

Capturing snapshots for structured P2P networks is particularly important. For instance, the top two most popular P2P file-sharing systems, BitTorrent [3] and eMule [4], are both structured P2P networks, and they adopt Kademlia [5]—one of a few Distributed Hash Table (DHT) protocols that are used in practice by tens of millions users [6], [7]. These two top P2P systems are also called BT-DHT and eMule-Kad, respectively.

However, existing crawlers are incapable of producing qualified snapshots in real world. These crawlers usually crawl networks in an iterative way [6], [8], [9], and overlook the structural characteristics of the crawled network. They frequently miss substantial portions of nodes while they crawl numerous nodes repeatedly. The crawling time to capture a full snapshot is also too long, e.g., up to 20 minutes for the BT-DHT network [6]. Due to the high churn rate of DHT networks, the captured snapshot is also inaccurate.

In this paper, we design a new, split-based crawler for Kademlia called **Splider.** We also implement a prototype of Splider to crawl a specific Kademlia network, i.e., the BT-DHT network. Splider leverages the structural characteristics of routing tables of nodes in Kademlia networks, and works in a recursive way. It can keep splitting ID spaces of nodes in the BT-DHT network into smaller ones, and thus is able to reach even tiny corners of the original ID space. Moreover, such a way of crawling greatly reduces the chance of repetition. Finally, Splider can be easily deployed in a distributed manner by parallelizing the split crawls.

We use two metrics to evaluate the accuracy and efficiency of Kademlia crawlers, i.e., the recall ratio and TCE (Traffic-Cost Effectiveness), respectively. The recall ratio [10] is the percentage of test nodes captured by the crawler. It thus reflects the crawler's ability of finding out all nodes in the whole node space (i.e., accuracy). The TCE measures how many nodes can be captured by consuming one unit of Internet traffic, and thus can be used to evaluate the efficiency of a crawler.

¶ Corresponding author. This research was supported in part by National Natural Science Foundation of China (Grant No. 61003216).

978-1-4799-2355-7/14/$31.00 ©2014IEEE

Experiments show that Splider is able to gather more than 16 million nodes with a 100% recall ratio; whereas for an iterative crawler, the best outcome is about 8 million nodes with a 66% recall ratio, and its TCE is only one half of ours. Moreover, the distributed version of Splider is able to capture a snapshot of the entire network in approximately 3 minutes, without any communication penalty. So, Splider can efficiently and accurately crawl the BT-DHT network.

Using our distributed crawlers, we have captured hundreds of snapshots of the BT-DHT network within 24 hours. As far as we know, such a data set is the most fine-grained and comprehensive measurement of the BT-DHT network. We then can learn characteristics of BT-DHT from these snapshots.

Spatial distribution of nodes: Due to the high-quality snapshots generated by Splider, we can obtain a more accurate geographic distribution of BT-DHT nodes than previous crawlers. For example, we found that Russia has the largest number of nodes at the moment, different from the conclusion in [11] and [9];

Temporal distribution of nodes: Existing study on this area is based on limited snapshots or trackers, so the results could be biased. Based on our fine-grained and complete snapshots, we can measure the temporal distribution of BT-DHT nodes from the network level and the node level respectively. For the former, we give the fluctuation of the whole BT-DHT network over time in a day. And for the latter, we find that the distribution of nodes' session lengths complies with a long tail distribution, similar to the Pareto distribution [12], rather than the Weibull distribution described in [13];

Sybil and eclipse attacks: Lacking complete network snapshots, few works have measured sybil and eclipse attacks in real networks. By analyzing snapshots, we can measure and identify sybil and eclipse attacks in the real BT-DHT network. In fact, substantial sybil and eclipse attacks have been identified.

The rest of the paper is organized as follows. After an overview of Kademlia and the related work in Section II, we present the design and implementation of Splider in Section III. The evaluation and comparison of Splider and existing iterative crawlers is given in Section IV. Section V presents an analysis of the captured BT-DHT snapshots. And Section VI concludes this paper.

II. Background and Related Work

A. Background on Kademlia

In Kademlia, both the routing tables and routing process are well structured, which ensure the high efficiency of node lookup and resource locating. The elementary structure of Kademlia networks is a shared 160-bit ID space for both nodes and keys.

According to Kademlia's specification, each node is assigned with an unique 160-bit ID generated by the SHA-1 hash function taking a random value as the input. For each file object or keyword, its hash value is used as its ID. The distance of two IDs is calculated using the bitwise *XOR* operation. The value associated with each key is stored in several nodes whose nodes are closest to the key's ID.

For routing, each node maintains a routing table consisting of 160 k-buckets. Each k-bucket has at most 8 entries, where each entry is a triple $\langle nodeID, IP, port\rangle$. In node A's routing table, nodes in its m-th k-bucket ($0 \leq m \leq 159$) share exactly an m-bit common prefix with A. For example, nodes in the 159-th k-bucket differ with A only on the last bit, while the first bit of every node in the 0-th k-bucket is different from A's.

In Kademlia, routing to a specific node (i.e., node ID) is carried out in an iterative way. The initiator sends a `FIND_NODE` request message with the target ID to a node that it already knows, and then the destination node replies this message with sufficient nodes in its routing table which are closest to the target ID. It is worth noting that, the replied nodes fall into exactly one k-bucket if the destination node has enough nodes in that k-bucket of the routing table. The initiator then queries the replied nodes and repeats this process until the target ID is found (iterative routing), instead of forwarding the request message to these replied nodes (recursive routing). While iterative routing experiences a slightly higher delay than recursive routing, it offers more robustness against packet loss. In case of Kademlia networks, it also greatly simplifies the job of crawling.

B. Related Work

As stated earlier, the essence of all existing Kademlia crawlers is iterative. The crawling process can be summarized as follows. A crawler chooses a target ID and sends requests to nodes in an initial node set, then adds the new nodes in the responses into the node set; and then starts a new round of querying with the updated node set, and so on, until some pre-determined conditions are satisfied.

Steiner *et al.* have developed a Kademlia crawler named *Blizzard* running in a single machine with 100 Mbps bandwidth [7]. This crawler adopted a simple breadth-first search and utilized an iterative query strategy. It took about 8 minutes and 3 GB of traffic-cost for crawling of the entire eMule-Kad network, and totally 4.3 million nodes were found. During the crawling, it has repeatedly crawled lots of nodes.

Jie Yu *et al.* made an improvement on the target ID selection strategy of iterative algorithms [6]. During earlier iterations for gathering bootstrapping nodes, their algorithm applies the breadth-first search; once the amount of bootstrapping nodes is enough, the depth-first search is used. However, their approach cannot truly guarantee that the bootstrapping nodes are distributed evenly in the ID space. Their crawler spent 20 minutes to collect 6.7 million nodes in the whole BT-DHT network, slower and less than our crawler's.

Xiangtao Liu *et al.* developed another iterative crawler Rainbow [9]. They theoretically analyzed the Rainbow and obtained its convergence condition, which determines the time complexity of crawling. They have measured the geographical distribution of BT-DHT nodes and found that the top 3 countries are United States, China and Russia, respectively. However, our measurement shows a different distribution, relying on the complete snapshot captured by our crawler.

In [13], Stutzbach *et al.* have systematically studied the node churn phenomenon in three typical P2P file sharing networks, Gnutella, eMule-Kad and BitTorrent. By contacting some specific BitTorrent trackers periodically, they found that session lengths of nodes comply with a Weibull or log-normal distributions, not heavy-tailed. Because one tracker is impossible to hold all nodes, and the tracker does not return all nodes to the querying node according to BEP15 [14], the trackers-based measurement is not accurate. In this paper, we analyze the node churn by capturing consecutive and complete snapshots of the BT-DHT network using Splider, and find a different conclusion.

DHT-based P2P networks are vulnerable to kinds of attacks, including the sybil and eclipse attacks. In [1], Douceur first defined the sybil attack: forging multiple identities on one physical entity. This attack can break the balance between nodes, and help other high-level attacks, e.g., eclipse attacks [2] and pollution attacks [15], to obtain enough controllable nodes in P2P networks. Subsequently, numerous sybil defenses were proposed, such as self-registration [16], net-print [17], Sybillimit [18] and SoK [19].

The eclipse attack intercepts all the requests directed to a specific resource [2]. Due to the way of locating resources, to launch an eclipse attack, the attacker first needs to forge (usually a sybil attack is beforehand needed) or control several nodes whose IDs are closer to the target ID than any real nodes. And then the attacker should announce these forged nodes to the normal nodes, in order to pollute their routing tables and to attract all lookup requests for the target ID. Several solutions have been proposed to defeat the eclipse attack, including ID-selection-based solutions [20]–[22] and computational-puzzles-based [23], [24]. Most of these defenses are based on one assumption: abundant sybil and eclipse attacks exist in these networks. However, no one measured the prevalence of these attacks under real networks. In this paper, we uncover these attacks by analyzing full BT-DHT snapshots captured by Splider.

III. Algorithm and Implementation

In this section we describe the proposed splitting approach that mitigates the shortcomings of iterative crawlers. We start by describing the basic idea of our approach, and then discuss its algorithm and implementation in detail.

A. Definitions and Observations

All nodes and keys in Kademlia share a 160-bit ID space. And all nodes in the m-th k-bucket in a node A's routing table share exactly an m-bit common prefix with A.

For the sake of discussion, we first introduce several concepts and present several observations:

*Definition 1 (**m-bit subspace**):* A subspace of the original ID space, in which all nodes share an m-bit common prefix, is called an m-bit subspace, or m-bit zone.

For each m ($0 \leq m \leq 159$), there are 2^m m-bit subspaces in the original 160-bit ID space. For instance, there is only one 0-bit subspace, i.e., the original ID space. And there are 2^{160} 160-bit subspaces which have only one node in each.

Observation 1: For any node A, all nodes in its m-th k-bucket must all belong to one same $(m+1)$-bit subspace.

*Definition 2 (**P-prefix subspace**):* If the common prefix of nodes in an n-bit subspace is P (in binary representation), we call this subspace a P-prefix subspace, or P-prefix zone.

There is an exception case, the 0-bit subspace (i.e., the original ID space) has no common prefix, and we then call it a *NULL*-prefix subspace. Of course, if an n-bit subspace is also a P-prefix subspace, the prefix P must have n bits. For any given prefix P which has less than 160 bits, there is only one corresponding P-prefix subspace.

Observation 2: P-prefix zone is a subspace of the Q-prefix zone, if and only if, Q is a prefix of P.

*Definition 3 (**Direct subspace**):* For any prefix P, the $P0$-prefix zone and the $P1$-prefix zone are called the direct subspaces of the P-prefix zone, where $P0$ is a bit pattern prefixed with P and postfixed with a 0, so does $P1$.

Observation 3: For any node A in any P-prefix zone (an m-bit zone) Z, and $n \geq m$, then all nodes in A's n-th k-bucket belong to one direct subspace of Z, i.e., a $(m+1)$-bit zone.

B. Algorithm of Splider

The core idea behind our crawler Splider is to recursively split the ID space into smaller ones, and let the crawler gather specific nodes within each subspace. It works in a recursive way as follows.

The crawler maintains a list of known IDs and updates this list dynamically. In round m, the crawler splits these known IDs into different clusters according to their m-bit prefixes. Each cluster is then called an m-bit ***s*-bucket**, and must fall into one m-bit subspace.

Then, for any given m-bit s-bucket B (belonging to the m-bit subspace Z) and any node A in it, the crawler queries A with two specific IDs, and A will response the crawler with all nodes in its m-th and $(m+1)$-th k-buckets, and maybe with some nodes in other n-th k-bucket ($n \geq m+2$) if there are no enough nodes in previous two buckets. According to Observation 3, these returned nodes fall into subspaces of Z, and thus different from known nodes not in the s-bucket B. So, the crawler can avoid meeting repeated nodes when crawling.

After all m-bit s-buckets are traversed, the crawler begins the $m+1$ round. In this recursive way, the crawler can reach even tiny corners of the ID space and traverse the whole space.

This split process is briefly demonstrated in the Figure 1. All s-buckets generated in a split process form a binary tree, called a ***S*-Tree**. For any m-bit s-bucket *B1* and $(m+1)$-bit s-bucket *B2*, an edge between them exists if and only if *B2*'s owner subspace is a direct subspace of *B1*'s. In a S-Tree, each node has at most two children nodes.

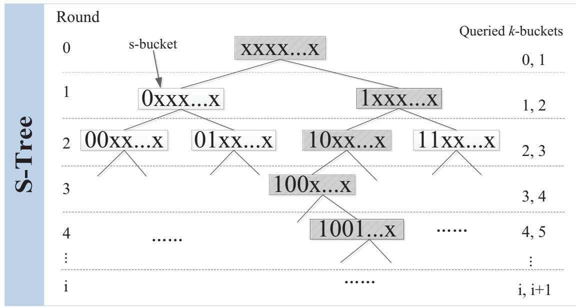


Fig. 1. An example of Splider's split process and crawling

In addition, this crawler can be configured to crawl arbitrary subspaces rather than the whole ID space by restricting the split direction. For example, if we only want to crawl the 1001-prefix subspace (a 4-bit subspace), the crawler will split along the direction as shown in the gray s-buckets in Figure 1. And thus, our crawler can be deployed in a distributed manner by nature. For example, if each crawler takes charge of an 8-bit subspace, then 256 crawlers can cooperate to crawl the whole ID space. Unlike other distributed iterative crawlers, e.g., Cruiser [25], our distributed split-based crawlers do not need synchronize any information between them when crawling respective subspaces. It can greatly speed up the crawling process and save communication resources.

Based on the scheme described above, we present the pseudo-code of the splitting algorithm in Algorithm 1. The core recursive function consists of three parts, i.e., a crawl process, a split process and a recursive invocation process.

C. Implementation

Based on the Algorithm 1, we have implemented a split-based crawler in the BT-DHT network, called Splider. To achieve a high degree of accuracy and efficiency, some concrete problems should be considered, including traffic shaping, termination condition and so on.

Traffic Shaping: In the `SplitAndCrawl` function in Algorithm 1, its crawl process will query all nodes in the target bucket with two different IDs. And thus there would be a mass of DHT queries waiting to be sent. As the split level increases, the number of queries would grow exponentially. Due to the restriction of our crawler's Internet bandwidth, we must adopt some appropriate traffic shaping and rate limiting policy. Otherwise, some queries which exceed the bandwidth limit would be dropped inadvertently, and then break the crawler's accuracy. To this end, we apply the classical token bucket algorithm [26]. The algorithm can effectively limit the rate of sending packets and efficiently utilize the crawler's bandwidth.

Termination Condition: Previous work [13] has confirmed that P2P networks are very dynamic and have a high churn rate. So if the crawler consumes too much time, a great mass of nodes may leave and join in during that time period. And thus it would compromise the accuracy of the captured snapshots. On the other hand, if the crawler terminates too early, it is difficult to capture a complete snapshot. Therefore,

Algorithm 1 The algorithm of our split-based crawler

```
Define:
  node:       struct{id, IP, port}
  s-bucket:   struct{prefix, list of nodes}
  // prefix is the common prefix of nodes in this s-bucket.
Output:
  knownList:  list of known nodes

  // get two IDs share an m-bit and (m+1)-bit prefix with node
  function CHOOSETARGETID(m, node)
      // reverse the specific bit of target ID
      id_1 = reverse_bit_at(node.id, m)
      id_2 = reverse_bit_at(node.id, m + 1)
      return id_1, id_2
  end function

  function SPLITANDCRAWL(s-bucket)
      m = s-bucket.prefix.bit_len()      // split round or level
      if m > MAX_LEVEL then Exit
      end if
      temp = s-bucket.nodes
      // Crawl process
      for all node in s-bucket.nodes do
          id_1, id_2 = CHOOSETARGETID(m, node)
          // query target node, and store returned nodes to temp
          query_node_with_ID_in_thread(node, id_1, temp)
          query_node_with_ID_in_thread(node, id_2, temp)
      end for
      sleep(1s)      // wait for the querying threads to finish
      knownList.append(temp)
      // Split process: split nodes in temp into (m+1)-bit s-buckets
      new-buckets = split_bucket(temp, prefix, m + 1)
      // Recursive invocation:
      for all bucket in new-buckets do
          SPLITANDCRAWL(bucket)
      end for
  end function

  function ENTRYFUNCTION
      initNode = pick_up_any_known_node()
      s-bucket = {NULL, {initNode}}
      SPLITANDCRAWL(s-bucket)
      return knownList
  end function
```

how to determine a reasonable termination condition is the fundamental problem when designing a distinguished crawler.

Previous crawlers usually adopt a fixed timeout, which is inflexible for our algorithm. In this paper, we propose a novel termination condition based on the maximum split level of the crawling process, referred to as *MAX_LEVEL* in Algorithm 1. Theoretically, the *MAX_LEVEL* can be up to 159 due to the 160-bit ID space. However, it is impossible and unnecessary to split so deeply, otherwise a great deal of traffic would be wasted and the crawler cannot get further benefits.

In order to determine the optimal value, we first use the crawler to crawl several subspaces and split as much as possible. Then the number of captured nodes is counted as the split level increases. From this statistical data, we can

infer the sweet spot of the split level.

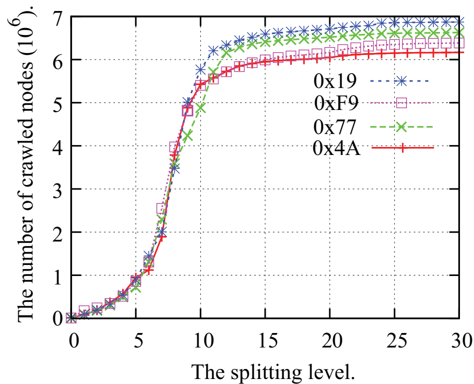


Fig. 2. The growth curve of the number of nodes crawled in an 8-bit subspace.

For instance, Figure 2 gives the growth curves of the number of crawled nodes in 4 randomly chosen 8-bit subspaces, i.e., the 0x19-prefix, 0xF9-prefix, 0x77-prefix and 0x4A-prefix subspaces. Although the counts of crawled nodes in these 4 subspaces are a little different, the trends of growth are highly consistent. It states clearly that the crawler is able to converge around the 25-th level, with a little gain after that. So we can empirically set *MAX_LEVEL* to 25. Next we will discuss this threshold from another perspective.

On the other hand, the split level has a strong relationship with the k-buckets in routing tables. For any split level m, the crawler will query target nodes' m-th and $(m+1)$-th k-buckets, and then tries to get back enough nodes from these k-buckets. However, the number of nodes in a k-bucket is dynamic. So, we conduct another experiment to show the distribution of the number of nodes in each k-bucket.

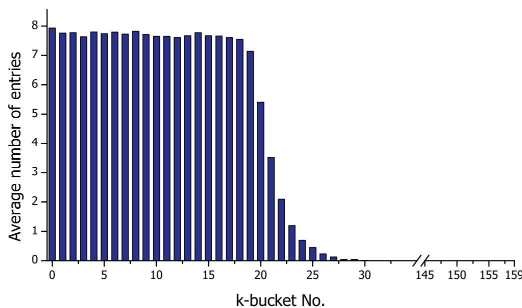


Fig. 3. The average number of entries in each k-bucket.

We randomly choose 2000 nodes from the BT-DHT network and query all k-buckets in their routing tables. The experiment has been conducted three times, and then the average number of entries in each k-bucket is calculated, as shown in Figure 3. It shows that the last 133 k-buckets (i.e., from 27 to 159) are almost empty, in consistence with the structure of routing tables in BT-DHT, and can be skipped when crawling. So only the 0-th to 26-th k-buckets need be crawled, and this goal can be achieved by setting *MAX_LEVEL* to 25. By this setting, 87.5% bandwidth resources are saved, without affecting the crawling outcome.

Some optimizations: When implementing Splider, we have introduced some optimizations in consideration of the practicality, which can be summarized as follows:

First, to reduce the number of unnecessary packets and save bandwidth resources, we stop sending queries to non-responsive nodes after two successive queries.

Second, our crawler may be added into the queried nodes' routing tables. These nodes may then include the crawler in their responses, and thus waste bandwidth resources. To avoid such case, we dynamically set the crawler's ID to make sure its first bit is different from the queried node. So, our crawler can only be added into the queried node's 0-th k-bucket. From the Figure 3, the 0-th k-buckets of almost all nodes are full (i.e., have 8 entries). And thus, our crawler is prevented from being inserted into queried nodes' routing tables in most cases.

Finally, if the *temp* bucket in `SplitAndCrawl` function in Algorithm 1 is too small, even after querying all nodes in the target s-bucket. The following split process (i.e., `split_bucket`) will not split *temp* directly because the outcome of following crawling would be limited. Instead, this split process will query for suitable nodes from some other adjacent s-buckets, until enough nodes are collected. Then, it continues the normal split process.

IV. Crawling Performance Evaluation

In this section, we evaluate the crawling performance of Splider, and compare it with the state-of-art iterative Kademlia crawler *Blizzard* [27]. Because Blizzard is not open source, and its original implementation targets another Kademlia DHT network (i.e., eMule-Kad), and thus we have re-implemented it targeting the BT-DHT network, according to its pseudo-code with aforementioned necessary optimizations. Then we give a brief description of a distributed version of our crawler Splider.

A. Single Deployment

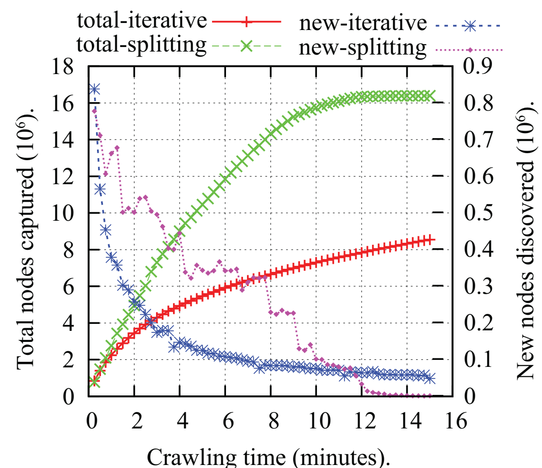


Fig. 4. Performance comparison of these two crawlers.

First, we want to examine the crawling performance of these two crawlers from a single deployment of view.

We have run two crawlers for full crawling lasting 15 minutes from 20:00 on February 25th, 2013, on two separate servers in the same location, whose configuration is 2.27 GHz Intel(R) Xeon(R) CPUs, 8GB RAM and 10 Mbps network bandwidth limitation, respectively. The main results, including total number of nodes captured and new nodes discovered, are presented in Figure 4. In Table I we summarize the comparison of the two crawlers regarding snapshot size, crawling speed, recall ratio and TCE.

Total number of crawled nodes: In the initial stage (about one minute), the crawling speeds of these two crawlers are basically identical due to the same initial nodes set. After that, the total number of nodes captured by the split-based crawler Splider rises dramatically to 15.7 million within 10 minutes, while the iterative crawler only gathers 7.3 million nodes. The splitting crawler converges after 12 minutes, with more than 16 million nodes captured in total. Yet the iterative one cannot converge when we stop the two crawlers 15 minutes later, and the snapshot size is only about 8 million. Therefore it states clearly that the splitting crawler does a better job in both snapshot accuracy and efficiency.

TABLE I. THE RECALL RATIOS AND TCES OF TWO CRAWLERS.

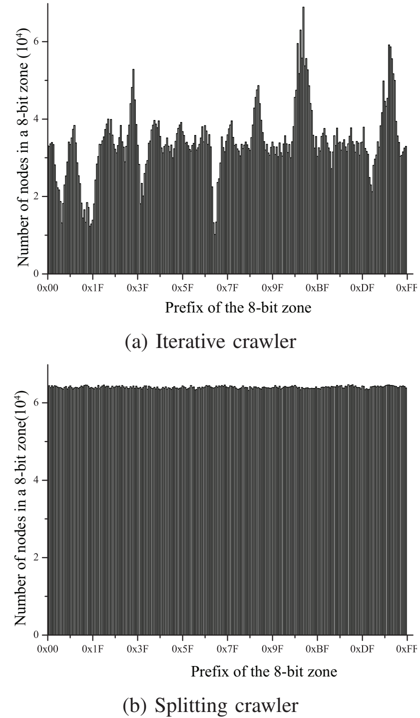
Algorithm	#Nodes captured	#Packets sent	TCE	Recall ratio
Iterative	8545605	27061701	0.316	66%
Splitting	16388586	26830961	0.611	100%

Further, we introduce a new metric to measure the completeness characteristic of one crawler, i.e., the *recall ratio*. The measurement process is as follows: we evenly deploy 256 test nodes in different 8-bit subspaces of the entire network (one in each subspace) before we start our crawlers, and then compute the percentage of test nodes captured by each crawler as its recall ratio. The result is presented in Table I. The column "Recall ratio" shows the splitting crawler has a 100% recall ratio, while the iterative crawler has recall ratio of only 66%. It indicates that the iterative one might miss a substantial portion of ID space due to the essence of blindly crawling.

To verify our conjecture, we count the nodes received in each 8-bit subspace, and the result is shown in Figure 5. It shows that the ID distribution of the iterative crawler's result is uneven, with numerous blind areas. Whereas the splitting algorithm can evenly divide the ID space. It is coincident with the fact that node IDs are generated under the consistent hash function, SHA1.

The rate of new nodes discovered: The rate of new nodes discovered by crawlers is also an important metric to evaluate crawlers' outcome. From Figure 4, the rate of the iterative crawler drops from 2.2 million/min to 0.1 million/min within the first 7 minutes, due to numerous repetitive responses. Whereas the rate of the splitting crawler is kept over 1.82 million/min even at the 7-th minute. After 12 minutes, the rate of the splitting crawler becomes lower due to its convergence.

Traffic-Cost Effectiveness: During the above experiment, the peak bandwidth cost of two crawlers is about 9.3 Mbps, and the average bandwidth cost is 8.1 Mbps. Here, we introduce another metric to evaluate the traffic effectiveness of a crawler: TCE (Traffic-Cost Effectiveness). This metric reflects how many nodes can be captured by consuming a unit of Internet traffic. The definition of TCE is as follows:



(a) Iterative crawler

(b) Splitting crawler

Fig. 5. Node ID distribution of the two approaches

$$TCE = \frac{\text{total nodes captured}}{\text{number of FIND_NODE packets sent}}.$$

Here, the traffic cost is the number of FIND_NODE packets sent during the crawling process, and the productivity is the total number of nodes captured. The result is presented in Table I. It tells that the TCE of the splitting crawler is nearly twice as much as the iterative crawler. Compared with the splitting crawler, the iterative one is more likely to waste a great deal of traffic due to repeated nodes. During this experiment, 4 million nodes are captured at least twice by the iterative crawler within the first 3 minutes, while only 150,000 nodes are encountered repeatedly by the splitting one.

B. Distributed Deployment

We know that the expensive communication overhead makes the distributed iterative crawler impractical. Since the splitting crawler can be deployed in a distributed manner by nature, we build the distributed version of it to greatly shorten the time of producing a full snapshot for Kademlia network. Obviously, the more crawlers we have or the smaller the subspaces are, the faster we obtain a full snapshot. But the bandwidth and computing resources are limited. To achieve a good trade-off, we divide the whole ID space into 256 8-bit subspaces, and assign them to 256 crawlers respectively.

It takes less than 3 minutes for a crawler to crawl a 8-bit subspace, and tasks can be executed in parallel.

Once finishing that, all crawling results would be merged into a full snapshot of the whole ID space. It means our distributed crawler is able to capture a full snapshot in approximately 3 minutes. For each server, the peak inbound and outbound traffic cost are 4.12Mbps and 6.12Mbps during the experiment, and the average inbound and outbound traffic cost are 2.2Mbps and 3.28Mbps, respectively. Based on our distributed crawler, we have measured the BT-DHT network, and some interesting phenomena will be discussed in the next section.

V. Analysis of BT-DHT Snapshots

In this section, we make use of the distributed version of Splider to continuously crawl the BT-DHT network, lasted for 24 hours, from 2013/03/12 00:00 to 2013/03/12 24:00 (UTC+8). The average time of capturing a full snapshot is about 3 minutes, and 481 continuous full snapshots in total are captured by Splider. To the best of our knowledge, it is the first comprehensive data set of BT-DHT, especially remarkable in the completeness and fine-granularity. Based on the data set, several characteristics of BT-DHT network are measured, including the spatial and temporal distribution of nodes, and the existence of stealthy attacks like sybil and eclipse attacks.

A. Spatial Distribution

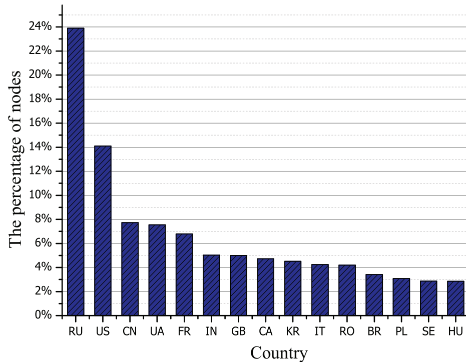


Fig. 6. The geographic distribution of nodes in BT-DHT seen on 2013/03/12.

Experiment results show that the nodes in BT-DHT are distributed in more than 200 countries. In Figure 6, we give the geographic distribution of nodes in top 15 countries. The IP addresses of nodes are mapped to corresponding countries using the latest version of Maxmind database [28]. Different from previous works [11] [9], we find that Russia rather than United States has the highest percentage of nodes, followed by United States and China, 23.9%,14.1% and 7.8%, respectively.

B. Temporal Distribution

Using Splider, we measure the temporal distribution of BT-DHT nodes from network level and node level respectively.

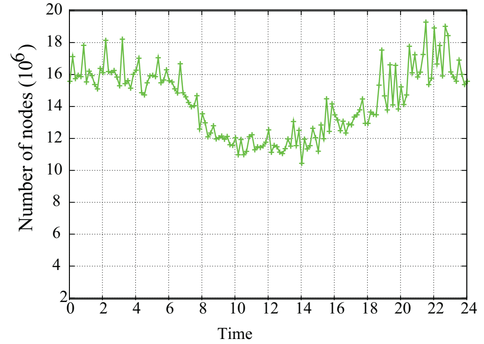


Fig. 7. The fluctuation of the BT-DHT network over time on 2013/03/12.

1) Fluctuation in Network Level: Based on our continuous snapshots of the whole network, we can figure out the fluctuation characteristics of BT-DHT network over time. In Figure 7, we give the fluctuation of the total number of BT-DHT nodes seen in a day. The distribution of BT-DHT nodes is uneven in different countries, as shown in Figure 6. In addition, due to users' habits, nodes' online behaviors likely vary at different moments even in the same country. So the scale of the BT-DHT network fluctuates over time. From 8:00 to 18:00 (UTC+8), the number of online BT-DHT users obviously falls. And between 10:00 and 15:00, the volume is only about 60% of peek hours' (21:00–23:00). To the best of our knowledge, the fluctuation of the whole BT-DHT network is measured for the first time.

2) Churn in Node Level: Churn is a basic characteristic of P2P networks, and reflects the collective effect caused by the independent arrivals and departures of numerous nodes. Stutzbach *et al.* measured the churn rate of the BT-DHT network from the view of trackers (centralized index servers) and got some preliminary conclusion in [13]. However, this view is very limited, because one tracker is impossible to hold all nodes and may not return all nodes to the querying node according to BEP15 [14]. Therefore it is necessary to understand churn by getting a full view of the BT-DHT network.

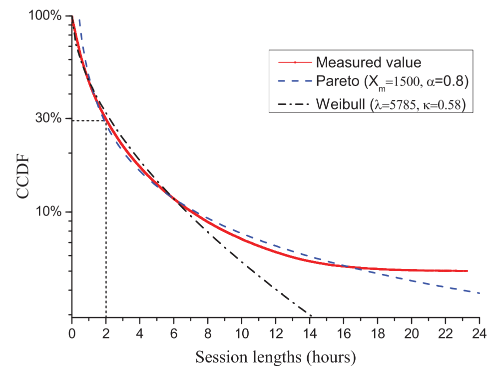


Fig. 8. The distribution of session lengths in BT-DHT.

We then evaluate the fundamental property of churn: *session length*, which means how long a node stays in the system during one session. In order to eliminate the bias towards shorter sessions in a limited measurement window, we adopt the "create-based method" [29] to fairly choose

sessions. Specifically, the measurement window is divided into two. only sessions beginning during the first half are considered. Moreover, the sessions longer than 24 hours cannot be measured due to the limited measurement window. In Figure 8, we plot the CCDF (Complementary Cumulative Distribution Function) curve of session lengths on log-linear scale. From the line “Measured value”, we find that less than 30 percent of sessions are longer than two hours and the average session length is 2.9 hours, which implies a high churn rate. However, there is an obvious long tail in this distribution, where about five percent of sessions last for 24 hours.

Using the non-linear least-squares method we fit the distribution. Figure 8 shows that the Pareto distribution is more suitable than Weibull distribution. Pareto distribution is a kind of heavy-tailed distributions, so it is coherent with the existence of a long tail in line “Measured value”. This conclusion is different from the research in [13], which claims that the distribution of session length is Weibull, not heavy-tailed.

C. Sybil and Eclipse Attacks

1) Sybil Attacks: Normally, a user sets up only one BT-DHT node on his host. However, the attacker can easily bypass this limitation and forge multiple sybil nodes by modifying the client’s source code or configurations. Hence, if multiple nodes with different IDs are locating on the same IP, a potential sybil attack is identified.

After analyzing one full captured snapshot, we identify over 200 simultaneous potential sybil attacks, each holding more than 100 sybil nodes in one IP.

TABLE II. TOP 10 SYBIL ATTACKS IN THE SNAPSHOT CAPTURED ON 2013/03/12.

rank	#sybil nodes	IP	country	organization
1	5218	91.218.230.248	Russia	eServer.ru
2	565	80.77.168.133	Russia	eServer.ru
3	435	184.73.154.187	United States	amazonaws.com
4	348	50.18.3.51	United States	amazonaws.com
5	309	106.120.108.6	China	China Telecom
6	296	149.142.151.3	United States	UCLA
7	295	67.215.242.138	United States	Secured Private Network
8	295	67.215.242.139	United States	Secured Private Network
9	266	54.247.69.34	Ireland	amazonaws.com
10	244	27.37.34.32	China	China Unicom

Table II presents the top 10 sybil attacks in the snapshot. Using the Maxmind IP database, the corresponding countries and organizations are identified. In Table II, the biggest attack has forged 5218 sybil nodes. Moreover, we can learn that sybil attackers prefer to rent public resources from hosting providers or cloud service providers. For example, the IPs ranked 1st and 2nd come from `eServer.ru`, and another two are from `Secured Private Network`, both of which are hosting service providers. And three out of the top ten IPs are from `Amazon AWS`, a well-known cloud service provider.

Another interesting finding is that IPs of numerous nodes are private, e.g., 192.168.1.1 and 192.168.0.1, which hold 3289 and 321 concurrent nodes respectively. It indicates some BT-DHT clients cannot verify the validity of nodes’ IP addresses when adding them into their routing tables. An experiment on a real uTorrent client confirms the above conjecture. Actually, these nodes with private IPs cannot make any contribution to the routing and content exchanging processes in the BT-DHT network. Therefore some validation mechanism should be introduced into BT-DHT clients.

2) Eclipse Attacks: Normal node IDs in BT-DHT are generated by SHA1, and thus roughly distribute evenly in the 160-bit ID space. Assume that the total node population is N, then the expected number of nodes (denoted as n) locating in an m-bit subspace can be computed as follows:

$$n = \frac{N}{2^m}. \quad (1)$$

During 24 hours, about 132 million unique node IDs in BT-DHT are captured by Splider. Then in a 27-bit subspace, n is nearly 1. In other words, it is suspicious that more than one node is located in the same 27-bit subspace.

On the other hand, according to previous works [23], [24], 8 nodes are sufficient to successfully conduct an eclipse attack in P2P networks holding 20 million concurrent nodes. So, if 8 or more nodes locate in the same 27-bit subspace, a suspect eclipse attack is identified.

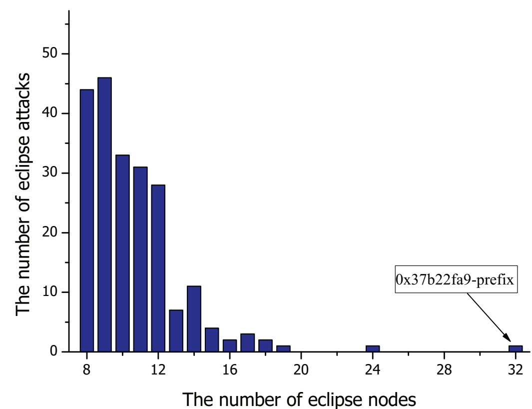


Fig. 9. The distribution of the number of eclipse nodes in an attack.

After analyzing the same snapshot used in the analysis of sybil attacks, 214 suspicious eclipse attacks were identified. Figure 9 shows the distribution of the these attacks. Actually, all nodes in over 80% eclipse attacks fall into a 32-bit subspace, much smaller than the estimated 27-bit subspace. For instance, the biggest eclipse attack has 32 eclipse nodes locating in the 0x37b22fa9-prefix subspace.

For each eclipse attack, we further calculate the IP distribution of their eclipse nodes. Then we find out that at least 80 eclipse attacks have some eclipse nodes sharing a single IP. On the other hand, real nodes sharing a single IP, e.g. nodes that adopt Network Address Translation (NAT), can hardly have adjacent node IDs (e.g., with a 32-bit common prefix), because all IDs are generated by a consistent hash function SHA1. So, these eclipse nodes sharing a single IP must be sybil nodes. In other words, these 80 eclipse attacks are built on sybil attacks.

For instance, the biggest attack in Figure 9 has 32 eclipse nodes, and 27 of them share a common IP address,

173.193.32.170, which is owned by an American hosting service provider `SoftLayer Technologies`. It is impossible that 27 real nodes sharing a single IP can fall into the same 32-bit subspace. So, we can confirm that these 27 nodes must be sybil nodes.

From this analysis, we can learn that attackers often make use of sybil attacks to carry out further attacks, such as eclipse attacks in P2P networks.

VI. CONCLUSION

In this paper, we proposed a split-based crawler Splider for taking snapshots of the BT-DHT network. Our approach addresses the major shortcomings of iterative crawlers, namely accuracy and efficiency. The experiments show that our split-based algorithm is much better than iterative algorithms in all respects, including snapshot size, recall ratio, time and bandwidth consumption, and traffic-cost effectiveness. With modest modifications, our approach can also be applied to other Kademlia networks, e.g. eMule-Kad.

Our crawler, Splider, can also be deployed in a distributed manner. Without introducing any additional communication cost, the distributed crawlers can capture a full snapshot of the entire network in only about 3 minutes. With the help of the distributed Splider, we have captured hundreds of snapshots of the BT-DHT network within 24 hours, resulting in a data set containing the most comprehensive and fine-grained measurement of the BT-DHT network, allowing us to measure BT-DHT's characteristics (node distribution and network fluctuation and churn) and detect sybil attacks and eclipse attacks, more accurately and easily.

REFERENCES

[1] J. R. Douceur, "The sybil attack," in *Proceedings of the 1st International Workshop Peer-to-Peer Systems (IPTPS)*, vol. 1. Springer, 2002, p. 251.

[2] L. Maccari, M. Rosi, R. Fantacci, L. Chisci, M. Milanesio, and L. M. Aiello, "Avoiding eclipse attacks on kad/kademlia: an identity based approach," in *Proceedings of ICC Communication and Information Systems Security Symposium*, 2009.

[3] B. Cohen, "Incentives Build Robustness in BitTorrent," in *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[4] The eMule Project. [Online]. Available: http://www.emule-project.net

[5] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[6] J. Yu, P. Xiao, Z. Li, and Y. Zhou, "Toward an Accurate Snapshot of DHT Networks," *IEEE Communications Letters*, vol. 15, no. 1, pp. 97–99, January 2011.

[7] S. Moritz, T. En-Najjary, and E. W. Biersack, "A Global View of KAD," in *Proceedings the 7th Internet Measurement Conference (IMC)*, 2007.

[8] M. Steiner, E. W. Biersack, and T. Ennajjary, "Actively Monitoring Peers in KAD," in *Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[9] X. Liu, T. Meng, K. Cai, and X. Cheng, "Rainbow: a Robust and Versatile Measurement Tool for Kademlia-based DHT Networks," in *Proceedings of International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 2010.

[10] K. Hirata and T. Kato, "Query by visual example," in *Proceedings of 3rd International Conference on Extending Database Technology (EDBT)*, 1992.

[11] P. Salvador and A. Nogueira, "Study on geographical distribution and availability of bittorrent peers sharing video files," in *Proceedings of IEEE International Symposium on Consumer Electronics (ISCE)*, 2008.

[12] B. C. Arnold, *Pareto Distribution*. Wiley Online Library, 1985.

[13] S. Daniel and R. Reza, "Understanding Churn in Peer-to-Peer Networks," in *Proceedings of the 6 th Internet Measurement Conference (IMC)*, 2006.

[14] UDP Tracker Protocol for BitTorrent. [Online]. Available: http://www.bittorrent.org/beps/bep_0015.html

[15] G. Montassier, T. Cholez, G. Doyen, R. Khatoun, I. Chrisment, and O. Festor, "Content pollution quantification in large p2p networks: A measurement study on kad," in *Proceedings of the 11th International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2011, pp. 30–33.

[16] J. Dinger and H. Hartenstein, "Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration," in *Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2006, pp. 8–pp.

[17] H. Wang, Y. Zhu, and Y. Hu, "An efficient and secure peer-to-peer overlay network," in *Proceedings of the 30th Annual IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2005, pp. 8–pp.

[18] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," in *Proceedings of 2008 IEEE Symposium on Security and Privacy*. IEEE, 2008, pp. 3–17.

[19] L. Alvisi, A. Clement, A. Epasto, S. Lattanzi, and A. Panconesi, "Sok: The evolution of sybil defense via social networks," in *Proceedings of 2013 IEEE Symposium on Security and Privacy*, 2013, pp. 382–396.

[20] S. Balfe, A. D. Lakhani, and K. G. Paterson, "Trusted computing: Providing security for peer-to-peer networks," in *Proceedings of the 5th International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2005, pp. 117–124.

[21] T. Condie, V. Kacholia, S. Sank, J. M. Hellerstein, and P. Maniatis, "Induced churn as shelter from routing-table poisoning." in *Proceedings of the 13th annual Symposium on Network and Distributed System Security (NDSS)*, 2006.

[22] L. M. Aiello, M. Milanesio, G. Ruffo, and R. Schifanella, "Tempering kademlia with a robust identity based system," in *Proceedings of 8th International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2008, pp. 30–39.

[23] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," in *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI)*. New York, NY, USA: ACM, 2002, pp. 299–314.

[24] R. Zhang, J. Zhang, Y. Chen, N. Qin, B. Liu, and Y. Zhang, "Making eclipse attacks computationally infeasible in large-scale dhts," in *Proceedings of 2011 IEEE 30th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2011, pp. 1–8.

[25] S. Daniel, R. Reza, and S. Subhabrata, "Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 267–280, 2008.

[26] The Token Bucket Algorithm. [Online]. Available: http://en.wikipedia.org/wiki/Token_bucket

[27] M. Steiner, T. En-Najjary, and E. W. Biersack, "Long Term Study of Peer Behavior in the KAD DHT," *IEEE/ACM Transactions on Networking*, vol. 17, no. 5, pp. 1371–1384, October 2009.

[28] An IP Dataase. [Online]. Available: http://www.maxmind.com

[29] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of Napster and Gnutella hosts," *Multimedia systems*, vol. 9, no. 2, pp. 170–184, 2003.