BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

MONITORING OF BITTORRENT TRAFFIC IN LAN

DATA COMMUNICATIONS, COMPUTER NETWORKS AND PROTOCOLS

TERM PROJECT

AUTHOR **TEREZA BURIANOVÁ**

BRNO 2023

Contents

List of Figures

Chapter 1

BitTorrent architecture and communication

BitTorrent is a protocol for downloading files, utilizing the peer-to-peer architecture. The files are divided into several pieces, and during the file download, all of the pieces are eventually downloaded from various users, forming the whole file. The already acquired pieces are also shared with other users who request them at the moment [1].

To create the network of peers, the BT-DHT protocol is used in modern BitTorrent. The information regarding node's neighbours is stored in a peer-to-peer distributed hash table called Kademlia. The Kademlia system allows to measure distance between two points stored in the table. The distance metric for this calculation utilizes the XOR operation. This way, a node can identify other „close" nodes and eventually fill its routing table with reliable and close nodes. The information about possible existing nodes is acquired by recursively asking the nodes, that the given node already knows about, for their lists of other nodes known by them [2, 4].

When a file download is started, the client needs to find peers who have one or more pieces of the requested file. The peers spread the information about pieces in their possession through the network using BT-DHT protocol, therefore the peers discover each other based on the pieces they request and possess. These pieces are then downloaded and further shared among other nodes who request them in a recursive way. The peer should aim to download and share corresponding amount of data, otherwise it gets labeled as a „leech". There are measures against such a behaviour, which may prevent the „leech" from further downloading. In that way, the network can keep running, as the downloading is not possible without peers that are also sharing [1, 2].

In this project, the behaviour of these protocols was analysed and the findings were further described in detail. Methods for protocol detection were proposed and finally, the implementation of the chosen methods was presented.

Chapter 2

Experiments with BitTorrent client

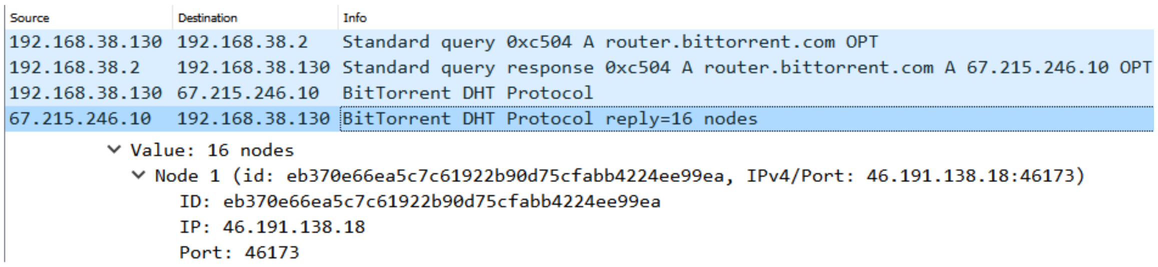
To analyse the BitTorrent protocol and its behaviour, two captures were created, using the qBittorrent client [1]. The *small.pcap* capture contains packets from the initial client launch and the download data of two torrents. The *large.pcap* capture does not include client initialization and includes the download data for one large torrent. Both of the captures were started before the client launch. The findings were also compared to the captures provided in the assignment.

The initial packets include DNS queries for the bootstrapping nodes in both captures. These nodes are hardcoded in the qBittorrent client. The only visible difference between the capture of the initial launch and the capture of another launch is the occurrence of the NAT-PMP requests for external address and port mapping during the initialization. The BT-DHT get_peers requests are then sent to the bootstrapping nodes in both cases. These requests can be identified by the „bs“ value in the protocol data, but that is not the case for all of the clients, therefore the value cannot be used to reliably get the bootstrap requests by itself. Some of the nodes used in the BT-DHT communication were not included in a get_peers response in the non-initial run capture, meaning the routing table was at least partially maintained from the previous launch.

The BT-DHT communication uses port number 8999 for the entire duration, as that is the default value in the qBittorrent settings. Unlike some other clients, such as BitTorrent Classic, the qBittorrent client uses get_peers messages to get both nodes and peers. An example of this transaction is shown in Figure 2.1. According to a blog post by Arvid Norberg [5], get_peers acts similarly to find_node in this context and reduces privacy threats. Not all of the requests get responded to, as not all of the nodes are active. The responses can be identified and matched using the unique value called „Transaction ID“. Once a node responds to a get_peers request with a list of nodes, these can be contacted and potentially also added to the node's routing table. The nodes are chosen based on their „closeness“ to the client node (XOR of node IDs, as described in Kademlia) and their activity and reliability. If the node does not reply to several requests, it is considered inactive and removed from the routing table. The node only contacts nodes from get_peers responses if its routing table is not filled with reliable nodes, otherwise the nodes information gets discarded [2].

To get information about the location of desired file pieces, the peers ask a tracker for peer contact regarding the requested file. In modern BitTorrent, the previously described BT-DHT protocol allows peers to store the information themselves, making all of the peers

[1] https://www.qbittorrent.org/

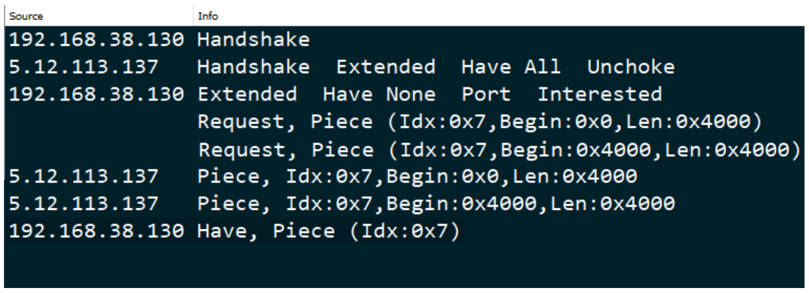


```
Source          Destination     Info
192.168.38.130  192.168.38.2    Standard query 0xc504 A router.bittorrent.com OPT
192.168.38.2    192.168.38.130  Standard query response 0xc504 A router.bittorrent.com A 67.215.246.10 OPT
192.168.38.130  67.215.246.10   BitTorrent DHT Protocol
67.215.246.10   192.168.38.130  BitTorrent DHT Protocol reply=16 nodes
         ⌄ Value: 16 nodes
           ⌄ Node 1 (id: eb370e66ea5c7c61922b90d75cfabb4224ee99ea, IPv4/Port: 46.191.138.18:46173)
               ID: eb370e66ea5c7c61922b90d75cfabb4224ee99ea
               IP: 46.191.138.18
               Port: 46173
```

Figure 2.1: An example of BT-DHT communication, showing a DNS query to get the bootstrap node IP, a get_peers request and a response containing information about 16 other nodes, with one shown in detail.

trackers. Several messages are defined in the BitTorrent protocol to get the desired information from the neighbouring peers. The communication starts with a Handshake message, which is responded to with another Handshake message to establish the connection. The message contains an info_hash, a SHA1 hash identifying the downloaded file (or collection of files) based on the filename, size and other attributes. Then, the peers share information about possessed file pieces with each other, using either the Have message or the Bitfield message. The peers start the communication as „Not interested“, meaning they have no interest in downloading from the other peer yet, and „choked“, meaning they do not allow the other peer to request and download pieces yet. A peer can send Interested message to show its interest in some of the data of the other peer. The other peer can send an Unchoke message when ready to share data, allowing the peer to send Request messages for the desired pieces. Peers can send Choke message at any time to choke the communication. If the peer received the request, it sends a Pieces messages, containing the requested piece data [6, 2].

An example of the communication can be seen in Figure 2.2. The client sends a Handshake and receives a Handshake in return. The Have All message is part of an extension and acts as a set of Have messages for all of the pieces. The peer immediately Unchoked the client, therefore the Requests can be sent immediately after the Interested message. The requested pieces arrive in the Pieces message and the client can then announce its possesion of the piece sending the Have message.



```
Source          Info
192.168.38.130  Handshake
5.12.113.137    Handshake  Extended  Have All  Unchoke
192.168.38.130  Extended  Have None  Port  Interested
                Request, Piece (Idx:0x7,Begin:0x0,Len:0x4000)
                Request, Piece (Idx:0x7,Begin:0x4000,Len:0x4000)
5.12.113.137    Piece, Idx:0x7,Begin:0x0,Len:0x4000
5.12.113.137    Piece, Idx:0x7,Begin:0x4000,Len:0x4000
192.168.38.130  Have, Piece (Idx:0x7)
```

Figure 2.2: An example of BitTorrent communication, showing the beginning of communication with one of the peers.

Chapter 3

Detection of BitTorrent traffic

To get information regarding the Bittorrent communication, at least two specific protocols need to be detected and analysed: BT-DHT for the peer contact data and BitTorrent for information about the downloading of files. These protocols do not have fixed ports and the values determined in standards are not always abode by, therefore other methods for detection need to be taken into consideration. Possible techniques, some of which were later used to implement a detection tool, are further described in this chapter.

3.1 BT-DHT

Detecting the BT-DHT packets, especially during heavy network traffic, is not a straightforward issue, as the protocol does not use fixed port values. Some of the options, which may lead to the protocol identification, include headers analysis, signature analysis or utilization of statistical methods [3].

During client initialization, possible ports utilized for the BT-DHT communication can be found by analysing the NAT-PMP or PCP packets, assuming the network capture is complete and unfiltered and the traffic was not particularly heavy at the time of the capture. The BT-DHT packets can be found based on the communication protocol (UDP) and the acquired ports. This method only works if NAT forwarding is enabled in the client and the packets from client initialization after the first launch or a settings change is available, so the method is not reliable enough to use in an automatic script.

One of the options is signature analysis, which is based on searching for the well known BT-DHT requests, like get_peers or find_node. These can be found in the form of a string in the UDP payload. That is not the case for the corresponding responses, which would have to be matched using transaction IDs.

For the purpose of the detection tool implementation, which aims to process simple network traffic captures, the signature analysis was not used as described due to the possibly high time requirements, especially for larger captures. Instead, the port values were found using signature analysis on a limited amount of packets and and the traffic was later filtered using the discovered port number. The combination of these methods is reliable, as most clients use the same port number during the whole communication and the utilized packets are usually present in the communication.

3.2 BitTorrent

The BitTorrent communication cannot be filtered based on a port number like in the case of BT-DHT, because the port regularly changes based on the peers that the client communicates with. Instead, packets can be reliably detected using the signatures analysis method, as there are several well known signatures that can be easily found in the packets data [1], although the method could possibly be slower for large captures, depending on the implementation.

3.3 Tool for detection

The tool implements detection and data analysis of the BT-DHT and BitTorrent (TCP version) protocols, using a .pcap or .pcapng file as input. The required TShark (Wireshark) version is 4.0.0 or higher, as the tool is utilized to filter and dissect the input packets. The syntax is following:

```
bt-monitor -pcap <file.pcap> -init | -peers | -download
```

The **-init** argument is used to get a list of bootstrap nodes, including their domains, IPs, port numbers, information about potential requests to these nodes during the client initialization and the potential responses to these requests.

The **-peers** argument shows a list of all of the contacted nodes in the BT-DHT protocol that responded at least once. Each of the records contains Node ID, IP, port and number of connections with that particular node. The name of this argument is inconsistent with the „peer“ definition in BT-DHT standard [2], which only considers peers to be the participants who implement the BitTorrent sharing. In the tool, peers are considered to be the participants in the BT-DHT protocol, otherwise called „nodes“.

The **-download** argument shows the info_hash of all of the downloaded files and a list of all of the pieces, including the contributing peers for each of them. It is recommended to redirect the output to a file for large captures.

3.3.1 Implementation

To filter the traffic and dissect the relevant packets, the tool TShark was used. Tshark can process display filters, equivalent to Wireshark, and export the chosen fields to .csv files, which are then further processed to get the desired output.

As described in section 3.1, the tool analyses initial requests to the bootstrap nodes, which are detected in DNS using several keywords. The port used in requests with these nodes as the destination is the port further used for the BT-DHT communication. This approach is valid if the protocol uses the chosen node during the whole communication without change. Fixed port number is the default behaviour of qBittorrent client, used to capture the included traffic.

For the **-init** implementation, the bootstrapping nodes are acquired from the initial DNS queries and saved to a list of IP addresses. The requests to the bootstrapping nodes are packets sent to these addresses and the matching responses are found using the „Transaction ID“. The port used for BT-DHT communication matches the destination port of these

[1] https://l7-filter.sourceforge.net/layer7-protocols/protocols/bittorrent.pat

packets, which allows to detect the whole BT-DHT communication and get all necessary information.

For the **-peers** argument, the capture does not always include the DNS queries, so instead, the filter gets the first BT-DHT request based on signature analysis, as described in section 3.1. The port is determined based on this packet and the communication can then be filtered. The amount of nodes included in the get_peers responses is too high, therefore only the nodes that responded at least once are included in the output. This approach only requires the list of responses, as the packets include the node ID, the IP and port and the number of connections is calculated based on the number of responses, which corresponds to the unique transaction IDs.

The **-download** argument uses the built-in functionality of TShark display filter to get the BitTorrent communication. First, the tool creates a list of all downloaded files, respectively their info_hash, and the peers with which the connection was established using handshakes. The remaining communication is then separated and for each file, pieces and contributors are acquired using the Piece messages. If no Pieces messages are found, UDP protocol or extensions are probably used, therefore not detected by the tool.

3.3.2 Testing

The tool was continuously tested during the implementation process on both of the captures, this subsection only discusses the final testing. The tests are based on comparison of some of the randomly chosen output values with data shown in Wireshark.

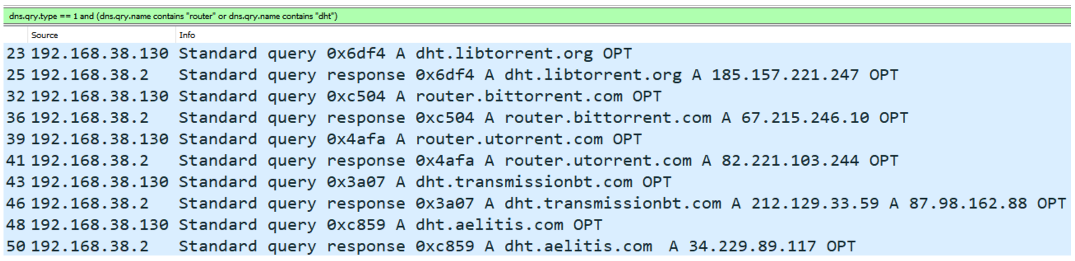
Output of -init contains data about bootstrap nodes and the initial requests and responses. This process is very similar in both of the captures, as described in chapter 2, so the *small.pcap* capture was chosen. The tool was started using the following command:

```
./bt-monitor -pcap ./pcap/small.pcap -init > ./tests/small_init.txt
```

The output of the tool:

```
185.157.221.247 : dht.libtorrent.org
1 initial request sent to port 25401, transaction ID 08ca
1 response received: BitTorrent DHT Protocol reply=3 nodes

67.215.246.10 : router.bittorrent.com
1 initial request sent to port 6881, transaction ID 6e98
1 response received: BitTorrent DHT Protocol reply=16 nodes

82.221.103.244 : router.utorrent.com
1 initial request sent to port 6881, transaction ID b2aa
no response received

212.129.33.59 : dht.transmissionbt.com
no initial request sent

87.98.162.88 : dht.transmissionbt.com
1 initial request sent to port 6881, transaction ID 8551
1 response received: BitTorrent DHT Protocol reply=8 nodes
```

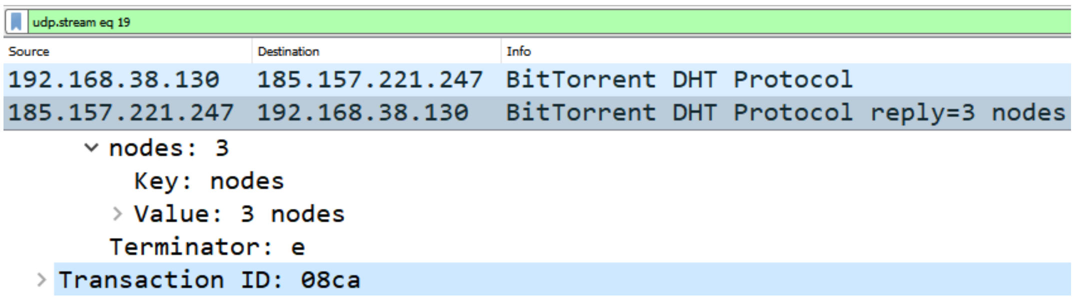
```
34.229.89.117 : dht.aelitis.com
1 initial request sent to port 6881, transaction ID ff5c
no response received
```



```
dns.qry.type == 1 and (dns.qry.name contains "router" or dns.qry.name contains "dht")
   Source         Info
23 192.168.38.130 Standard query 0x6df4 A dht.libtorrent.org OPT
25 192.168.38.2   Standard query response 0x6df4 A dht.libtorrent.org A 185.157.221.247 OPT
32 192.168.38.130 Standard query 0xc504 A router.bittorrent.com OPT
36 192.168.38.2   Standard query response 0xc504 A router.bittorrent.com A 67.215.246.10 OPT
39 192.168.38.130 Standard query 0x4afa A router.utorrent.com OPT
41 192.168.38.2   Standard query response 0x4afa A router.utorrent.com A 82.221.103.244 OPT
43 192.168.38.130 Standard query 0x3a07 A dht.transmissionbt.com OPT
46 192.168.38.2   Standard query response 0x3a07 A dht.transmissionbt.com A 212.129.33.59 A 87.98.162.88 OPT
48 192.168.38.130 Standard query 0xc859 A dht.aelitis.com OPT
50 192.168.38.2   Standard query response 0xc859 A dht.aelitis.com  A 34.229.89.117 OPT
```

Figure 3.1: DNS communication regarding bootstrapping nodes in initial BT-DHT communication.

As seen in Figure 3.1, all of the qBittorrent bootstrap nodes were correctly recognized. There is no communication sent to IP *212.129.33.59*, as expected based on the output. The node chosen for the check was *dht.libtorrent.org*. As seen in Figure 3.2, the information matches the output information.



```
udp.stream eq 19
Source          Destination     Info
192.168.38.130  185.157.221.247 BitTorrent DHT Protocol
185.157.221.247 192.168.38.130  BitTorrent DHT Protocol reply=3 nodes
      ˅ nodes: 3
          Key: nodes
        › Value: 3 nodes
        Terminator: e
    › Transaction ID: 08ca
```

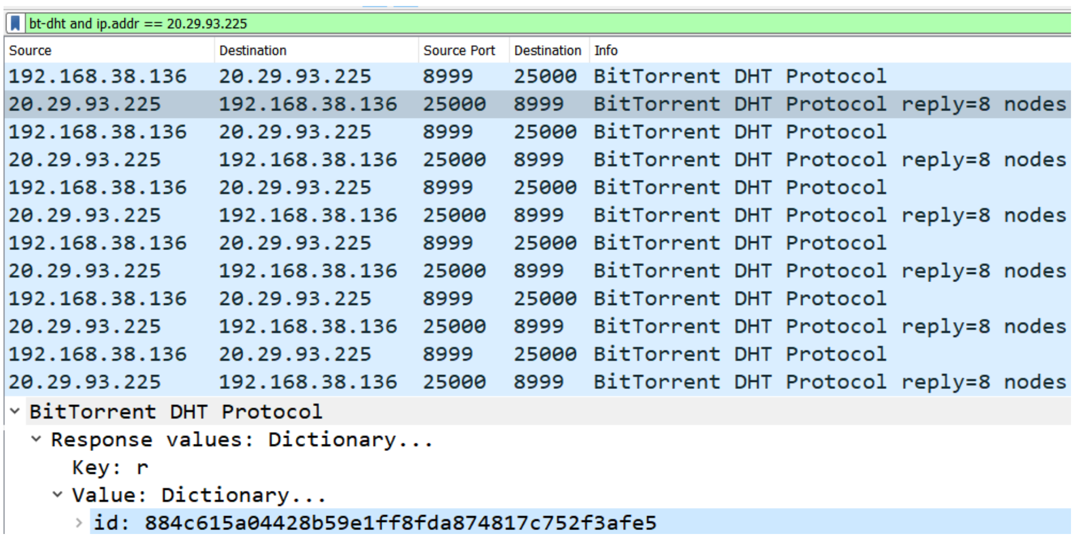
Figure 3.2: Initial communication with bootstrap node dht.libtorrent.org.

Output of -peers is more informative in the *large.pcap* capture, as there was more time for BT-DHT communication. The tool was started using the following command:

```
./bt-monitor -pcap ./pcap/large.pcap -peers > ./tests/large_peers.txt
```

The output contains information about 86 nodes. One of the nodes with a larger amount of connections was chosen for a check:

```
Node ID 884c615a04428b59e1ff8fda874817c752f3afe5:
20.29.93.225:25000, 6 connection(s)
```



```
bt-dht and ip.addr == 20.29.93.225
Source          Destination     Source Port  Destination  Info
192.168.38.136  20.29.93.225    8999         25000        BitTorrent DHT Protocol
20.29.93.225    192.168.38.136  25000        8999         BitTorrent DHT Protocol reply=8 nodes
192.168.38.136  20.29.93.225    8999         25000        BitTorrent DHT Protocol
20.29.93.225    192.168.38.136  25000        8999         BitTorrent DHT Protocol reply=8 nodes
192.168.38.136  20.29.93.225    8999         25000        BitTorrent DHT Protocol
20.29.93.225    192.168.38.136  25000        8999         BitTorrent DHT Protocol reply=8 nodes
192.168.38.136  20.29.93.225    8999         25000        BitTorrent DHT Protocol
20.29.93.225    192.168.38.136  25000        8999         BitTorrent DHT Protocol reply=8 nodes
192.168.38.136  20.29.93.225    8999         25000        BitTorrent DHT Protocol
20.29.93.225    192.168.38.136  25000        8999         BitTorrent DHT Protocol reply=8 nodes
192.168.38.136  20.29.93.225    8999         25000        BitTorrent DHT Protocol
20.29.93.225    192.168.38.136  25000        8999         BitTorrent DHT Protocol reply=8 nodes
˅ BitTorrent DHT Protocol
  ˅ Response values: Dictionary...
      Key: r
    ˅ Value: Dictionary...
      › id: 884c615a04428b59e1ff8fda874817c752f3afe5
```

Figure 3.3: An example of all BT-DHT communication with one node.

As seen in Figure 3.3, the IP, port number, node ID and number of transactions match the output. The BT-DHT response containing this node was not found, meaning it was probably stored from the previous client launch.

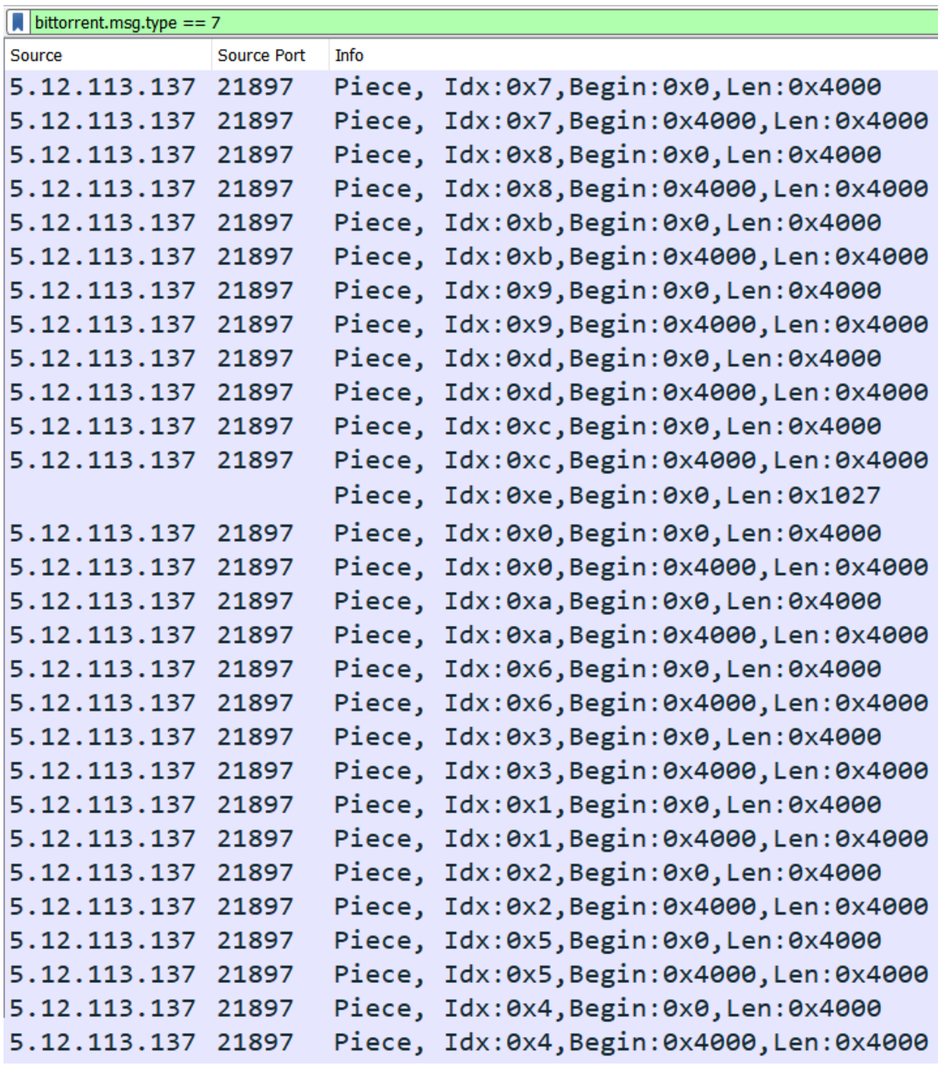
Output of -download is checked on the *small.pcap* capture, as the output is shorter and contains data about two file downloads. The tool was started using the following command:

```
./bt-monitor -pcap ./pcap/small.pcap -download > ./tests/small_download.txt
```

The output of the tool:

```
info_hash: 8c271f4d2e92a3449e2d1bde633cd49f64af888f
pieces and their contributors:
No Pieces messages found for info_hash
8c271f4d2e92a3449e2d1bde633cd49f64af888f.
Possibly UDP or an extension used.

info_hash: 13fe7880330170e8b9a2712846994bd3252c8384
pieces and their contributors:
piece index 0x00000007, contributors ['5.12.113.137:21897']
piece index 0x00000008, contributors ['5.12.113.137:21897']
piece index 0x0000000b, contributors ['5.12.113.137:21897']
piece index 0x00000009, contributors ['5.12.113.137:21897']
piece index 0x0000000d, contributors ['5.12.113.137:21897']
piece index 0x0000000c, contributors ['5.12.113.137:21897']
piece index 0x0000000e, contributors ['5.12.113.137:21897']
piece index 0x00000000, contributors ['5.12.113.137:21897']
piece index 0x0000000a, contributors ['5.12.113.137:21897']
piece index 0x00000006, contributors ['5.12.113.137:21897']
piece index 0x00000003, contributors ['5.12.113.137:21897']
piece index 0x00000001, contributors ['5.12.113.137:21897']
piece index 0x00000002, contributors ['5.12.113.137:21897']
piece index 0x00000005, contributors ['5.12.113.137:21897']
piece index 0x00000004, contributors ['5.12.113.137:21897']
```

As seen in Figure 3.4, the information matches the output for one of the files. The other file had no Piece messages in the communication, meaning the capture was corrupted, the communication was not detected or the download used an extended functionality, not supported by this tool. The absence of Piece messages was confirmed by manual capture analysis.



```
bittorrent.msg.type == 7
Source         Source Port  Info
5.12.113.137   21897        Piece, Idx:0x7,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x7,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x8,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x8,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0xb,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0xb,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x9,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x9,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0xd,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0xd,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0xc,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0xc,Begin:0x4000,Len:0x4000
                            Piece, Idx:0xe,Begin:0x0,Len:0x1027
5.12.113.137   21897        Piece, Idx:0x0,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x0,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0xa,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0xa,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x6,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x6,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x3,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x3,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x1,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x1,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x2,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x2,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x5,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x5,Begin:0x4000,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x4,Begin:0x0,Len:0x4000
5.12.113.137   21897        Piece, Idx:0x4,Begin:0x4000,Len:0x4000
```

Figure 3.4: Pieces of a currently downloaded file.

Chapter 4

Discussion and conclusion

The BitTorrent research is demanding due to the large amount of various information, including unofficial extensions. The communication does not always adhere to rules given in standards and differs quite significantly between client implementations, therefore detection and dissection can be difficult or even impossible. In this project, some of the possible methods were proposed and later used to implement a detection tool. The research was mostly based on analysis of the captured communication from the qBittorrent client, which, with some exceptions, corresponds with the general description of the BitTorrent architecture. In most cases, the tool successfully processes the given captures and shows basic information to the user. The BitTorrent architecture is an interesting concept that is not as documented as some other technologies and utilizes some mathematical concepts, therefore a lot of time and dedication is required to fully understand all of the processes.

Bibliography

[1] Cohen, B. *The BitTorrent Protocol Specification*. 2008. Available at: https://www.bittorrent.org/beps/bep_0003.html.

[2] Loewenstern, A. and Norberg, A. *DHT Protocol*. 2008. Available at: https://www.bittorrent.org/beps/bep_0005.html.

[3] Matoušek, P. *Identifikace síťových protokolů a detekce anomálií*. Brno University of Technology, 2023.

[4] Maymounkov, P. and Mazières, D. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Druschel, P., Kaashoek, F. and Rowstron, A., ed. *Peer-to-Peer Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, p. 53–65. ISBN 978-3-540-45748-0.

[5] Norberg, A. *DHT routing table maintenance*. Available at: https://blog.libtorrent.org/2014/11/dht-routing-table-maintenance/.

[6] Widman, K. *How to Write a Bittorrent Client – Part 2*. 2012. Available at: http://www.kristenwidman.com/blog/71/how-to-write-a-bittorrent-client-part-2/.