Torrent Forensics: Are your Files Being Shared in the BitTorrent Network?

Ali Alhazmi

Department of Information Systems
Jazan University
Jazan, Saudi Arabia 45142
Email: alihazmi@jazanu.edu.sa

Gabriel Maciá-Fernández

Network Engineering and Security Group, CITIC-UGR
University of Granada
Granada, Spain 18071
Email: gmacia@ugr.es

José Camacho

Network Engineering and Security Group, CITIC-UGR
University of Granada
Granada, Spain 18071
Email: josecamacho@ugr.es

Saeed Salah

Department of Computer Science
Al-Quds University
Abu Dees, Palestine 20002
Email: sasalah@staff.alquds.edu

***Abstract*—BitTorrent is the most common protocol for file sharing nowadays. Due to its distributed nature, monitoring BitTorrent is a difficult task. Under this perception of anonymity, BitTorrent has motivated the rise of criminal activities such as copyright infringement or the sharing of stolen secret documents. This work-in-progress paper focuses on identifying whether a given resource has been shared in the BitTorrent network. We have termed this problem *torrent forensics*. We propose a methodology to solve this problem as well as the design of an operational system to implement the solution. The system is run in two different phases. First, we monitor the network and collect *.torrent* files that describe the resources being shared. Second, a detection module analyzes a given resource and decides if it was observed in the network. We carry out preliminary experiments to support the hypotheses for the design of the system.**

***Keywords*–BitTorrent; P2P; Torrent Forensics.**

I. Introduction

Recently, Recently, peer-to-peer (P2P) has become popular for sharing-files around the world. P2P networks are often used to share diverse digital contents such as movies, music, books, and software. According to Cisco's estimation in 2015, P2P file-sharing users consumed 5,965 petabytes of traffic per month, which was about 15% of all the Internet traffic [1]. BitTorrent is the most common P2P file sharing nowadays. It is estimated to be responsible for more than 50% of file-sharing bandwidth and 3.35% of all total bandwidth [2]. There are millions of users sharing a huge amount of resources everyday. According to [3], BitTorrent had 15-27 million concurrent users at any time in 2013. In addition, BitTorrent Inc. claims that more than 170 million people use BitTorrent products every month [4].

The widespread popularity of BitTorrent has attracted the attention of many researchers, with the aim of studying the nature of shared resources and developing monitoring methodologies to understand the traffic evolution [5]–[8]. Bauer *et al.* [9] proposed active methods to monitor extremely large BitTorrent swarms using trackers. They developed an active probing framework called BitStalker that identifies active peers and collects concrete forensic evidences showing that they were involved in the sharing of a particular resource. Additionally, there exist some works focused on crawling *torrent-discovery sites* [7]. In the previous reference work, five of the most popular torrent-discovery sites were crawled over a nine-month period, identifying 4.6M of torrents and 38,996 trackers. They also obtained peer lists from the Vuze and Mainline Distributed Hash Tables (DHTs) in order to investigate the nature of the exchanged contents. Authors of [10] worked on large-scale monitoring of BitTorrent, crawling resources from two torrent-discovery sites: Pirate Bay and Mininova. They collected 148M of IP addresses and 2M resources over 103 days in order to identify content providers and highly-active users. Other works focus on the crawling of Mainline DHT [5] [11]. Authors in [11] collected 10M magnet links and received over 264M `get_peers` messages from more than 57M unique peers over 10 days in order to provide statistical information. In their resultsthey found that, for example, Russian and China were playing dominant roles in Mainline DHT, contributing 35% of peers, and that 5% Internet users using Mainline came from Europe.

The major contribution of this work-in-progress paper is different from those in the mentioned works. It focuses on the specific problem of identifying whether a given resource has been shared in the BitTorrent network. We have termed this problem *torrent forensics*, due to its forensic nature. To our knowledge, there is no previous published research on this topic. From a cyber security perspective, many participants can take advantage of a solution to this problem. The most widespread interest comes from end users, who may be interested in identifying

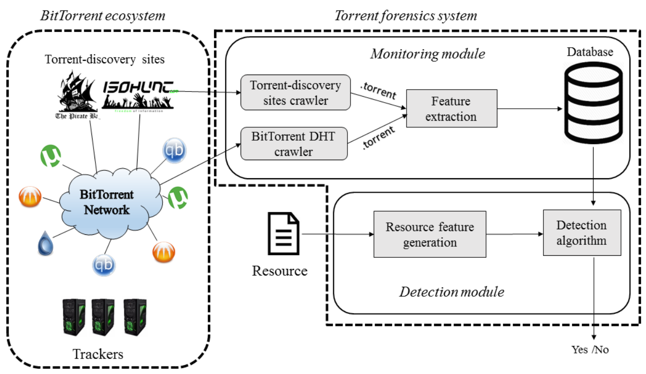

Figure 1. Architecture of the torrent forensics system.

if their private files or images are being shared in the BitTorrent network; another example is that of companies that have confidential documents and could also be interested in monitoring possible information leakages; finally, identifying the sharing of copyright materials is essential for many industries.

We propose a methodology to solve the problem of *torrent forensics* and an operational system to implement this solution. The proposed system considers two different phases. First, *.torrent* files that describe the resources being shared in the network are collected by monitoring both torrent-discovery sites and the BitTorrent DHT network. Subsequently, we build a database with the relevant features of the resources previously monitored. A main advantage of this approach is that only metadata of the resources, and not the resources themselves, need to be downloaded from the network. This saves time and storage space. In a second phase, we design a module capable of analyzing a given document and deciding if it is present in our database by comparing its features with those in the database. In this work-in-progress paper, there are some preliminary experiments to validate the main hypotheses under which our system is built.

The remainder of this paper is organized as follows. Section II discusses the design of our system and explains its components. Section III describes the experiments to evaluate the main hypotheses our system is based on. Finally, we draw conclusions and outline some future work in Section IV.

II. Torrent Forensics System

Here, we describe the design of the torrent forensics system and discuss the details and hypotheses in which it is based on. As shown in Figure 1, the system contains two main modules:

- *A monitoring module:* It is responsible for monitoring the BitTorrent network in order to obtain *.torrent* files of the resources being shared in the network. This module is also in charge of extracting some features from these files and building a database containing the monitored information.
- *A detection module:* It runs in parallel with the other module. It takes a given document as input, processes it and detects if it has been shared in the network by comparing its features with those saved in the database of monitored resources.

A. Monitoring Module

The monitoring module for our system is based on three submodules: (a) a torrent-discovery sites crawler, (b) a BitTorrent DHT crawler and (c) a feature extraction module.

1) Torrent-Discovery Sites Crawler: The purpose of the torrent-discovery sites crawler is to obtain *.torrent* files of resources that are being published in the BitTorrent network. Recall that torrent-discovery sites publish *.torrent* files that are previously uploaded by users or transferred from other torrent discovery sites. These sites usually have a query interface that allows users to get information by using an Application Program Interface (API).

In order to obtain *.torrent* files from these sites, two different strategies follow:

- Passive search: when available, Rich Site Summary (RSS) feeds to get updated information from the site about new *.torrent* files announced in the network.
- Active search: it is possible to use active crawling navigation of the web pages of the torrent-discovery sites or use APIs provided by these sites to query for existing *.torrent* files.

2) BitTorrent DHT Crawler: The main goal of this module is to obtain *.torrent* files of resources being announced in the BitTorrent DHT. For this module, we use a similar strategy to that used by the authors in [12], namely, we adapt the crawling mechanism to enable the collection of features that are used in our detection algorithm. Note that the process followed is specific for Mainline DHT, although minor modifications can be extended to the Vuze DHT [13].

The crawling process is as follows. It first gets a list of the active nodes in a specific zone of the network by sending `find_node` messages to a list of bootstrap nodes. Bootstrap nodes are used to join the network initially. Their addresses can either be hardcoded in the client software or looked up in a known directory. Subsequently, we keep active communications with them by sending `ping` messages periodically. Then, a great amount of sybil nodes are inserted as neighbors in the chosen zone of the network, in order to be included in the routing tables of known nodes in that zone. At this point, when legitimate nodes share a resource, they send `announce_peer` messages periodically, containing the *infohash* for that resource.

Once a new infohash is observed, the associated *.torrent* file must be collected. For this purpose, we send a `get_peers` message for that *infohash*, obtaining the list of peers in the swarm. Then, we query those peers so that they send us the *.torrent* file. For this purpose, the BitTorrent extension for peers to send metadata files [14] is used, in a similar way as magnet links are used to download a resource.

3) Feature Extraction: This module takes *.torrent* files as inputs from the crawling modules, and extract some features from them. A parser processes these files to read bencoded information and extract these features: (i) the *name* of the resource as identified in the *.torrent* file; (ii)

```
function GENERATE_SHA1_LIST(resource)
    Initialize piece_size_list
    SHA1_list = $\emptyset$
    for piece_size in piece_size_list do
        pieces $\leftarrow$ split(resource,piece_size)
        SHA1_list += SHA-1(pieces)
    end for
    return SHA1_list
end function
```

Figure 2. Algorithm to generate *SHA1_list* for a given resource.

the *length* of the resource in bytes; (iii) the *piece_size*; and (iv) a Secure Hash Algorithm *SHA1_list*, *i.e.*, a list of SHA-1 hashes, one for every piece that forms the resource. Finally, all this information is logged into a database, where each record includes the mentioned features for every *.torrent* file.

B. Detection module

The main aim of this module is to identify if the features obtained for a given resource are present in the monitored resources database. It is composed of two submodules: a *resource feature generation* module and a *detection algorithm* module.

1) Resource Feature Generation: This module takes the resource of interest, that is the one we are looking, as an input. In it, we emulate the data processing prior to uploading the file to the Bittorrent network. The goal is to generate the set of features that will allow the next module (detection algorithm) to find if the resource is present in the database or not. Obtaining the *length* and the *name* of the resource is straightforward. In order to obtain the *SHA1_list*, the resource is needed to be split into pieces of *piece_size* bytes and the corresponding hashes need to be calculated. The problem here is that the *piece_size* that was used in case the resource was uploaded to the network is unknown. As a matter of fact, as we will show later, the same resource may have been uploaded several times to the network with different *piece_size* values. For this reason, a list of possible candidate values for *piece_size* is selected and an *SHA1_list* is generated for every considered value. The final *SHA1_list* is compiled by joining all these SHA1 lists into a single one (see Figure 2). In Section III, we discuss a selection method for the candidate values for *piece_size*.

2) Detection algorithm: The main aim of this module is to detect whether the set of features obtained by the previous module are present in the monitored resources database. The algorithm used is shown in Figure 3. Recall that our database contains, for every resource, a record with the *name*, *length*, *piece_size* and *SHA1_list* features. *length* and *piece_size* will first be used to narrow our search and speed up the searching process, selecting only those resources with exact match. Note also that the *SHA1_list* is considered instead of the *infohash* of the resource. The main reason is that, as it is shown in Section III, BitTorrent clients might generate different *infohashes* even for a same resource. On the contrary, the *SHA1_list* remains unaltered when generating a *.torrent* file with different BitTorrent clients. Observe that the *name* is not considered in the search. Actually this feature is included to add semantic information in case a resource was renamed before being shared in BitTorrent.

Finally, our proposed system is highly dependent on two algorithms i.e. generate *SHA1_list* and search algorithm. Therefore, if any of two algorithms does not work for any reason, the proposed system will be useless.

```
function SEARCH(length,piece_size,SHA1_list)
    records $\leftarrow$ getRecordsFromDB(length,piece_size)
    for record in records do
        if record.SHA1-l in SHA1_list then
            return True
        end if
    end for
    return False
end function
```

Figure 3. Algorithm to search in the database for features extracted of a given resource.

III. Preliminary Experiments

As indicated in our introduction, in this work-in-progress paper we are only interested in verifying that the main hypotheses for the design of our torrent forensic system are validated by experimental support.

First, we check how *infohashes* for a same resource are distinct when different BitTorrent clients are used and the reasons behind that. This supports our design decision to search information in the database based on the *SHA1_list* instead of using *infohashes*.

Nowadays, there exist more than 50 BitTorrent clients that are freely available [15]. To check that *infohashes* generated by different clients for a single resource are not necessarily the same, the four most used BitTorrent clients have been selected [16] in their current versions: uTorrent v3.5 [17], Deluge v1.3.12 [18], Vuze v5.7.5.0/4 [19], and bitComet v1.45 [20]. Then, a PDF file with size 96 MB has been uploaded to every selected BitTorrent client. We choose 128 KB as piece size for all clients, obtaining the corresponding *.torrent* files with the values shown in Table I. The table shows that the *infohashes* are different.

TABLE I. INFOHASHES FOR A 96MB PDF FILE

BT client	Infohash
uTorrent	c31527aa36f7f27744c653e216b9c175223e8672
Deluge	381b9a152f902faf16a39bebd1a72cc56f946756
Vuze	b36f6aa3fbed37108a0af3eb07f4d8b7c139c38a
BitComet	feb8aaa48eac7a6a33c61270459194f2feb233da

We have investigated the reasons behind these differences in the *infohashes* among BitTorrent clients. We found that there are some differences in the `info` section generated for the *.torrent* file. First, a private parameter is added in Deluge client, even when the torrent is not private. Second, the name of the file is inserted by the Vuze client in a different place than in the others, locating it after the *SHA1_list* of hashes. Finally, the order of other parameters, such as the name, length and piece's length also lead to different *infohashes* for our selected

BitTorrent clients. In conclusion, these minor differences in the `info` section lead to different *infohashes*. Yet, in all our experiments, we have checked that the *SHA1_list* for all the pieces remains the same with all the clients. Therefore, *infohashes* cannot be used for torrent forensics, while the *SHA1_list* can.

The second hypothesis that this paper is interested to validate is with regard to the need of generating different *SHA1_lists* for every piece size in the 'document feature generation' module. The selection of a value for the piece size is a matter of optimizing the transfer speed for the download of the resource. According to the recommendation in [21], a torrent should have 1000-1500 pieces in order to get reasonably small torrent file pieces and an efficient download. In many clients, there is an auto-size option that generates *.torrent* files choosing the *piece size* parameter automatically. In our experiment, we check if all the BitTorrent clients implement the auto-size option in a similar way or they differ. We upload a file of size 175 MB to the same selected BitTorrent clients except Deluge because it does not have the auto-size option. The results from this experiment show that Vuze splits our file by 128 KB while uTorrent and BitComet choose to split it by 256KB even though the size of the file is the same. Thus, we confirm the need to generate different *SHA1_lists* for every possible piece size in the 'document feature generation' module.

Finally, regarding the initialization of the `piece_size_candidate_list` parameter in Figure 2, we consider that a good set of values are those offered to the users by these set of BitTorrent clients, *i.e.*, the set given by $j \cdot 16KB, j \in [1, 11]$ (most common clients) and $j \cdot 48\ KB, j \in [1, 7]$ (only Vuze client).

IV. Conclusions and Future Work

In this paper, we have suggested a methodology and designed a system to identify whether a given resource has been shared in the BitTorrent network. The system is based on two main modules:(i) a monitoring module to crawl the network and obtain *.torrent* files of shared resources, extracting features and saving them in a database; and (ii) a detection module, that finds if a given resource has been observed during the monitoring of the network.

Our system is currently a prototype that shows the feasibility of a partial solution for the Torrent Forensics problem. Some scale experiments should be done to complete the conclusions obtained in this paper. In addition, as future work, we plan to deal with the problem when the resources are modified before being shared in the network.

Acknowledgment

This work is supported by Jazan University through the Saudi Arabian Cultural Mission in Spain, the Spanish Ministry of Economy, and FEDER funds through project TIN2014-60346-R.

References

[1] "White paper: Cisco vni forecast and methodology 2015-2020," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html, [retrieved: September, 2017].

[2] M. Scanlon and H. Shen, "An analysis of bittorrent cross-swarm peer participation and geolocational distribution," in Computer Communication and Networks (ICCCN), 2014 23rd International Conference on. IEEE, 2014, pp. 1–6.

[3] L. Wang and J. Kangasharju, "Measuring large-scale distributed systems: case of bittorrent mainline dht," in Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on. IEEE, 2013, pp. 1–10.

[4] "Bittorrent." [Online]. Available: http://www.bittorrent.com/company/about [retrieved: September, 2017].

[5] R. A. Rodríguez-Gómez, G. Maciá-Fernández, L. Sánchez-Casado, and P. García-Teodoro, "Analysis and modelling of resources shared in the bittorrent network," Transactions on Emerging Telecommunications Technologies, vol. 26, no. 10, 2015, pp. 1189–1200.

[6] N. Andrade, E. Santos-Neto, F. Brasileiro, and M. Ripeanu, "Resource demand and supply in bittorrent content-sharing communities," Computer Networks, vol. 53, no. 4, 2009, pp. 515–527.

[7] C. Zhang, P. Dhungel, D. Wu, and K. W. Ross, "Unraveling the bittorrent ecosystem," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 7, 2011, pp. 1164–1177.

[8] P. K. Hoong, I. K. Tan, and C. Y. Keong, "Bittorrent network traffic forecasting with arma," arXiv preprint arXiv:1208.1896, 2012.

[9] K. Bauer, D. McCoy, D. Grunwald, and D. Sicker, "Bitstalker: Accurately and efficiently monitoring bittorrent traffic," in Information Forensics and Security, 2009. WIFS 2009. First IEEE International Workshop on. IEEE, 2009, pp. 181–185.

[10] S. L. Blond, A. Legout, F. L. Fessant, W. Dabbous, and M. A. Kaafar, "Spying the world from your laptop–identifying and profiling content providers and big downloaders in bittorrent," arXiv preprint arXiv:1004.0930, 2010.

[11] Z. Xinxing, T. Zhihong, and Z. Luchen, "A measurement study on mainline dht and magnet link," in Data Science in Cyberspace DSC, IEEE International Conference on. IEEE, 2016, pp. 11–19.

[12] R. A. Rodríguez-Gómez, G. Maciá-Fernández, P. García-Teodoro, M. Steiner, and D. Balzarotti, "Resource monitoring for the detection of parasite p2p botnets," Computer Networks, vol. 70, 2014, pp. 302–311.

[13] S. Wolchok and J. a. Halderman, "Crawling bittorrent dhts for fun and profit," Proc 4th USENIX Workshop on Offensive Technologies, 2010, pp. 1–8.

[14] G. Hazel and A. Norberg, "Bittorrent specification. extension for peers to send metadata files," 2017. [Online]. Available: http://bittorrent.org/beps/bep_0009.html [retrieved: September, 2017].

[15] "Bittorrent clients." [Online]. Available: http://en.wikipedia.org/wiki/BitTorrent_client [retrieved: September, 2017].

[16] W. Mazurczyk and P. Kopiczko, "Understanding bittorrent through real measurements," China Communications, vol. 10, no. 11, 2013, pp. 107–118.

[17] "utorrent." [Online]. Available: http://www.utorrent.com/ [retrieved: September, 2017].

[18] "Deluge." [Online]. Available: http://deluge-torrent.org/ [retrieved: September, 2017].

[19] "Vuze." [Online]. Available: http://www.vuze.com/ [retrieved: September, 2017].

[20] "Bitcomet." [Online]. Available: https://www.bitcomet.com/en/downloads [retrieved: September, 2017].

[21] "Torrent piece size." [Online]. Available: http://wiki.vuze.com/w/Torrent_Piece_Size [retrieved: September, 2017].