UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Torrent client using Go programming language

Diploma Thesis

John Eliades

Supervisor: Thanos Georgios

Volos 2021

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Torrent client using Go programming language

Diploma Thesis

John Eliades

Supervisor: Thanos Georgios

Volos 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Torrent client με χρήση της γλώσσας Go

Διπλωματική Εργασία

Γιάννης Ηλιάδης

Επιβλέπων: Θάνος Γεώργιος

Βόλος 2021

Approved by the Examination Committee:

Supervisor	**Thanos Georgios** Laboratory Teaching Staff, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Korakis Athanasios** Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Fevgas Athanasios** Laboratory Teaching Staff, Department of Electrical and Computer Engineering, University of Thessaly

Στους γονείς μου, Γεώργιο και Κωνσταντίνα.

Acknowledgements

First and foremost, I have to start by thanking Prof. George Thanos for the supervision, collaboration and continuous support at every stage of this diploma thesis, as well as the co-supervisors Prof. Korakis Athanasios and Prof. Fevgas Athanasios. In addition, I would like to thank my parents, Georgios and Konstantina, who have always been there for me by providing help, support and motivation unconditionally. Finally, I would like to thank all of my professors for the guidance and my friends and colleagues of University of Thessaly for the unforgettable moments.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Abstract

Nowadays software that uses peer-to-peer(P2P) networking is becoming more and more popular. Peer-to-peer is a distributed system that has many advantages compared to the classic server-client approach because each peer acts like a server. Some of them are the quick sharing of large files, the reduced costs as there is no need for a central server, more flexibility as it is easier for new peers to join and more reliability due to the fact that each peer contributes resources to the network.

Currently the most widespread protocol for file sharing is the BitTorrent protocol. Because of its simplicity, there are many implementations available in many languages, most of them using event-driven programming. Event-driven programming is usually a good candidate as there is no need for synchronization making it easier for programmers and avoiding the overhead of the communication between the threads. In this thesis we developed flash torrent, a BitTorrent client aiming to to be as parallel as possible, to fully utilize a computer's resources. The implementation is written in one of the most suitable programming languages for this job, the Go programming language as it features goroutines which are lightweight threads.

Keywords : peer-to-peer, P2P, BitTorrent

Περίληψη

Στις μέρες μας, το λογισμικό που χρησιμοποιεί P2P γίνεται όλο και πιο δημοφιλές. Ένα δίκτυο υπολογιστών peer-to-peer (ή P2P) έχει πολλά πλεονεκτήματα συγκριτικά με την απλή προσέγγιση server-client επειδή κάθε χρήστης συμπεριφέρεται και σαν server. Κάποια από αυτά είναι η γρήγορη κοινή χρήση μεγάλων αρχείων, το μειωμένο κόστος μιας και δεν υπάρχει ανάγκη πλέον για κεντρικούς servers, η ευελιξία που παρέχει μιας και είναι πολύ εύκολο να προστεθούν καινούργιοι χρήστες και η αξιοπιστία λόγω του ότι κάθε χρήστης συνεισφέρει στην διαδικασία.

Επί του παρόντος, το πιο διαδεδομένο πρωτόκολλο για διαμοιρασμό αρχείων είναι το BitTorrent όπου λόγω της απλότητας του υπάρχουν αρκετές υλοποιήσεις διαθέσιμες σε πολλές γλώσσες, με τις περισσότερες να βασίζονται σε γεγονότα(event-driven). Αυτή η προσέγγιση έχει τα οφέλη ότι δεν υπάρχει ανάγκη για συγχρονισμό καθιστώντας την μια καλή επιλογή για τους προγραμματιστές, και ότι αποφεύγει το κόστος επικοινωνίας μεταξύ των thread. Σε αυτή την διπλωματική αναπτύχθηκε ο flash torrent, ένας client που στοχεύει στο να είναι όσο πιο παράλληλος γίνεται, για αυτό τον λόγο έχει επιλεχθεί η γλώσσα προγραμματισμού Go, η οποία έχει τις goroutines που είναι threads κατάλληλα για αυτή την δουλειά.

Table of contents

List of figures

Abbreviations

BT	BitTorrent
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
FTP	File Transfer Protocol
P2P	Peer-to-Peer
DHT	Distributed Hash Table
CLI	Command-line Interface
GUI	Graphical User interface
ETA	Estimated Time of Arrival
BDP	Bandwidth-delay-product
KB	Kilobytes
MB	Megabytes
TB	Terabytes
FIFO	First in, first out
CGI	Common Gateway Interface
PID	Process ID
URL	Uniform Resource Locator
NAT	Network Address Translation
IP	Internet Protocol
DNS	Domain Name System
CSP	Communicating sequential processes
OS	Operating System(s)
ASCII	American Standard Code For Information Interchange
PEX	Peer Exchange

RSS	Really Simple Syndication
UTP	uTorrent Transport Protocol

Chapter 1

Introduction

1.1 History of P2P Networks

Peer-to-peer networks were utilized in the past for many purposes. They allowed millions of people to connect with one another through the internet and form teams, which worked together to construct search engines, file systems, and virtual supercomputers. However P2P networking gained popularity, because of file-sharing apps like the Napster software, which was first published in 1999. Bram Cohen, the developer of the BitTorrent protocol, was employed by the MojoNation business. Users of MojoNation were allowed to divide confidential files into smaller encrypted parts and share them with other users of the same program. If someone desired this file, they could get it from a number of people at the same time. On the basis of this concept, Bram Cohen, inspired by other applications of the day, such as KaZaA, which took a long time to download a large file since it generally came from a source, created the BitTorrent protocol. He considered that such a strategy is ideal for huge files since the user will be able to get the file from various peers, including those who already have the file or parts of the file. As a result, the more popular a file is, the more peers it will have, resulting in a significant reduction in download time. The BitTorrent protocol was conceived in April 2001 by Bram Cohen and he released his first version on July 2, 2001. BitTorrent was also called the software that used the protocol. The moment it launched, BitTorrent became the most important protocol for transfering big files. It was calculated that those networks, on February 2009, were related to 43% to 70% of the total traffic data on the internet, depending on the region. In February 2013, the BitTorrent was estimated to be responsible for 3.35% of the global bandwidth. Then, on 2018, Cohen created a corporation named Chia Network that later deployed Chia, a new cryptocurrency based on HDD's.

1.2 Why P2P?

The core idea of a peer-to-peer network, contrasting the traditional server/client relationship, in which those downloading connect to a central server, is that those participating in the BitTorrent system, the peers, send pieces of files to one another. Thus making it a peer-to-peer protocol. This core difference means that every peer that is added not only doesn't stress the network but on the contrary the content-serving capacity increases with each user. That is what makes P2P networks the best candidate for file sharing.

1.3 Flash Torrent

In this thesis we present flash torrent, a client implementation of the widely known P2P protocol BT that is currently used for file sharing. It is implemented in golang a language that features goroutines which is a lightweight thread different than the threads in other programming languages. The advantages of these lightweight threads are that they are not hardware dependent, they have an easy communication medium known as channel that helps one goroutine communicate with another with low latency and they have faster startup times than threads. The BitTorrent protocol requires that many TCP connections are kept open at the same time so that a peer can communicate with the others, making golang the ideal candidate for this project. We will attempt to create a concurrent torrent client with a language made for this job although this approach is not usually preferred for clients, proving it can be done without affecting performance. [1]

1.4 Content Organization

The following sections of this thesis are organised as shown:

1. In the 2nd chapter we provide an introduction of peer to peer networks.
2. In the 3rd chapter we present a top level view of the BitTorrent protocol and we describe the distinct components that are involved.
3. In the 4th chapter we present the algorithms that flash utilizes.
4. In the 5th chapter we explain the Go concurrency model.
5. In the 6th chapter the CLI application is shortly presented.
6. In the 7th chapter a summary of the conclusions is provided.

Chapter 2

Concepts and Tools

2.1 Introduction

Before going into detail on how the BitTorrent protocol works, an overview of the essential components on which it is based, as well as an introduction to the protocol and the fundamental principles that make it work, will be provided.

2.2 Peer-to-Peer

A P2P network is created based on the notion that nodes act as both ”servers” and ”clients” to the others in the network at the same time. This network configuration is different from the more common server-client approach, in which the communication is done using a central server as seen in 2.1. The FTP service, which has separate server and client applications, is an example of a transfer that employs the server–client model. The clients begin the transfer, and the servers fulfill these requests. On a P2P network, a communication session can be initiated by any of the participants' peers. The objective of this network is to achieve a goal by sharing resources such as processing power, storage, and bandwidth. Each participant can be both a consumer and a supplier, in the sense that it can contribute its resources while also requesting and using those of the other participants. [2]

Peer-to-peer networks are being used in a variety of applications, including file sharing, video and music streaming, cryptocurrencies, search engines, and more. Peer-to-peer networks are characterized by the ability to link people from all over the world without being restricted or censored, allowing for the free flow of information and files. [3] [4]

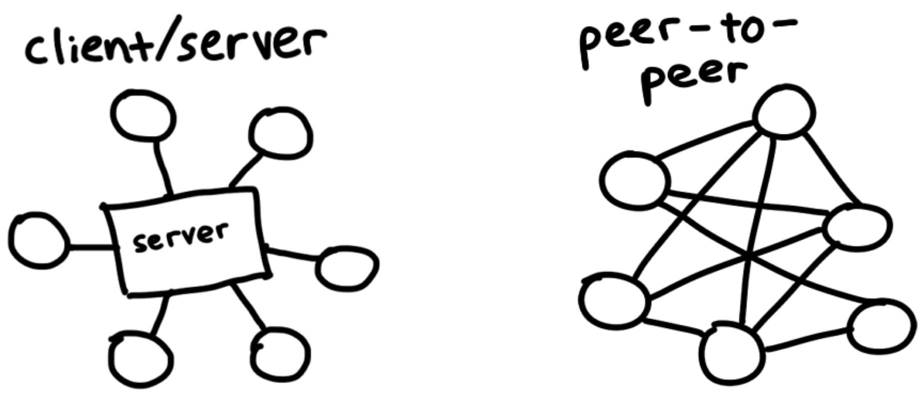


Figure 2.1: Client-Server vs P2P

Peer-to-peer networks are frequently used to handle large files. The BitTorrent protocol is the most widely used method. Many businesses and enterprises choose peer-to-peer networks because they provide numerous benefits such as:

- Easy file sharing: A peer-to-peer network can distribute files efficiently and fast even in great distances.
- Reduced cost: You either don't need money for a server, or it's considerably less expensive than a typical client-server type server.
- Adaptability: A new peer can join the network with ease. Peer-to-peer networks are more dynamic than client-server networks because of their flexibility.
- Reliability: In a peer-to-peer network, if one peer fails and opts out, the network will most likely continue to function.
- High performance: The more clients who join the peer-to-peer network, the more efficient it becomes, because each client shares the file. In a client-server network, on the other hand, as more individuals join clients, performance suffers.

2.2.1 Peer-to-Peer Topologies

The are three distinct classes of P2P networks:

- Structured networks(2.2): Also known as "first generation P2P networks," they use an Index Server as a central entity to store information about the contents of lists that members want to exchange. Users connect to the Index Server through a client and search for the files they're looking for. The Index Server sends the details of peers who have the file after locating it. Client selects a peer with whom he wishes to collaborate and establishes a connection with him in order to transmit the file. [2]

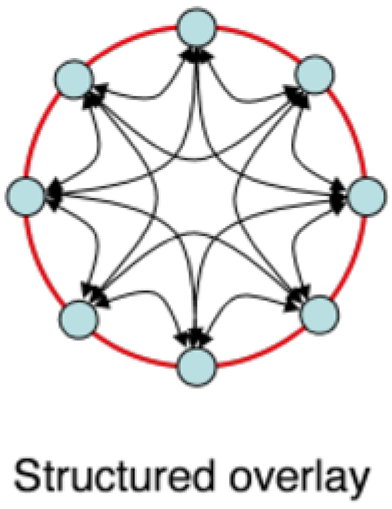


Figure 2.2: Structured network topology

- Unstructured networks(2.3): The philosophy here is completely different. Every system that takes part is a server and a client at the same time. When someone logs in using a similar P2P client, it advertises their existence to a small group of previously connected computers, which then forwards their presence to a wider computer network, and so on. The user may now search amongst shared files for any information. These networks are also called second generation. File transfer resembles that of centralized peer-to-peer (P2P) networks. [2]

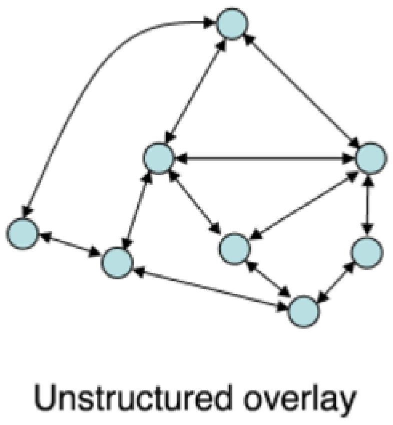


Figure 2.3: Unstructured network topology

- Hybrid networks(2.4): Peer-to-peer and client–server approaches are combined in hybrid models. A popular hybrid approach has a central server that assists peers in locating one another. Until 2014, Spotify was a popular model of this type. Many hybrid models exist, all of which try to balance between the functionality of a structured network and the node equality found in pure peer-to-peer unstructured networks. The BitTorrent protocol is implemented over this type of network. [2]

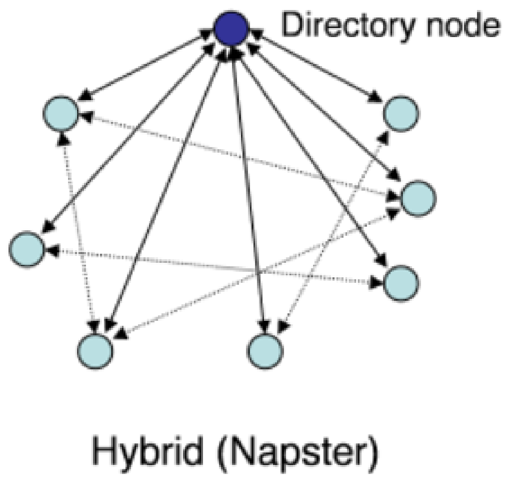


Figure 2.4: Hybrid network topology

Chapter 3

BitTorrent

3.1 BitTorrent

BT is a P2P sharing communication protocol being used for data and file transmission over the Web. It is mainly used to lessen the influence of big file distribution on servers and networks. Instead of downloading data from one server, the protocol lets users to connect to many hosts and send and receive files from them. It is a replacement for the former single or multiple source method of data distribution, and networks with lower bandwidth can benefit from it. Common machines, such as home computers, replace huge servers while distributing content more efficiently to multiple receivers using the BitTorrent protocol. BitTorrent files are split into pieces, and each of them is broken down into blocks that are then transmitted.

3.2 Components

Typically, the BitTorrent architecture consists of the following components.

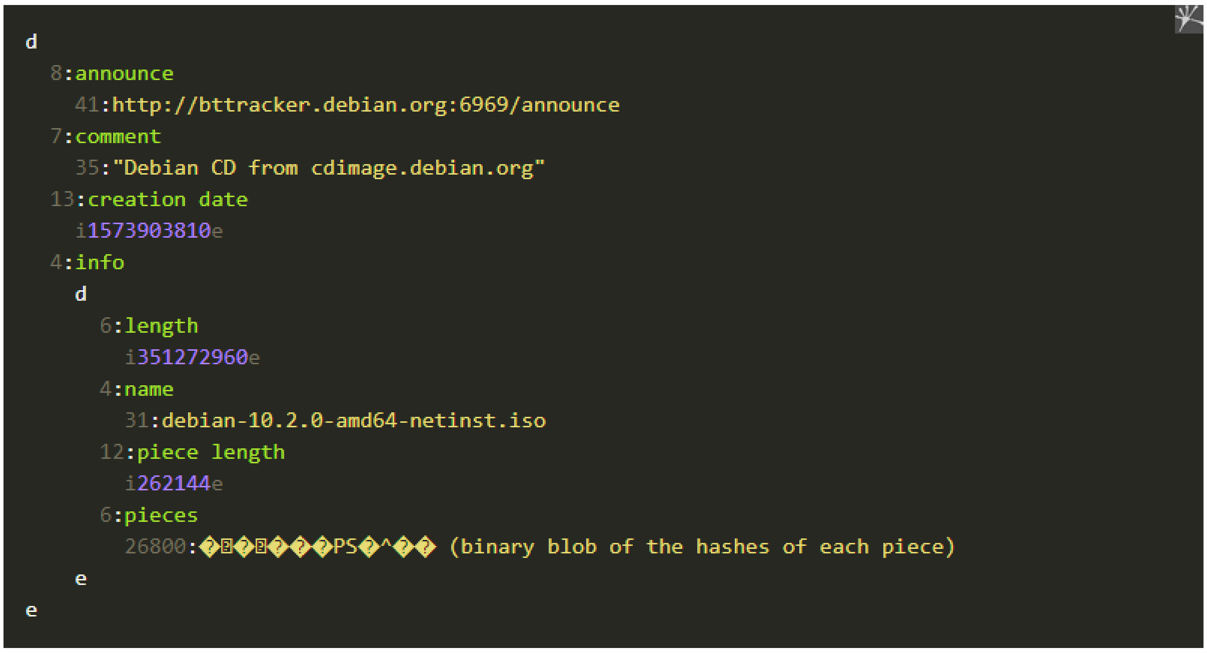
3.2.1 Metainfo File

In the BT sharing platform, a metainfo file, commonly known as a torrent, includes details about the directory structure to be transmitted and it is specific to the content being shared and holds all of the information about a download. This information is extracted by a client from this file, which has the ’.torrent’ suffix. The material to be transmitted is not included in a torrent file. Instead, it contains the information described below. The information that exists in this file are encoded using a method specific in bittorrent known as ‘bencoding’.

3.2.1.1 Metainfo File Structure

Everything stored in this file is a dictionary, while the characters are encoded using UTF-8. It consists of these keys: [5]

Here is how the prettified bencoded torrent file of debian looks 3.1:



```
d
  8:announce
    41:http://bttracker.debian.org:6969/announce
  7:comment
    35:"Debian CD from cdimage.debian.org"
  13:creation date
    i1573903810e
  4:info
    d
      6:length
        i351272960e
      4:name
        31:debian-10.2.0-amd64-netinst.iso
      12:piece length
        i262144e
      6:pieces
        26800:�����������PS�^�� (binary blob of the hashes of each piece)
    e
e
```

Figure 3.1: Prettified bencoded torrent file

Here, we can get the tracker's URL, the creation date, the file's name and length, and the SHA-1 binary hashes of every piece.

Note: the underlined values are the mandatory ones.

- <u>announce</u>: The URL used for tracker communication. (string)
- announce-list: A list of URLs used to communicate with the trackers. This exists in order to make the protocol backwards-compatible as now DHT and PEX are used. (list of lists of strings)
- creation date: this torrent's creation time in POSIX time (seconds counting from 1-Jan-1970 00:00:00 UTC and after)
- comment: any comment the creator wants to convey. (string)
- created by: The information of the application that created the torrent (string)

- encoding: the encoding which was used in order to encode the pieces part in the dictionary. (string)
- info: a dictionary describing the structure of the files within the torrent. It may be a single file torrent or a multiple file torrent with directories.

  One file mode:

  - name: The file's name. Could be ignored. (string)
  - length: the file's size calculated in bytes. (integer)
  - md5sum: The MD5 sum consisting of a string of 32 characters.

  Multiple file mode:

  - name: the parent directory name which will contain all of the directories. This is only for informational purposes. (string)
  - files: a dictionary collection, each for a different file The following keys are found in each dictionary on this list:
    - length: the file's size counted in bytes. (integer)
    - path: The total file's path described as a list of strings. The last string corresponds to the file name while the rest of the strings are the path leading to it. The file "directory1/directory2/example," for example, would have three elements: "directory1," "directory2," and "example." l10:directory110:directory27:example is an example of the bencoded list of strings.
    - md5sum: The MD5 sum consisting of a string of 32 characters.

  Common fields:

  - piece length: each piece's length counted in bytes. (integer)
  - pieces: a string that consists of the concatenation of each piece's SHA1 hash value(20 bytes each). (string of bytes)
  - private: if this int is set to '1', then the client must only find peers using the trackers described in the torrent file. On the other hand if set to '0' or doesn't exist, the client may find peers using methods like: PEX peer exchange or DHT. "private" therefore means "no external peer source".

3.2.1.2 Bencoding Structure

Bencoding is an encoding that is less human-readable than JSON while also being able to encode the same types but more efficiently and with great ease on streams. The starting delimiters are ‘i’ for integers, ‘l’ for lists, and ‘d’ for dictionaries and the ending delimiter is ‘e’. Byte strings do not use delimeters. [5]

- Byte Strings : <length of string>:<string>
- Integers: i<integer>e.
- Lists: : l<list elements in string format>e
- Dictionaries: d<string><element>e

Some examples of how bencoding looks like:

- 4:test // is the string “test”
- i4e // is the integer “4”
- l5:test15:test25:test3e // are the entries test1, test2, test3
- d3:keyl6:value16:value2ee // represents the dictionary “key” => [“value1” , “value2”]

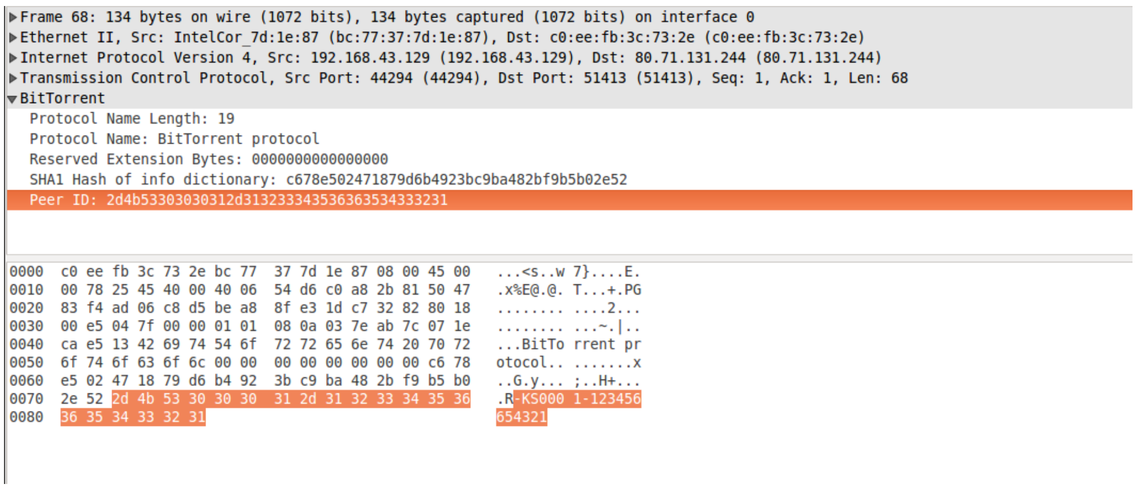
3.2.2 Peer

A peer is generally a BitTorrent client user who wants to join a swarm to transfer data. A peer can be either a leecher or a seeder. The leecher state occurs while the peer is still getting data and has not yet completed the download, and the seeder state occurs when the peer has. TCP is used in order for the peers to talk to each other and transmit data. Because TCP ensures the reliable data transmission of all the packets, it is preferred over alternative protocols such as UDP (User Datagram Protocol). UDP cannot provide such assurances, as data may be jumbled or lost entirely. Peers can connect with some trackers through HTTP using plain text (Hypertext Transfer Protocol) or with others using UDP messages and can use them to find other peers that have the data and start communicating with them. This happens after a handshake. The way handshake takes place is described from now on. The handshake starts with the character 19 and ends with the text ”BitTorrent Protocol.” The info value from the metainfo is then delivered as a 20-byte bencoded SHA1 hash. The connection is terminated if this does not match between peers. A 20-byte peer identifier is provided, which is subsequently utilized in tracker and peer requests. The connection is terminated if the wrong peer id is received compared to the expected one.

3.2.2.1 Handshake

The first message sent by the client must be the handshake. It is (49+len(pstr)) bytes. The handshake packet looks like this: 3.2 [5]

- pstrlen: the length of the string as one byte.
- pstr: the protocol identified using a string.
- reserved: Eigth bytes of zeroes. Each bit is reserved so that the protocol's behaviour can be altered.
- info_hash: The value used in tracker requests.
- peer_id: The value sent in tracker requests, meaning a 20 byte string id that identifies each unique peer.



```
▶Frame 68: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface 0
▶Ethernet II, Src: IntelCor_7d:1e:87 (bc:77:37:7d:1e:87), Dst: c0:ee:fb:3c:73:2e (c0:ee:fb:3c:73:2e)
▶Internet Protocol Version 4, Src: 192.168.43.129 (192.168.43.129), Dst: 80.71.131.244 (80.71.131.244)
▶Transmission Control Protocol, Src Port: 44294 (44294), Dst Port: 51413 (51413), Seq: 1, Ack: 1, Len: 68
▼BitTorrent
  Protocol Name Length: 19
  Protocol Name: BitTorrent protocol
  Reserved Extension Bytes: 0000000000000000
  SHA1 Hash of info dictionary: c678e502471879d6b4923bc9ba482bf9b5b02e52
  Peer ID: 2d4b53303030312d313233343536363534333231

0000  c0 ee fb 3c 73 2e bc 77  37 7d 1e 87 08 00 45 00   ...<s..w 7}....E.
0010  00 78 25 45 40 00 40 06  54 d6 c0 a8 2b 81 50 47   .x%E@.@. T...+.PG
0020  83 f4 ad 06 c8 d5 be a8  8f e3 1d c7 32 82 80 18   ........ ....2...
0030  00 e5 04 7f 00 00 01 01  08 0a 03 7e ab 7c 07 1e   ........ ...~.|..
0040  ca e5 13 42 69 74 54 6f  72 72 65 6e 74 20 70 72   ...BitTo rrent pr
0050  6f 74 6f 63 6f 6c 00 00  00 00 00 00 00 00 c6 78   otocol.. .......x
0060  e5 02 47 18 79 d6 b4 92  3b c9 ba 48 2b f9 b5 b0   ..G.y... ;..H+...
0070  2e 52 2d 4b 53 30 30 30  31 2d 31 32 33 34 35 36   .R-KS000 1-123456
0080  36 35 34 33 32 31                                  654321
```

Figure 3.2: Peer handshake

A connection's initiator is expected to send their handshake right away. If the recipient is capable of providing several torrents at the same time, it may wait for the initiator's handshake (each infohash is unique representing a different torrent). The recipient must reply the moment it gets the handshake and the connection must be immediately terminated if a client receives a non matching handshake. Furthermore, if the one that connects gets a handshake with a peer id that differs from the expected, the connection is dropped. That is because the tracker's peer id and the handshake's peer id must be the same.

3.2.2.2 Messages

After the handshake the peers can communicate using the following messages described loosely that look like this 3.3: [5]

- choke : This prevents the other peers from requesting data.
- unchoke: The block is lifted allowing the peer to start downloading from you again.
- interested: If the required data exist the the peer is interested.
- not interested: The required data don't exist in the peer.
- have: Asks for the pieces the peer owns.
- bitfield: First message after handshake. The payload includes the pieces that are already downloaded and therefore served.
- request: A request for a piece providing the index, the begin location and length.
- piece: Sent together with request messages. The payload contains the same integer values.
- cancel: payload is the same as 'request' and is used to cancel already obtained blocks.

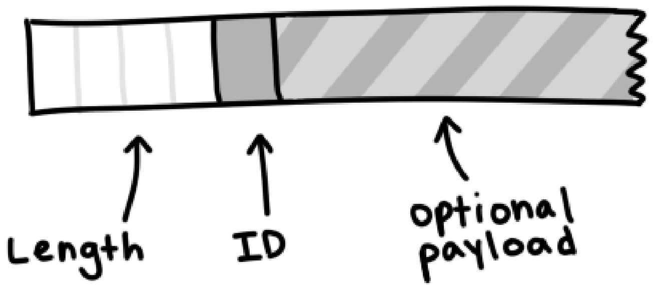
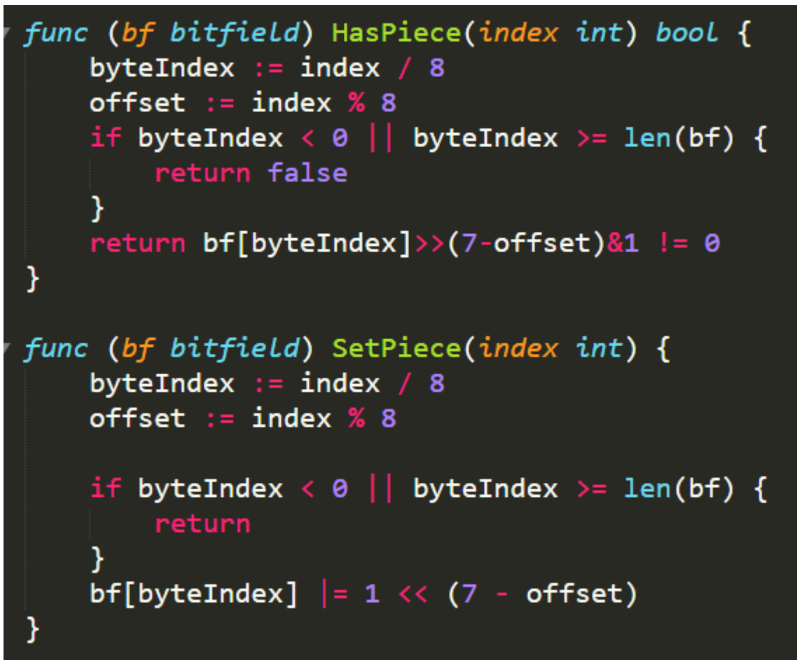


Figure 3.3: Message structure

After the client and server have exchanged handshakes, the peer generally sends a "bit-field" message, indicating which parts of the file are complete. It may also send a succession of ”have” messages, one for each piece he possesses. The client analyses this information after storing it in a bitfield, and he discovers which parts he is lacking. Each byte in a bitfield can be thought as eight individual bits indicating which pieces a peer has (i.e. 1111110111111111 (the pieces from 0 to 15, with piece 6 missing). The next stage is for him to send an interested message in which he expresses his desire to take blocks. If the peer answers with an ”unchoke” message, the client can send a request message which will request a specific block, and the peer will reply with a piece message, which will include the block requested. The peer can send the message ”choke” at any time during the conversation, and all requests received from that moment will be ignored. After the download is complete, the client can send cancellation messages to any peers who have submitted requests, informing them that the client no longer requires the blocks requested. These messages are sent and received in both directions.

In figure 3.4 we get a look at the bitwise operations that search or set a specific piece in the bytes array of the bitfield.



```go
func (bf bitfield) HasPiece(index int) bool {
    byteIndex := index / 8
    offset := index % 8
    if byteIndex < 0 || byteIndex >= len(bf) {
        return false
    }
    return bf[byteIndex]>>(7-offset)&1 != 0
}

func (bf bitfield) SetPiece(index int) {
    byteIndex := index / 8
    offset := index % 8

    if byteIndex < 0 || byteIndex >= len(bf) {
        return
    }
    bf[byteIndex] |= 1 << (7 - offset)
}
```

Figure 3.4: Bitfield manipulation

3.2.2.3 Connection State Information

Each remote peer must have state information saved for the other remote peers. [5]

1. choked: whether or not this client is blocked by the peer for requesting data(3.5). When

a client is blocked, it means that the requests will be processed only after the client is unblocked. Then it must wait before submitting requests for blocks, and it should assume that the remote peer would reject any pending (unanswered) requests. Choking is performed for a variety of reasons. When transmitting data over a large number of connections at once, TCP congestion management performs badly. Choking also allows each peer to employ an ”eye for an eye” strategy to guarantee an uninterrupted download.

2. interested: If the peer wants the services provided by this client. When the client unchokes the remote peer, this signals that block requests will start arriving.

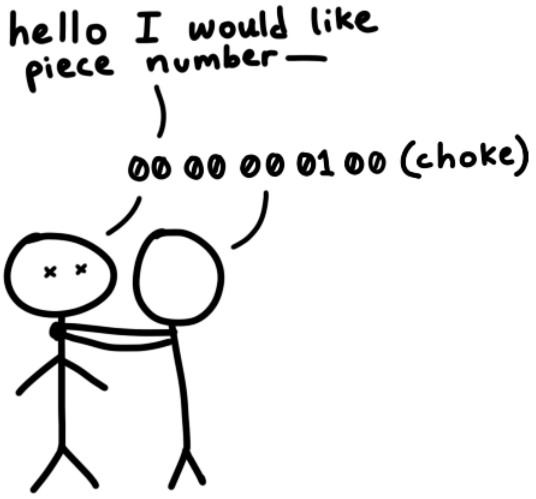


Figure 3.5: Choke message

3.2.3 Data

BitTorrent is extremely flexible, and the data transfer may be a single file, numerous files of any sort, or several directories. The size of a file can range from a few KB’s to many TB’s.

The information is broken down into small chunks of the same size and transmitted between peers through the BitTorrent protocol. This also divides the file into chunks that can be easily verified, each of which may be issued a hash code, which the downloader can check for data integrity. These hashes are saved in the ’metainfo file.

With the exception of the final piece, which is irregular, the piece sizes are the same throughout every torrent’s file. The quantity of data determines the piece size a torrent is assigned. When downloading, greater piece sizes result in inefficiency (bigger pieces have a higher chance of data corruption due to fewer integrity checks), whereas smaller parts require more hash checks.

As the number of pieces grows, more hashes are necessary. Therefore, pieces should be chosen so that the torrent file is less than 60 kb in size. The primary reason for this is to keep indexing servers' hosting storage and bandwidth usage to a minimum. 256kb, 512kb, and 1mb are the most frequent piece sizes. As a result, pieces are calculated using the following formula: length / pieceSize. There can be an overlap of pieces and file boundaries.

An example would be: a 1.4MB file can get broken down to 6 chucks. 5 of the pieces being 256KB, and the last being 120KB. (1.4Mb = 5 * 256 + 120 Kb)

3.2.4 Tracker

A tracker is a machine that helps in BitTorrent peer-to-peer communication. The most important role is to help peers 'find one another' and communicate. It maintains track of which peers own each file, and which are active when the client sends the request, and coordinates effective file transfer and reconstruction. Think of it as the local pub when trying to meet people. Trackers can be either TCP trackers that communicate with GET requests or UDP trackers [6]. Clients who started getting the file, can keep communicating with the tracker on a regular basis to request other peers in order to achieve faster data transmission with them. It is not required though, as after the file download has begun, P2P communication continues without the need for a tracker connection. In the BitTorrent network, a tracker is the sole centralized entity.

However, because the protocol promotes decentralization, a new technology called ”DHT” was developed. The peers that implement the DHT act as a tracker, storing other nodes that can be used in search of new peers. And another one called ”PEX” that allows peers to exchange their connected peers with each other.[7] [8] [9]

Note: In order to avoid ambiguity, we'll refer to the BitTorrent peer operating on the local system as 'client,' while every other peer will be 'peer.' Readers may see themselves as the client who connects with a large number of peers.

After getting a torrent from an online server, a client can join an existing torrent swarm. As previously indicated, the metainfo(torrent) file contains the trackers' urls, so the client attempts to request peers from them, and they ultimately answer with the IP's of the swarm peers. The client then establishes TCP connections with the received IP's in order to exchange data blocks with the remainder of the swarm.

3.2.4.1 Tracker Request

To begin downloading the file, the torrent client must first obtain a list of active peers from the tracker. The torrent client must first submit a GET request to the tracker in order to do this. The foundation of the URL to which the client will make the request may be found in either the ”announce” field or the ”announce-list” field of the metainfo file (.torrent). The relevant parameters are added after the basic URL, using conventional CGI techniques, which include a ”?” after the announce URL and parameters in the form ”param=value”, separated by a ”&”. [5]

- info_hash: The info_key found in the torrent, urlencoded as a SHA1 hash of 20 bytes.
- peer_id: a urlencoded string of 20 bytes produced by the client at startup and used as the client's unique ID. This can be anything, even binary data. The standard does not describe how to generate this peer ID for now. However, it is reasonable to assume that it at least should be one of a kind for your local computer, and hence should probably include things such as a PID and maybe a unique timestamp.
- port: The client's port number to which it is listening. The ports 6881-6889 are usually allocated for BitTorrent. If a client is unable to establish a port inside this range, it may opt to give up.
- uploaded: The amount of bytes that got uploaded since the tracker received the 'started' event despite the fact that it is not expressly specified in the official specification.
- downloaded: The total amount of data downloaded in base ten ASCII since the tracker received the 'started' event from the client. Despite the fact that it isn't mentioned clearly in the original specification, it is often assumed that this represents the total amount of bytes downloaded.
- left: The amount of bytes in base ten ASCII that this client still needs to download. Clarification: The total amount of bytes required to download the torrent in its entirety and receive all of the contained files.

- compact: When this value is set to 1, the client accepts a compact answer. A peers string having 6 bytes for each peer replaces the peers list. The host is the first four bytes, while the port is the final two bytes (again in network byte order) as shown here 3.6. Some trackers only accept compact answers (to save bandwidth) and will either deny requests that don't have a value of compact = 1 or just respond unless the request includes a value of compact = 0 (The request will be refused).

- no_peer_id: The peer id can be omitted by the tracker. If compact is enabled, this option is ignored.

- event: If a value is provided, it should be ”started”, ”completed”, or ”stopped”. If not provided, this will be sent regularly.

  - started: This value must be present in the first request.
  - stopped: If the client is gently quitting, then this information should be transmitted to the tracker.
  - completed: When the download is finished, it must be transmitted to the tracker. However, if the download had already reached a 100% completion when the client opened, it should not be delivered. This is most likely to help the tracker update the ”finished downloads” number based on this event only.

- ip: The client machine's actual IP. Note: In most cases, this is an unnecessary argument because a client's IP can be deduced from the HTTP request. The argument is required only when the IP from which the request originated is not the same as the client's. If the client communicates with the tracker using a proxy server, this will happen. It's also required when both sides are behind the same NAT.

- numwant: The number of peers that is preffered by the client. It is permissible for this value to be zero. If not specified, a default of 50 peers is used.

- key: A private unique identifier. Its purpose is to enable a client to confirm their identification in the event that the IP changes.

- trackerid: If it was included in a prior announcement, it should be set here.

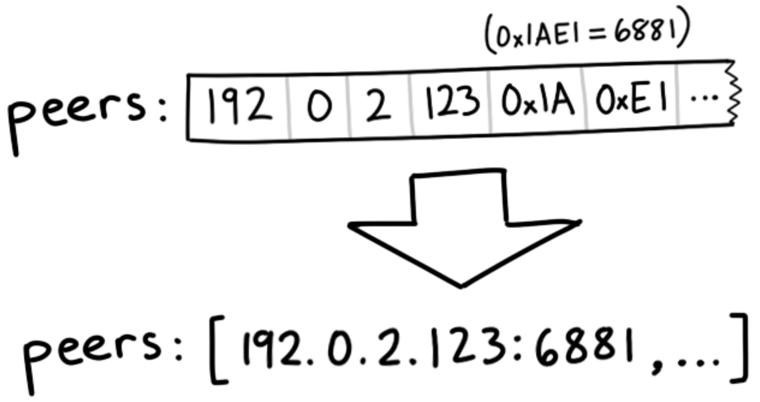

(0x1AE1 = 6881)
peers: 192 0 2 123 0x1A 0xE1 ...
peers: [192.0.2.123:6881,...]

Figure 3.6: String of compact answer

3.2.4.2 Tracker Response

After sending the first request to the tracker, the tracker sends a response, which is a bencoded dictionary. The following fields are included in the response: [5]

- failure reason: If a key is present, no further keys are possible. The value is an error message that explains why the request was unsuccessful (string).
- warning message: (optional, new) Although the answer is handled normally, it is similar to the failed reason. The warning message appears in the same way as an error message.
- interval: The time in seconds for waiting before communicating again with the tracker.
- min interval: The minumum interval for re-announcing.
- tracker id: Important string for each subsequent communication with the tracker.
- complete: the amount of peers that have the entirety of the file, known as seeders (integer)
- incomplete: the amount of ”leechers” (integer)
- peers: A dictionary list with keys:
  - peer id: a unique peer id. (string)
  - ip: the peer IP being a string containing IPv4, IPv6 or a DNS name
  - port: the open port the peer uses (integer)

3.2.5 Client

A BitTorrent client is a software that runs on your computer and implements the BitTorrent protocol. It interacts with the tracker and peers while running alongside the operating system on the user's computer. The client is in charge of reading and writing data, as well as opening sockets.

To participate in a torrent, the client must first open a metainfo file. After reading the file, the relevant data must be extracted, and a socket must be opened in order to communicate with the tracker. A client can handle several torrents at the same time.

Clients come in a variety of flavors, ranging from simple programs with few capabilities to highly complex, customizable applications. Metainfo file wizards and integrated trackers are two examples of sophisticated functionality. Because of these extra characteristics, various clients have different behaviours and may require several ports depending on the number of running processes. There are no incompatibility concerns because all apps implement the same protocol; nevertheless, due to individual modifications and enhancements across clients, a peer may enjoy greater download speeds from others using the same client. [1]

Chapter 4

Flash Algorithms

Some of the most common algorithms used in Flash are listed below. These methods were proposed by the BitTorrent founders and community [10]. They are not part of the original BitTorrent specification but they've been adopted by the majority of other BT clients, as well as Flash. [1]

4.1 Queueing

On each connection, the client maintains a few unfilled requests. This happens to avoid the round trip time that is necessary from the end of one block's download to the start of the next block's download. This might result in a significant performance loss on connections with a large BDP (bandwidth-delay-product).

4.2 Piece Selection

In contrast to what most torrent clients do, choosing to download the pieces in rarest first order, flash downloads pieces sequentially. This theoretically should reduce the performance but in practice this wasn't observed. The benefit of downloading pieces sequentially is that the client can use some of the first files before even the torrent is completely downloaded or even watch a movie stream while the next pieces are downloading. [11]

4.3 Strict Policy

When it comes to piece selection, Flash's first rule is that all sub-pieces of the first requested piece are downloaded together, followed by sub-pieces from other pieces. This is an effective method for obtaining full parts as fast as feasible.

Chapter 5

Go's Concurrent Architecture

5.1 Why Go?

Go, also called golang is a compiled, imperative, statically typed programming language developed by Google that includes built in memory safety, garbage collection, structural type, and CSP like [12] [13] concurrency that make it syntactically comparable to C. For developing concurrent applications, Golang provides features including a huge number of libraries. The goroutine, a kind of light-weight thread, is the fundamental concurrency construct. A new goroutine is launched when before the function call, the keyword "go" exists. Existing implementations, multiplex the goroutines of a Go process onto a smaller set of os threads although it is not clearly defined in the language specification. While there is an std package that includes the conventional concurrency structures like locks and mutexes, concurrent applications use channels, which allow messages to be shared from a goroutine to another. Buffers can also be used to store data in first in first out and enable transmitting goroutines to continue running nonblock. Channels are written in such a way that a channel can only transmit messages of its type T. To work with them, special syntax is needed; for example, <-ch forces the running goroutine to stop executing until the channel ch receives a value, whereas ch <- x transmits x. (potentially stopping its execution until the value is received by another goroutine). Non-blocking communication over many channels can be implemented using the select statement. Go includes a memory model that describes how goroutines must safely transfer data by using channels or other operations. [12] [13]

Apart from having great thread support, golang has a fixed style of coding unlike any other language. Many languages allow the programmer to program in any style. This results in having too many different styles thus making the code harder to read. Go avoids this by having a unified formatting style forcing it in all programs thus creating more readable code. This is accomplished by a tool named gofmt that takes all your packages and ensures they confront to the Go coding style.

Go also allows the same flexibility that C provides using the unsafe package. This package gives access to arbitrary pointer use letting you bypass the runtime and write in any address you want. Though the use of this package should be limited as much as possible.

Chapter 6

Flash Client

6.1 Architecture

First of all, the flash's main goroutine communicates with the trackers by spawning one goroutine for each of them. The trackers put the gathered peers on a channel and then all the peers are received from the main goroutine. Then it proceeds to spawn one goroutine for each of the peers and starts handshaking with them. The new goroutines store the state of their respective peer, whether they are chocked, unchocked, interested or not and their bitfields. Each of them then starts asking for unique pieces(a piece is never downloaded by 2 different goroutines simultaneously, and the same goroutine is responsible for downloading all the subpieces in this piece). If the piece download fails, then it is returned in a channel named workQueue for another goroutine to attempt getting it from their peer. When they have successfully downloaded a piece, it is sent to main using a channel named results as seen in figure 6.1 and it decides in which file the piece belongs. Then it gets instantly positioned correctly and written.

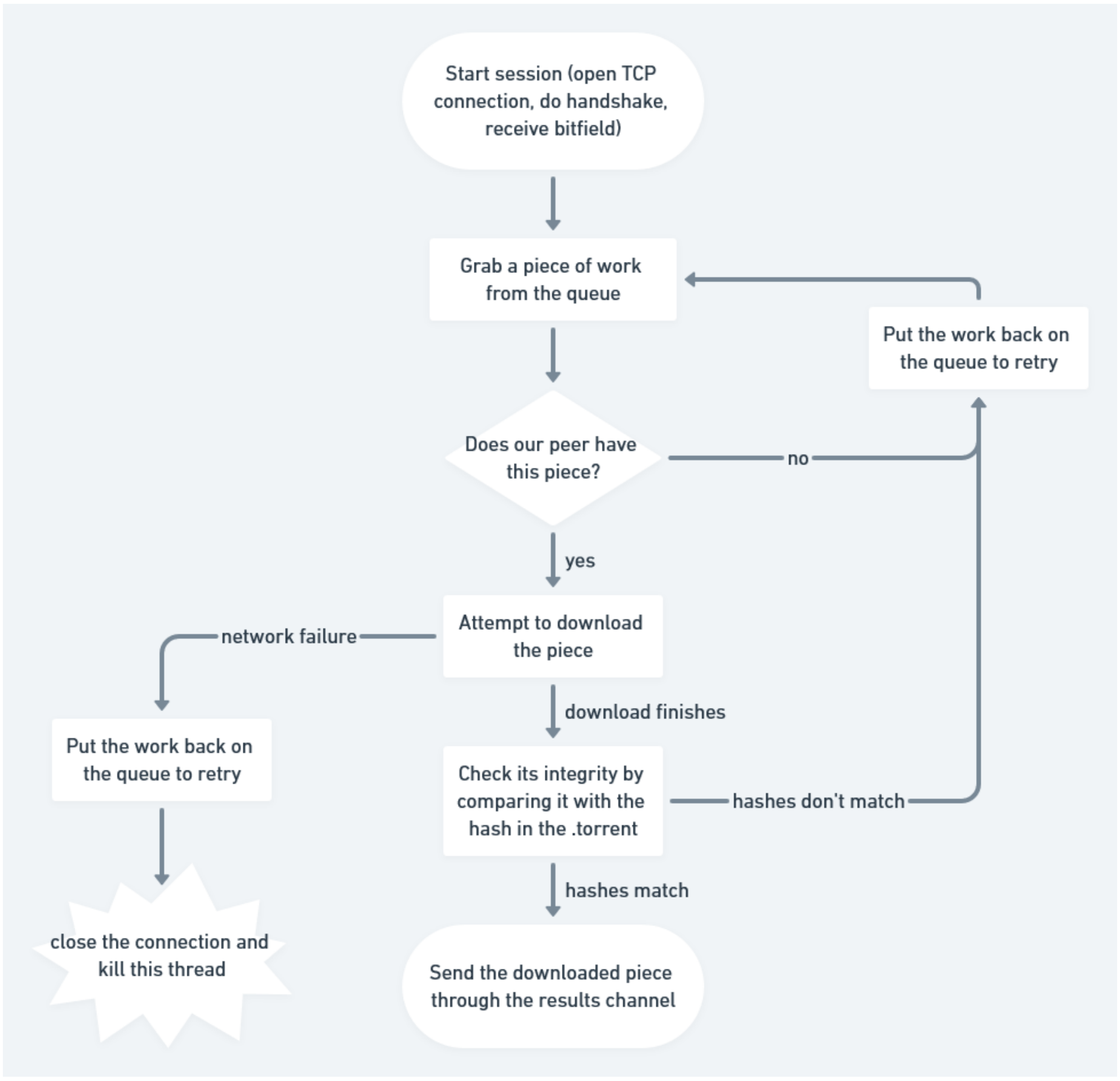
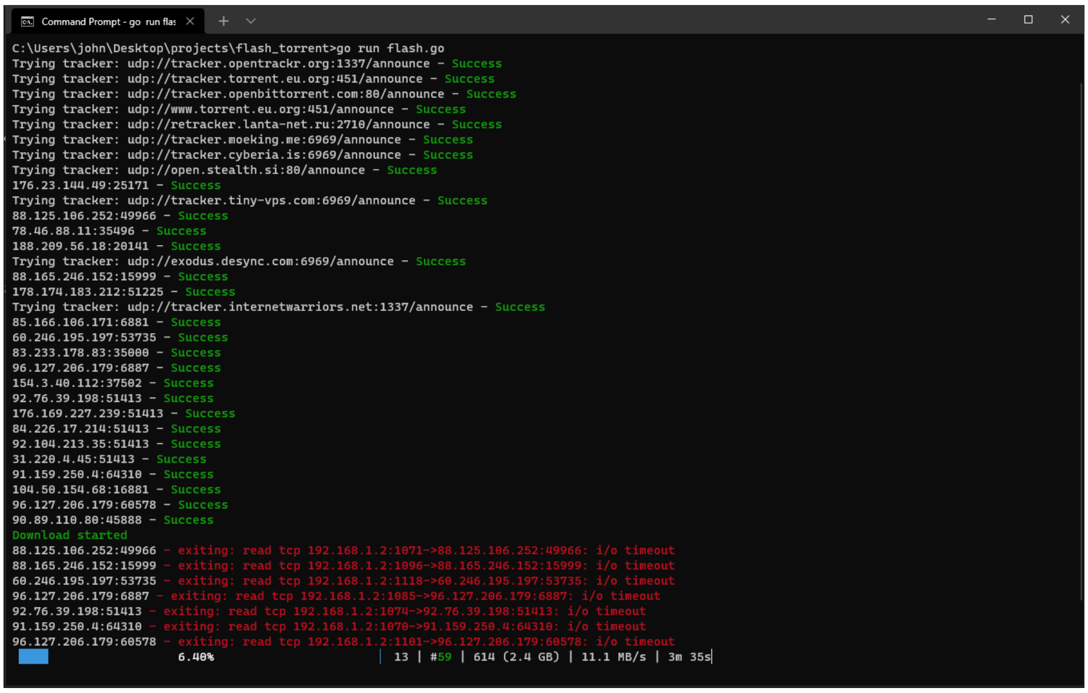

Start session (open TCP connection, do handshake, receive bitfield)
Grab a piece of work from the queue
Put the work back on the queue to retry
Does our peer have this piece?
no
yes
Attempt to download the piece
network failure
download finishes
Put the work back on the queue to retry
Check its integrity by comparing it with the hash in the .torrent
hashes don't match
hashes match
close the connection and kill this thread
Send the downloaded piece through the results channel

Figure 6.1: Work flow diagram

6.2 Client Preview

A CLI application has been built using the Flash library. In the first few lines of 6.2 we can see the url's of the trackers that were successfully connected giving us their list of peers to connect. Then we can see the peers' IP's that created a tcp connection successfully and started transmitting pieces. With red letters we can see which of their peers became inactive later and stopped sending data and below that we can see the download progress bar updating realtime.

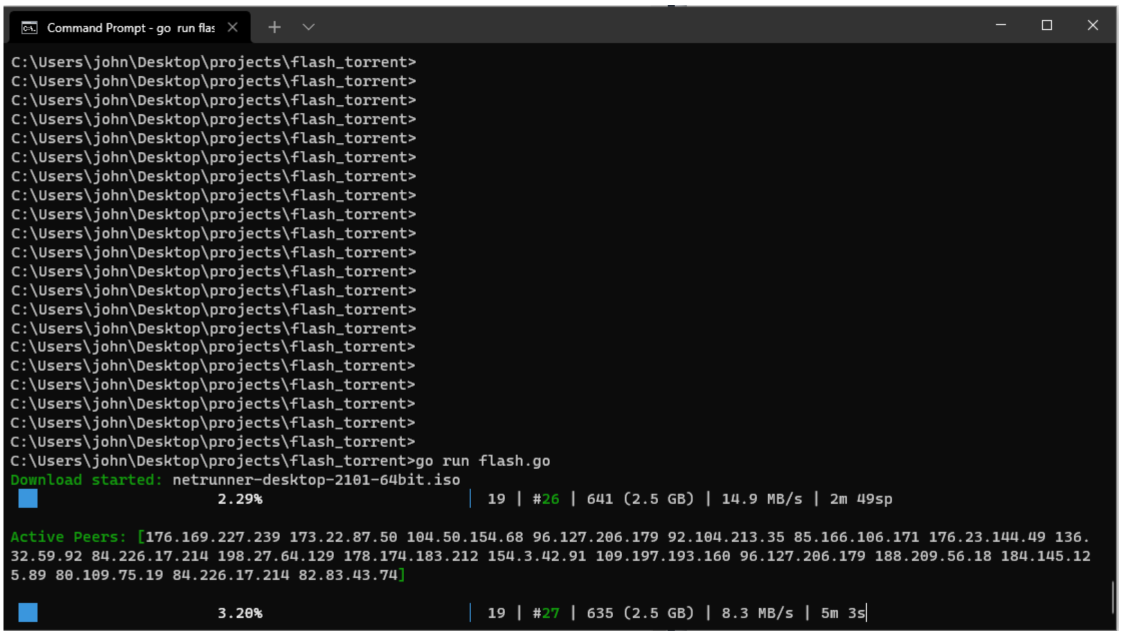
The first field near the progress bar shows how many peers are currently connected and active. The second shows the index of the latest downloaded piece. The third field shows the number of remaining pieces and their total size. The fourth is the download speed and the



```
C:\Users\john\Desktop\projects\flash_torrent>go run flash.go
Trying tracker: udp://tracker.opentrackr.org:1337/announce - Success
Trying tracker: udp://tracker.torrent.eu.org:451/announce - Success
Trying tracker: udp://tracker.openbittorrent.com:80/announce - Success
Trying tracker: udp://www.torrent.eu.org:451/announce - Success
Trying tracker: udp://retracker.lanta-net.ru:2710/announce - Success
Trying tracker: udp://tracker.moeking.me:6969/announce - Success
Trying tracker: udp://tracker.cyberia.is:6969/announce - Success
Trying tracker: udp://open.stealth.si:80/announce - Success
176.23.144.49:25171 - Success
Trying tracker: udp://tracker.tiny-vps.com:6969/announce - Success
88.125.106.252:49966 - Success
78.46.88.11:35496 - Success
188.209.56.18:20141 - Success
Trying tracker: udp://exodus.desync.com:6969/announce - Success
88.165.246.152:15999 - Success
178.174.183.212:51225 - Success
Trying tracker: udp://tracker.internetwarriors.net:1337/announce - Success
85.166.106.171:6881 - Success
60.246.195.197:53735 - Success
83.233.178.83:35000 - Success
96.127.206.179:6887 - Success
154.3.40.112:37502 - Success
92.76.39.198:51413 - Success
176.169.227.239:51413 - Success
84.226.17.214:51413 - Success
92.104.213.35:51413 - Success
31.220.4.45:51413 - Success
91.159.250.4:64310 - Success
104.50.154.68:16881 - Success
96.127.206.179:60578 - Success
90.89.110.80:45888 - Success
Download started
88.125.106.252:49966 - exiting: read tcp 192.168.1.2:1071->88.125.106.252:49966: i/o timeout
88.165.246.152:15999 - exiting: read tcp 192.168.1.2:1096->88.165.246.152:15999: i/o timeout
60.246.195.197:53735 - exiting: read tcp 192.168.1.2:1118->60.246.195.197:53735: i/o timeout
96.127.206.179:6887 - exiting: read tcp 192.168.1.2:1085->96.127.206.179:6887: i/o timeout
92.76.39.198:51413 - exiting: read tcp 192.168.1.2:1074->92.76.39.198:51413: i/o timeout
91.159.250.4:64310 - exiting: read tcp 192.168.1.2:1070->91.159.250.4:64310: i/o timeout
96.127.206.179:60578 - exiting: read tcp 192.168.1.2:1101->96.127.206.179:60578: i/o timeout
                      6.40%                   | 13 | #59 | 614 (2.4 GB) | 11.1 MB/s | 3m 35s
```

Figure 6.2: Flash client in action

last one is an eta(estimated time of arrival) of the complete file.



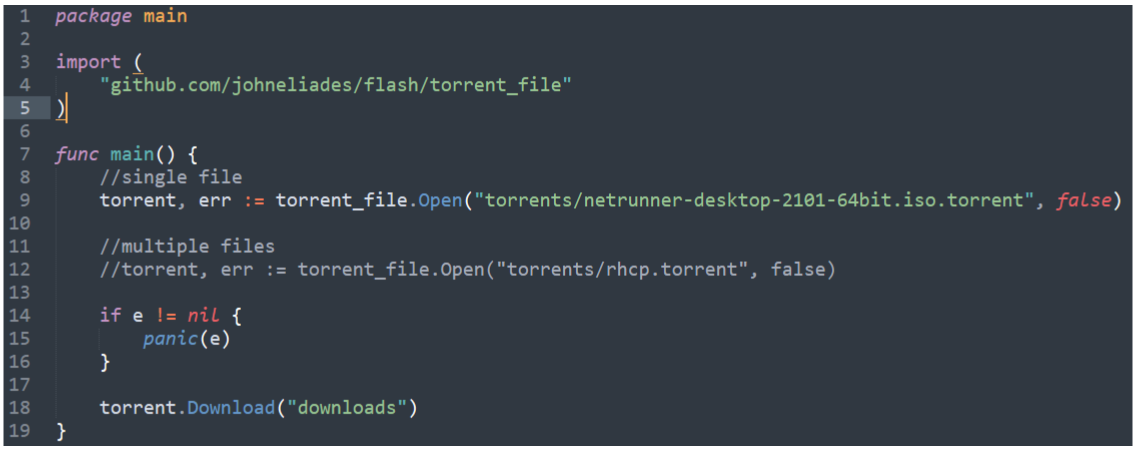
```
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>
C:\Users\john\Desktop\projects\flash_torrent>go run flash.go
Download started: netrunner-desktop-2101-64bit.iso
                      2.29%                   | 19 | #26 | 641 (2.5 GB) | 14.9 MB/s | 2m 49sp

Active Peers: [176.169.227.239 173.22.87.50 104.50.154.68 96.127.206.179 92.104.213.35 85.166.106.171 176.23.144.49 136.
32.59.92 84.226.17.214 198.27.64.129 178.174.183.212 154.3.42.91 109.197.193.160 96.127.206.179 188.209.56.18 184.145.12
5.89 80.109.75.19 84.226.17.214 82.83.43.74]

                      3.20%                   | 19 | #27 | 635 (2.5 GB) | 8.3 MB/s | 5m 3s
```

Figure 6.3: Active peers

Pressing ”p” shows a list of all the currently active peers of this torrent as shown in 6.3

6.3 Library

The flash torrent client can be found in this repository hosted on Github. There is a package named torrent_file that implements the protocol, that is specifically designed for simplicity and ease of use. It is as simple as opening a torrent and then downloading it as seen in 6.4:



```
1  package main
2  
3  import (
4      "github.com/johneliades/flash/torrent_file"
5  )
6  
7  func main() {
8      //single file
9      torrent, err := torrent_file.Open("torrents/netrunner-desktop-2101-64bit.iso.torrent", false)
10 
11     //multiple files
12     //torrent, err := torrent_file.Open("torrents/rhcp.torrent", false)
13 
14     if e != nil {
15         panic(e)
16     }
17 
18     torrent.Download("downloads")
19 }
```

Figure 6.4: The library

The false flag in line 9 means that no debugging messages will appear while downloading.

Chapter 7

Conclusion

7.1 Synopsis

This thesis examined the BitTorrent protocol and presented the theoretical foundations on which it is built. Peer-to-peer protocols, potential network topologies, and the BitTorrent functionality principle, as well as key components that allow for a thorough understanding of it, were all discussed. We looked at the information in the torrent file, the tracker protocol, which explains how a client gets the list of peers it requires from the tracker, as well as the fields in the tracker request messages and tracker response. After that, the peer protocol was mentioned that explains how the initial contact between two swarm participants starts and what messages are exchanged, as well as the potential states of a peer. Finally, algorithms for selecting the blocks were mentioned.

7.2 Future Work

Possible features that could be added later:

- GUI: A graphical user interface would make the flash torrent client more user friendly for managing many simultaneous torrents at once. Concurrent download of many torrents is already supported but the progress bars alternate making it hard to see the percentages.

- Seeding: Current implementation only leeches.

- Magnet Links: Would be easier to add torrents directly from the web browser instead of first downloading the ".torrent" file. [14]
- DHT(distributed-hash-table): Would help in finding more peers without using the tracker but by directly communicating with the peers that pretend to be a tracker. [14]
- PEX: Similary to DHT this is another method of obtaining IP addresses through peer exchange without relying on a tracker. This technique asks for the connected peers of the connected peers. PEX will frequently find better peers than DHT or a tracker because it uses the already active peers of your peers, despite the fact that it requires an initial help from a tracker. [14]

7.3 Conclusion

BitTorrent has quickly become the most popular peer-to-peer networking technology in just a few years. It's easy to use and has significant capabilities that allow for quicker file downloads and greater peer fairness than most standard peer-to-peer protocols. Based on widespread adoption and use by Internet users, it has shown to be a well-designed and effective file sharing protocol. However, it is far from perfect, and as a result, it has the potential to improve with time.

Originally, the protocol used to have a central tracker to allow peers to discover other peers, making it a hybrid peer-to-peer protocol. This central tracker has been eliminated in newer versions of the protocol, and the function has been dispersed to the peers themselves. BitTorrent is now a true peer-to-peer protocol as a result of this enhancement.

The protocol is quite adaptable. Without any user participation, peers discover each other using the centralized or distributed tracker. Using policies and algorithms included in the protocol, peers pick which peers to download portions of the file from.

BitTorrent has transformed the broadcast media and file delivery landscapes. The internet has become the world's greatest source of Video-on-Demand content. BitTorrent has significantly reduced the expenses of distributing shows and movies, allowing practically anybody with access to the Internet to watch them. This has a huge impact on major networks, particularly content suppliers that in the end all use P2P solutions.

Bibliography

[1] Bittorrent clients. `https://en.wikipedia.org/wiki/Comparison_of_BitTorrent_clients`.

[2] Peer-to-peer. `https://en.wikipedia.org/wiki/Peer-to-peer`.

[3] Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon. *Handbook of Peer-to-Peer Networking*. 01 2010.

[4] J.F. Buford, H. Yu, and Eng Lua. P2p networking and applications. *P2P Networking and Applications*, 01 2009.

[5] Bittorrent. `https://wiki.theory.org/BitTorrentSpecification`.

[6] Udp. `https://www.libtorrent.org/udp_tracker_protocol.html`.

[7] Distributed hash table. `https://en.wikipedia.org/wiki/Distributed_hash_table`.

[8] Brandon Wiley. Distributed hash tables, part i. *Linux Journal*, 2003:7, 01 2003.

[9] Trackerless torrents. `http://bittorrent.org/beps/bep_0005.html`.

[10] Bram Cohen. Incentives build robustness in bittorrent. In *Proc IPTPS'03*, 2003.

[11] Arnaud Legout and Guillaume Urvoy-Keller. Understanding bittorrent: An experimental perspective. 01 2005.

[12] Alan A. A. Donovan. *The Go Programming Language*. Pearson Education, 1 edition, 2015.

[13] Katherine Cox-Buday. *Concurrency in Go: Tools and Techniques for Developers*. O'Reilly Media, Inc., 2017.

[14] Pex dht and magnet. `http://wiki.bitcomet.com/peers_seeds_torrent_tracker_dht_peer_exchange_pex_magnet_links`.