BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

MONITORING AND TRAFFIC DETECTION IN BITTORRENT

MONITOROVÁNÍ A DETEKCE PROVOZU BITTORRENT

PDS PROJECT
PDS PROJEKT

AUTHOR **TOMÁŠ HLADKÝ**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. PETR MATOUŠEK, Ph.D., M.A.**
VEDOUCÍ PRÁCE

BRNO 2023

Abstract

In this project, we created a tool that enables the detection and retrospective monitoring of BitTorrent traffic communication. We focused on detecting torrent protocols, creating a graph of the peer-to-peer network, and analysis of torrent downloads. The outcome from torrent download analysis contains information that can be used in further processing, such as the detection of downloads of files protected by copyright.

Abstrakt

V tomto projekte sme vytvorili nástroj, ktorý umožňuje detekciu a retrospektívne monitorovanie prevádzky BitTorrent komunikácie. Zamerali sme na detekciu torrentových protokolov, tvorbu grafu peer-to-peer siete a analýzu sťahovaných torrentov. Výstupná analýza sťahovaných torrentov obsahuje informácie, ktoré môžu byť ďalej použité pri ďalšom spracovaní, ako je napríklad deketkcia sťahovaných súborov chránených autorským právom.

Keywords

bittorrent, monitoring, dht, detection, sťahovanie, pcap, scapy

Klíčová slova

bittorrent, monitorovanie, dht, detekcia, download, pcap, scapy

Reference

HLADKÝ, Tomáš. *Monitoring and traffic detection in BitTorrent.* Brno, 2023. PDS project. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Petr Matoušek, Ph.D., M.A.

Monitoring and traffic detection in BitTorrent

Declaration

I hereby declare that this project was prepared as an original work by the author under the supervision of doc. Ing. Petr Matoušek, Ph.D., M.A. I have listed all the literary sources, publications and other sources, which were used during the preparation of this project.

.....
Tomáš Hladký
April 23, 2023

Contents

Chapter 1

Introduction

Peer-to-peer file sharing has emerged as an innovative approach to data distribution. One of the most widespread protocols for peer-to-peer file sharing is BitTorrent. This protocol allows each user to own some parts of a file, enabling users to share pieces of the file across the network with less resource usage compared to traditional client-server architectures. However, this protocol can be used for sharing copyrighted material over the internet which is not legal in some countries.

In response to this problem, we created a tool capable of detecting running torrent clients, performing retrospective monitoring of BitTorrent traffic, and listing individual contributors of a single torrent, including their contributed amount and contribution time. The tool can also identify individual torrents with their „info_hash“, and estimate the total torrent size while also providing the ability to generate a graph of the peer-to-peer network.

Our work is organized as follows: Chapter 2 describes BitTorrent architecture according to their specifications. Chapter 3 presents implementation details while extending BitTorrent specification used in BitTorrent clients. Chapter 4 contains an evaluation of our implementation on publicly available torrents, and Chapter 5 provides a summary and conclusion of this work.

Chapter 2

BitTorrent architecture

BitTorrent architecture is created by two major protocols: BitTorrent-Distributed hash table (BT-DHT) protocol [6] and BitTorrent protocol [3]. BT-DHT communication takes place over UDP (User Datagram Protocol) while BitTorrent protocol is over UDP but also TCP (Transmission Control Protocol). BT-DHT uses a bencode [3] to encode structural data responses, which is also used in communication with the tracker. The tracker is responsible for responding with a list of peers who own the specific torrent. Information about the tracker is stored in a specific torrent file.

When a node receives a „get_peers“ message in the BT-DHT protocol, it stores information about other nodes, including their IP address, port number, and Node ID, in DHT. The included peers correspond to those that the responding node knows for the „info_hash,“ which is a unique identifier for a torrent. This allows a single node to store a portion of other nodes in the network.

According to the DHT Protocol specification, nodes can transmit „ping“ messages to determine whether an individual node is alive [6]. However, when a node is downloading some torrent from other nodes it needs to verify liveliness through the BitTorrent protocol and maintain a connection with the nodes that possess the corresponding torrent data [1].

When downloading a torrent, it is divided into a number of pieces. This number is set by the author of the torrent, who determines the size of a single piece, which is usually a power of 2 and typically ranges from 512 KiB to 4 MiB [5]. Each piece is further divided into blocks, where the typical size is 16 KiB. Information about the piece index and the byte offset from the start of a single piece is stored in „piece“ messages of the BitTorrent protocol.

Chapter 3

Implementation

We implemented a *bt-monitor* tool capable of detecting BT-DHT (BitTorrent-Distributed hash table) protocol and BitTorrent protocol network communication.

We designed the *bt-monitor* to fulfill the following requirements:

1. **BitTorrent client initialization.** Allow analyzing initial torrent client communication with bootstrap nodes — gathering their domain name, IP address, port number, and the number of new nodes retrieved.

2. **Peers communication.** Detect potentially active torrent nodes in the network identified by Node ID, acquire their IP address and port number, and map their neighbors.

3. **Torrent download.** Detect torrent file(s) transmission through the BitTorrent protocol, specifically, file pieces transmission between individuals and other peers that share these pieces. Besides that, detect torrent file(s) with their unique hash (i.e., „info_hash“), their total size estimation, and completeness estimation with download amount and download time from each particular peer that contributes.

The bt-monitor is implemented in Python and uses several publicly available modules (e.g., Scapy [2] for pcap file analysis). All aftermentioned detections can be performed under pcap files that store captured BT-DHT and BitTorrent protocols communication. The tool does not support live capture and other formats than pcap.

3.1 Initialization

All BT-DHT communication is encoded using bencode [3] which is specifically used in BitTorrent. We analyzed existing BT-DHT communication and found that only one message type was transferred from our BitTorrent client-captured communications (see Chapter 4 for the operating system and client used) — „get_peers“ and its response. This type of message is necessary for the functionality of BitTorrent because nodes need to map the network and acquire information about other nodes. A request of the „get_peers“ message type contains one of the two following prefixes:

- „`d1:ad2:bsi1e2:id20:`“ - This prefix was used when the BitTorrent client sends the message to bootstrap nodes.

- „`d1:ad2:id20`“: - This prefix was used in communication from BitTorrent with non-bootstrap nodes.

To distinguish bootstrap nodes from non-bootstrap nodes we also analyzed DNS (Domain Name System) communication. When BitTorrent starts it needs to have initial point(s) to connect (i.e., bootstrap nodes) which are for security and also for practical reasons hard-coded as domain names in BitTorrent client implementation. The port number on which the bootstrap node listens in the BT-DHT protocol is also hard-coded[1]. By analyzing DNS query A record response we obtain IP addresses that are further requested by BitTorrent client in BT-DHT protocol with a „get_peers“ message. Response on this message type contains the following prefixes: „`d1:rd2:id20:`“ or „`d2:ip6`“. A list of bootstrap nodes can be obtained by filtering packets with a specific prefix. However, we also provide a number of unique nodes that were received by the bootstrap node. This also means that our solution reacts to situation when some bootstrap node does not respond in captured traffic. This is done by „get_peers“ response analysis. Each message contains a list of nodes which can be obtained by decoding the bencode. The content of each element is created by the IP address, port number, and ID of a new node. ID is SHA1 hash which has 20 bytes, IPv4 contains four bytes, and the rest two bytes are for the port. We also support alias mapping to the original domain name if the DNS response returns also a CNAME (Canonical Name) record.

3.2 Peers communication

This feature analyzes messages from all nodes (not just bootstrap nodes) and creates a graph of peers in the network. Similar to the initialization process, we decode bencode messages for „get_peers“ in the following manner:

- **Requests.** Give information about the node that may respond with its neighbors that can be further contacted with another „get_peers“ message.
- **Responses.** Contains data of retrieved nodes from responding node. It is essential to note that the responding node may not be directly connected to retrieved nodes. Retrieved nodes may not even be active in the network at that time. If the responding node does not share invalid information, it knew about them. Nevertheless, there may be an intersection between new nodes retrieved from multiple other nodes.

The output of peer communication analysis is a graph of the peer-to-peer network at a specific moment including active and also potentially inactive nodes. Each node contains an IP address, port number, and unique ID. However, the unique ID appears to be problematic in captured communication. Some nodes responded with an ID for the node which was already occupied by another node. This is visible when debug flag is enabled. Examples are also shown in Section 4.2.

For more in-depth processing of the network, such as in forensic analysis, we emphasize to readers that multiple network scans at different times have to be performed to create a more stable version of the graph.

[1]In qBittorrent, there are five different bootstrap nodes which can be found at qBittorrent repository `https://github.com/qbittorrent/qBittorrent` in file `src/base/bittorrent/sessionimpl.cpp` at line 1852. This observation was performed in commit `5dcc14153f046209f1067299494a82e5294d883a`.

3.3 Torrent download

Although we address multiple tool features in this work, we give particular attention to monitoring BitTorrent file downloads. The output produced by our retrospective monitoring solution includes details about the downloaded torrent, such as its estimated size and the percentage of data that has been downloaded. Moreover, the output contains a list of the individual peers who have contributed to the download process, including their IP addresses and the port numbers utilized for BitTorrent communication. This list further contains the volume of data downloaded from each peer and the approximate download time from each of them. In our approach, we do not detect tracker communication. Instead, we focus on peer communication throughout the BitTorrent protocol, specifically on messages: „handshake“, „bitfield“, and „piece“.

BitTorrent protocol can run over TCP or UDP and we support the monitoring for both. Detecting tracker communication is problematic because trackers communicate over UDP, HTTP (Hypertext Transfer Protocol), HTTPS (Hypertext Transfer Protocol over SSL/TLS), WS (WebSocket), or WSS (WebSockets over SSL/TLS). Trying to gather information from the tracker that communicates over HTTPS or WSS would not yield fruitful results. The tracker response contains useful information to start downloading the torrent. However, these data are not useful for our monitoring approach.

The analysis starts by detecting the BitTorrent protocol. This protocol utilizes a „handshake“ message to initiate communication where the prefix of this message is „`0x13Bittorrent protocol`“. Thus we can filter response from the peers that replies to the „handshake“ message. This message also contains „info_hash“, unique torrent identification which can be used to determine different torrent downloads.

BitTorrent clients communicate with peers over TCP or UDP. In TCP messages, we noticed that peers that send the „handshake“ reply also prepare part of the upcoming „bitfield“ message at the end, particularly the „bitfield“ message length and message type. We check the message type number if the upcoming message is going to be „bitfield“, which is when it equals five. The message length is compared to a packet containing „bitfield“ that the length of the received data is correct. When the upcoming packet with „bitfield“ content is received, we count the individual bits to get a number of pieces. To retrieve data from the „piece“ message we look for the „`0x0000400907`“ prefix where „`0x00004009`“ is the typical block size (16 KiB) and seven is the number for the „piece“ message type. This prefix is followed by „piece index“ which is an important value to count the size of a single piece. By storing this information we can conclude which peer was involved in a specific piece contribution.

Analyzing UDP messages is more challenging because these packets do not have identical structures. Individual packets contain data that is difficult to determine to what they belong. Unexpectedly, Wireshark, a packet analyzing tool also fails to identify BitTorrent protocol communication in UDP packets. These packets contain similar data to TCP, however, we cannot rely on their content order. For the „handshake“ message prefix („`0x13Bittorrent protocol`“), we can only check occurrence in content, but not take it as a prefix of the whole packet. After we find this in the packet, the position for „info_hash“ is the same as in the TCP packet (8 bytes after the „`0x13Bittorent protocol`“ occurrence). The „bitfield“ messages (or part of it) are stored within the „handshake“ packet. To precisely identify the start of „bitfield“ we check for the prefix „`0x05ffffffff`“ in the „handshake“ message. The „`0x05`“ is the „bitfield“ message type number and the following „`0xffffffff`“ is are bytes that mark the peer fully own these pieces. Inspecting only a

„0x05“ number in the content would lead to a false „bitfield“ message position and for this reason, we depend on a situation where at least one peer that communicates over UDP fully owns the first pieces to get a number of pieces (if there is no peer that communicates over TCP). The „piece“ message is detected similarly as in TCP — by the „0x0000400907“ prefix in the content.

To avoid false positives in the detection of „piece“ messages in both UDP and TCP, we also verify the following fields: „piece index“ and „begin offset of piece“. Both fields are four bytes long, and our method examines whether the first byte is equal to zero. This limits the maximum number of pieces to 16,777,215, which is fully sufficient because most torrents do not exceed this limit[2] (i.e., torrents with a size of TiBs or larger are split into larger pieces).

Our solution is also capable to detect multiple different torrent downloads in a single capture. However, it cannot differentiate among peers who are interested in the same torrents. In Chapter 5 we state how this issue can be resolved.

3.3.1 Estimated filesize calculation

We would like to emphasize that our method of calculating the size of a torrent does not rely on tracker communication or require the original torrent file to be read. The file size estimation is solely based on communication with peers, which is achieved by dividing the number of pieces by the size of each piece. The number of pieces is obtained from the „bitfield“ message. However, the piece size is not fixed and is determined by the torrent author. Our approach calculates the piece size by multiplying the average number of blocks per piece across all indexes with the well-known size of a single block (16 KiB). While the accuracy of the results improves with longer durations of captured communication, in Section 4.3, we demonstrate that even from short captures, we can achieve low differences compared to real size. We adopted this approach because, from Wireshark analysis, we discovered that peers may retransmit some blocks in a piece or not transmit some blocks at all.

3.3.2 Download time computation

To compute downloading from individual peers we subtract captured last and first packet timestamps with the „piece“ message type from the same peer. Downloaded size is derived from received individual pieces and calculated using single piece size.

3.4 Debug option

The debug option can be enabled with the „-debug“ flag which provides additional information, including:

- Invalid bencode message or IP address with the occupied Node ID.
- Generate network graph visualization.
- Estimations of the block count per piece, the total number of torrent pieces, and a number of downloaded pieces.

[2]We arrived at this conclusion based on a review of a sample of torrents available on websites such as https://academictorrents.com and https://thepiratebay.org/index.html.

Chapter 4

Evaluation

An evaluation was performed on Mac OS Ventura 13.1 with qBittorrent 4.5.2 [4]. One pcap (`kali_linux.pcap`) was captured on Ubuntu 20.04.6 using Deluge 2.1.1 [7]. Packet analysis and export were done in Wireshark 4.0.4. For safety reasons, the pcap files containing torrent downloads were obtained through a Wi-Fi connection to an iPhone hotspot, where the download speed was limited to 20 Mb/s.

4.1 Initialization evaluation

For `bt_bootstrap_dht.pcap` running with the `-init` argument, the tool returns 5 domain names (see Figure 4.1). However, three of them returned a list of nodes with a non-zero count from which one returned only one node. This packet can be analyzed in Wireshark to see that this bootstrap node responded with eight nodes but they all have the same IP address, port number, and also ID.

```
Bootstrap servers, that responded with certain number of nodes:
+-------------------------------------------+-----------------+-----------------+-------+-------------------------+
|                  Domain                   |      Alias      |        IP       |  Port | Number of nodes received |
+-------------------------------------------+-----------------+-----------------+-------+-------------------------+
| ec2-34-229-89-117.compute-1.amazonaws.com. | dht.aelitis.com. |  34.229.89.117  |  6881 |            0            |
|           router.bittorrent.com.          |                 |  67.215.246.10  |  6881 |           16            |
|            router.utorrent.com.           |                 |  82.221.103.244 |  6881 |            0            |
|          dht.transmissionbt.com.          |                 |   87.98.162.88  |  6881 |            1            |
|            dht.libtorrent.org.            |                 | 185.157.221.247 | 25401 |           16            |
+-------------------------------------------+-----------------+-----------------+-------+-------------------------+
```

Figure 4.1: Output from tool running with `-init` with `bt_bootstrap_dht.pcap` file.

4.2 Peers evaluation

Peers' communication can be evaluated on all included pcap files. In Figure 4.2, we have evaluated `bt_bootstrap_dht.pcap`. The graph is displayed in a table, however, if debug flag is enabled, the tool will also create an additional file `graph.pdf` that contains a visual representation of the graph (see Figure 4.3).

In our evaluation, we observed numerous nodes where the neighbor count was only one, indicating that only a single node responded to the request for information about that particular node. This occurrence can have several possible explanations, including:

- The node is not actively participating in BitTorrent and therefore does not respond to messages sent through the BT-DHT protocol. It may be stored as an inactive node in the DHT, where it was retrieved from.
- The BitTorrent client did not establish a connection with this node, potentially due to reaching the limit of DHT.
- The UDP packet was lost during transmission.
- The captured communication was either too brief or too prolonged for the node to respond.
- The node does not have any peers to share.
- The node intends to remain hidden.

ID	IP	Port	Neighbor count
3838383838383844fdd5f58d0e23ca10bab880	103.250.184.81	5353	1
aa6e48e7ac0f40531317b2a08bd92f1d6f538e9e	31.46.255.231	43607	1
69faff71309b7ac83793a7249734e109052717b6	176.105.36.69	32000	1
c3d5415e501cfb9052e50e532b15b45987c852b6	90.202.212.72	52225	9
2d68c57762409a12d59db77de952129f38c91bdd	151.225.140.39	47189	1
916a0cf7b8e8a6e2d816533a9e25a0d05a0b63b8	109.252.174.89	1436	1
63f1fb40c0957ae91dcceb510701d8f51d6985d7	31.128.76.150	62602	1
1abdee27c8eb54f17dff59aa97169fbc158a46a3	102.72.117.150	6881	1
157d0a8341f542184da9d6f964ceb6703597807b	188.119.58.29	62311	1
8faefb807ad6d2b420680f7010011d0cd426d385	93.123.133.16	6881	1
0c01373d8bc94b9a89e0a9ff10f986a948981934	181.50.102.144	20069	1

Figure 4.2: Part of the output from tool running with `-peers` with `bt_bootstrap_dht.pcap` file.

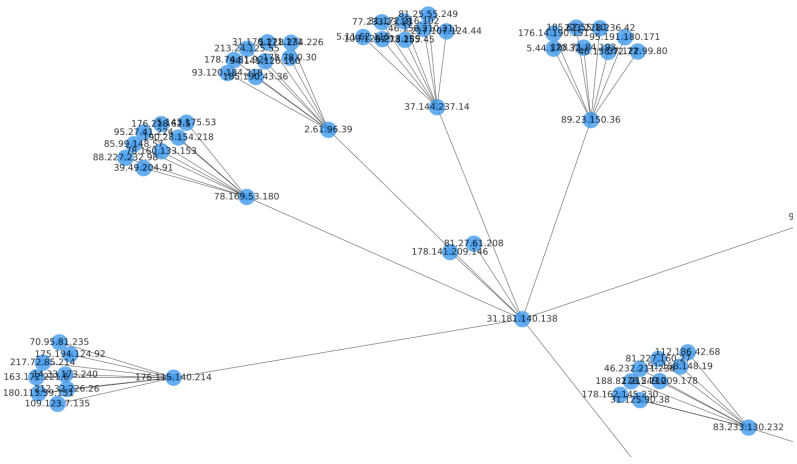

Figure 4.3: Graph generated from tool with `bt_bootstrap_dht.pcap` with `-peers` and `-debug` flags enabled. The figure only shows part of the graph for better visibility.

Table 4.1: Comparison between the actual total size and estimated size of the torrent, as well as a comparison between the actual downloaded size percentage and estimated downloaded size percentage.

PCAP filename	Torrent	Real size	Est. size	Real %	Est. %
`ac_crater.pcap`	1/1	30.9 MiB	31.02 MiB	100%	100%
`two_torrents_diff.pcap`	1/2	3.79 GiB	3.32 GiB	1.7%	2.0%
`two_torrents_diff.pcap`	2/2	2.49 GiB	1.93 GiB	1.7%	2.3%
`ac_paris_build.pcap`	1/1	2.43 GiB	2.41 GiB	26.4%	26.6%
`kali_linuxs.pcap`	1/1	3.60 GiB	2.93 GiB	1.0%	0.7%

```
Torrent 1/1:
+------------------------------------------+---------------------+-------------------------+
|                Info hash                 |   Total size est.   | Total % downloaded est. |
+------------------------------------------+---------------------+-------------------------+
| 30748b1a7ac99b1c5ff66f0bc5c5f7428ed035c5 | 32.52 MB (31.02 MiB) |         100.0%          |
+------------------------------------------+---------------------+-------------------------+
+---------------------+-------------------+-------------+----------------------------+
|    Peer IP + Port   | Amount downloaded | % downloaded |  Estimated downloading time |
+---------------------+-------------------+-------------+----------------------------+
|   72.21.17.2:25000  | 6.00 MB (5.72 MiB) |    18.4%    | 2 minutes and 12.95 seconds |
| 185.203.56.27:55887 | 6.03 MB (5.75 MiB) |    18.5%    | 2 minutes and 14.23 seconds |
| 185.149.90.114:51033 | 6.21 MB (5.93 MiB) |    19.1%    | 2 minutes and 13.58 seconds |
|  95.85.214.21:51413 | 6.00 MB (5.72 MiB) |    18.4%    | 2 minutes and 10.79 seconds |
| 140.211.167.14:25000 | 2.79 MB (2.66 MiB) |     8.6%    | 2 minutes and 10.41 seconds |
| 91.114.189.62:16881 | 5.49 MB (5.24 MiB) |    16.9%    |  2 minutes and 4.86 seconds |
+---------------------+-------------------+-------------+----------------------------+
```

Figure 4.4: Output from tool running with `-download` with `ac_crater.pcap` file.

4.3 Torrent download evaluation

The evaluation of torrent downloads was performed using multiple sources[1]. The downloaded content from academictorrents.com was not shared with other users, i.e., no other individuals were downloading the same torrent. In cases where torrents were downloaded from websites such as thepiratebay.org, it was ensured that the content was protected by the GNU GPLv3 (General Public License version 3). This license grants users the right to distribute content that is covered by the license while confirming compliance with legal requirements. The purpose of such variety is to test communication with peers that may also share content protected by copyright. More details are included in the `Readme.txt` file.

In Table 4.1, we present the results of our algorithm for estimating the torrent size and evaluating its accuracy. Our analysis indicates that the accuracy of the estimated total size and downloaded percentage increases with longer durations of captured communication. Additionally, in Figure 4.4, we present a tool output for a torrent that was completely downloaded.

[1] https://torrent.ubuntu.com/tracker_index, https://academictorrents.com, https://thepiratebay.org/index.html, https://www.kali.org/get-kali/

Chapter 5

Conclusion

We presented a tool for detecting and retrospectively monitoring torrent traffic. Our tool detects the BitTorrent client initialization and gathers information about nodes in the peer-to-peer network, while also providing an option to generate a network graph. In addition, the tool analyzes BitTorrent protocol communication for file(s) downloads and produces a list of nodes that contributed to the torrent content, including their amount contributed amount and the contribution time. Furthermore, we also proposed an approach for estimating the total torrent size. Achieved results show that the tool can be used for detecting illegal data sharing. If the torrent estimation size is relatively small (in the range of a few GiBs), it could be recognized as a film protected by copyright. Additionally, we can check if a known IP address appears in the list of contributors that are sharing data through the BitTorrent protocol.

The tool can be further extended to detect BitTorrent traffic from multiple torrents involving the same subset of peers which can be achieved by detecting different ports in the „handshake“ message as each torrent download from an individual peer opens a new port in the BitTorrent client for downloading. Moreover, peer-to-peer network analysis can be improved to detect only active nodes from longer communication, resulting in a more stable version of the output graph.

Bibliography

[1] *BitTorrentSpecification* [online]. [cit. 2023-04-20]. Available at: https://wiki.theory.org/BitTorrentSpecification.

[2] Biondi, P. *Scapy library* [online]. 2003 [cit. 2023-04-20]. Available at: https://scapy.net/.

[3] Cohen, B. *The BitTorrent Protocol Specification* [online]. 2008 [cit. 2023-04-20]. Available at: https://www.bittorrent.org/beps/bep_0003.html.

[4] Dumez, C. *QBitTorrent* [online]. 2006 [cit. 2023-04-20]. Available at: https://www.qbittorrent.org/.

[5] Hamra, A., Legout, A. and Barakat, C. Understanding the Properties of the BitTorrent Overlay. august 2007.

[6] Loewenstern, A. and Norberg, A. *DHT Protocol* [online]. 2008 [cit. 2023-04-20]. Available at: https://www.bittorrent.org/beps/bep_0005.html.

[7] Tibbitts, Z. et al. *Deluge* [online]. 2006 [cit. 2023-04-20]. Available at: https://deluge-torrent.org/.