Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica (Classe L-31)

BitTorrent
Peer2Peer Networking

Laureando
De Vitis Fabio Michele

Matricola 127836

Relatore
Michele Loreti

Correlatore
None

A.A. 2023/2024

Indice

Elenco dei codici

Elenco delle figure

Elenco delle tabelle

Abstract

This report presents a comprehensive study of the BitTorrent protocol and its crucial role in peer-to-peer (P2P) networking. The study begins with an introduction to the fundamental principles and core components of BitTorrent, including trackers, metainfo files, seeders, and leechers, followed by an in-depth examination of how BitTorrent operates. Key algorithms and techniques used in BitTorrent, such as piece selection algorithms (rarest first, random first piece) and peer selection algorithms (choking and unchoking), are analyzed to understand their impact on performance and efficiency.

Security and privacy are addressed, highlighting potential threats, legal risks, privacy concerns, and the countermeasures available to mitigate these issues. A notable feature of this project is the development of an educational simulator that demonstrates piece selection and peer selection algorithms using Python. This simulator, available at GitHub, provides a practical demonstration of the theoretical concepts discussed, offering valuable insights into BitTorrent's functionality.

The report concludes by discussing the future potential of BitTorrent and P2P networking, considering the advancements in technology and network infrastructure that could lead to more robust, efficient, and secure P2P systems. This study underscores the power and versatility of BitTorrent, presenting it as a scalable solution for distributed file sharing and a catalyst for future innovations in P2P networking.

1. Introduction

BitTorrent is one of the piooners of distributed communications protocols and was invented by Bram Cohen in 2001. We should first start by introducing the concept of Peer-To-Peer networks. Peer-To-Peer or P2P, refers to computer networks using a distributed a distributed architecture; in P2P all the computers and devices that are part of it are referred to as Peers and they share and exchange workloads. There are no privileged peers, and there is not a a primary administrator, this ensures that every peer has the exact same right and duties as all the other ones. BitTorrent ,similarly, facilitates the transfer of large and highly demanded files without having a trusted central server. In fact in this context every client software endpoint, called *"Clients"*, collaborate with each other to enable reliable distribution of large files to multiple clients. This is a huge advantage because we do not rely on a single server to exchange files, and we could avoid the potential failure of this single node.

The primary goal of peer-to-peer networks is to share resources and help computers and devices to work in collaboration, provide specific services or execute specific tasks. P2P is used to share all computing resources such as processing power, network bandwith and disk storage space. However the most common usage of peer-to-peer networking is to share files on the internet. It is an ideal technology in such matter because it allows computer connected to them to receive and send files simultaneously.

A classic example to explain the functionality of the peer-to-peer technology is refering to the two road mechanism. In fact when you download a file from a website in a web browser, there's a simple mechanism that can be described as a one way road. The website works as a server while our machine, or computer, acts as a client. This can be compared to a one-way road where the file that we download, goes from a starting point A (the website) to a destination point B (our device).

Peer-to-peer networks work differently, they act as a two-way road, where the file is downloaded on our device bit by bits from different computers that are part of the peer-to-peer network, at the same time the file gets uploaded from our device and sent to other devices that are requesting the same files as us. It is clear that we can see this example as a two way road where our device acts as a connection-node between other peers, receiving and sending the same file simultaneously.

But why are peer-to-peer networks particularly useful and how does BitTorrent, specifically work? This report aims to provide a compehensive analysis of BitTorrent, focusing on its protocols, algorithms and applications. The justification for this report lies in the significant role that BitTorrent plays in efficient file sharing, resource utiliza-

tion, and maintaining system coherence in peer-to-peer networks. The central research question addresses how BitTorrent's underlying algorithms and techniques contribute to its efficiency, performance, and scalability in various distributed environments.

Additionally, this study explores the evolving role of BitTorrent in contemporary digital ecosystems, particularly its application in decentralized content distribution and emerging trends such as the blockchain technology. Highlighting its significance in ensuring efficient data distribution, enhancing security, and maintaining the integrity of shared resources, BitTorrent represents a paradigm shift in peer-to-peer networking, introducing new challenges and opportunities for optimization.

The report is organized as follows:

- **Chapter One:** Introduction
- **Chapter Two:** Overview of the BitTorrent protocol with a detailed explanation of its main architectural components and an example of the working stage.
- **Chapter Three:** We analyzed the Piece Selection Algorithms and the Peer Selections algorithms and described the principles of Swarming and End Game Mode.
- **Chapter Four:** We considered the security and privacy aspect in general, without delving into too intricate details but providing a basic understanding on the matter.
- **Conclusion**

2. BitTorrent Protocol: An Overview

This chapter will look into the BitTorrent protocol in detail, covering the fundamental principles, major components, and step-by-step operation of the protocol. All this understanding gives insight into the working of BitTorrent and is a big reason for its huge popularity as a protocol for peer-to-peer file sharing.

2.1 Basic Principles and Concepts

In this section, we delve into the foundational principles and main concepts that characterize the BitTorrent protocol and, in general, peer-to-peer networks. We will highlight the primary mechanisms of file sharing, the advantages over classical client-server structures, and the entities that take part in this system.

First of all it's better to understand why BitTorrent got so popular in the first place and what are the innovations that it brings to the table. As we introduced in the Chapter 1, the most common method before the advent of BitTorrent by which files were transferred on the internet, was a client server model.

In this particular structure a central server has the objective to send the entire file to all the clients that request it, this mechanism is used both in *HTTP* and *FTP*. In this environment the client do not speak between each other and have no interaction. Thus this structure is pretty simple to set up and has some advantages, the disadvantages outweigh them. In fact it has significant problems, with files particularly big in size and highly requested, imagine in fact a huge number of clients that try to download a large file from the same server simultaneously. This is gonna put a high stress on the server bandwidth and overall resources to distribute this file to every client, causing delays and potential server overloads. Having a central server that handles all the request could be also a fatal error as it result in a single point of failure.

Here is were peer-to-peer networks and BitTorrent come in clutch. Practically the file is divided into smaller chunks and distributed to multiple clients. This way a client can download a small piece N from the original bigger file from another client that holds it, at the same time the new client can upload the chunk he downloaded so another peer can do the same thing. **Peers** are all the computer/devices that are part of the peer-to-peer networked system and the ones that are active are defined as **active peers.**

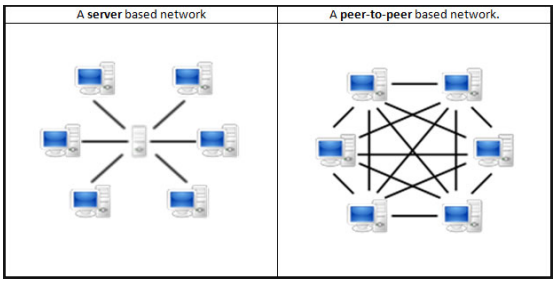

Figura 2.1: Difference between client-server and P2P network.

It's important to make a clear distinction between the entities that compose a peer-to-peer network, in fact other than peers we can distinguish the following two main entities:

- **Leechers**: they are nodes that are downloading the blocks of file from the seeders or peers.
- **Seeders:** they are the peers that have the complete file downloaded in their system. Obviously a higher number of seeders will result into a higher download speed for the leechers that are trying to download the file. If there is only one seeder we are similarly close to a client-server architecture regarding file distribution/sharing.

To make a brief recap, the shift to P2P Networks like BitTorrent introduces several key advantages over traditional methods:

- **Decentralization:** In a decentralized network, such as P2P, there is no single point of failure. The absence of a central server means that the system is more resilient and if one peer goes offline and stops working, the others can continue sharing the file, ensuring higher availability and reliability.
- **Piece and Block Division:** Files are divided into smaller pieces, which are further divided into blocks. This division allows for parallel download from multiple peers, increasing the overall download speed and making an effective usage of the bandwidth.
- **Tit-for-Tat:** BitTorrent in particular, employs a fairness mechanism where peers prioritize uploading to those who opload to them. This tit-for-tat strategy encourages mutual sharing and discourages the so called *free-riding*, which is basically the act of downloading without uploading. As a result the network mantains balanced communications and exchange of data. We will dive into a deeper explanation of this concept in next chapters.
- **Single point of Failure:** Traditional client-server methods suffer from the risk of a single point of failure. If the central server goes down, all clients are affected, and file access is lost. In contrast, BitTorrent's decentralized nature means that the failure of one or more peers does not cripple the network.

BitTorrent's approach, by decentralizing file distribution and promoting mutual sharing, has transformed how large files are shared over the internet. The combination

of piece and block division, swarming, and the tit-for-tat strategy ensures that the network remains robust, efficient, and scalable. This innovative protocol addresses the shortcomings of traditional client-server models, providing a more resilient and faster method for file sharing.

2.2 Components of BitTorrent

Other than the entities that I have described in the 2.1 there are other entities that take part into forming a P2P Networked System. This section explores the essential components that constitute the BitTorrent ecosystem, including trackers, metainfo files (torrents), seeders, and leechers. Understanding these components is crucial for comprehending how BitTorrent functions as a decentralized file-sharing protocol.

2.2.1 Trackers

A BitTorrent tracker is server software that centrally coordinates the transfer of files among users. Actually, it does not store a copy of the file on the tracker itself; its primary purpose is to help peers discover each other. A very simple protocol, built on top of HTTP, makes trackers and clients' communication easy. Clients inform the tracker about the file they wish to download, along with their IP address and port number. In response, the tracker returns a list of peers for that file and their contact information. This collection of peers for the same torrent is what is known as a "swarm." It is only through the tracker that a collection of peers can come together to connect for data exchange sessions. Due to the presence of the central coordination server, the BitTorrent protocol is classified under hybrid P2P implementation. For example, there is a famous Linux ISO image tracker called LinuxTracker.

2.2.2 Metainfo File (Torrent File)

A metainfo file is usually referred to as a torrent file and has a .torrent extension. It contains encoded information that is highly important for correct work with the BitTorrent protocol; data included are the URL of the tracker, the name of the file, and hashes of the file's pieces used for verifying the integrity of downloaded data. Torrent files are usually generated by client software. To make a torrent file, the user only requires a list of trackers and the original file that is to be shared. Torrent files can then be distributed in the usual ways: email or websites that host files. For example, to distribute a new version of a Linux distribution, developers place the tracker URL and the new Linux ISO image into a BitTorrent client so that it produces a .torrent file. This file is then distributed by the official website of the distribution and users can download the new version using BitTorrent.

2.2.3 Seeders

A seeder is a peer who holds the complete file for downloading. The original uploader of the file, in fact, starts as the first seeder, and they must continue to upload the file until at least one complete copy is made available among other peers. In other words, every user in the swarm can continue downloading the file at their will as long as there is one complete copy of the file inside the swarm. This is particularly useful while

distributing a Linux ISO image, where developers with the full file work as seeders. They make sure to always upload, so that the file does not disappear and is available to every new downloader.

2.2.4 Leechers

Leechers are those clients who have not received the whole file. Leechers receive a list from the tracker of who has the correct pieces of the file and proceed to download those pieces. Importantly, leechers can also upload the pieces they have already downloaded to other peers, contributing to the overall swarm. As leechers download pieces of the file, they verify these with the hashes the original author provided. When a leecher has all the pieces, he becomes a seeder. For example, users will be leeching a Linux ISO image using BitTorrent; initially, they are leechers. The more pieces they download and upload to others, the better it is for swarm efficiency. Once they have the entire file, they become seeders and continue uploading the file to other leechers.

2.2.5 Piece Selection Algorithms

BitTorrent uses advanced piece selection algorithms for an optimized downloading procedure. The prime motive behind this is to achieve maximum replication for each piece in the swarm; hence, if a collection has rare pieces, it circulates them rapidly to enable availability. Strategic piece selection avoids certain pieces from being bottlenecks and assures an adequate and smooth download experience for all members in the process. For instance, one of the more popular algorithms is a "rarest first" approach in which a peer downloads those pieces that are available the least in number. This way, the availability of all pieces of a file in a swarm is maximized, further enhancing the robustness and efficiency of the protocol.

2.2.6 Peer Protocol and Messages

The BitTorrent peer protocol operates over TCP or uTP and governs the communication between peers, encompassing both seeders and leechers, during the file-sharing process. This protocol ensures efficient data exchange through a sequence of structured messages.

Handshake

The handshake is the initial step in establishing a connection between peers. This process begins with each peer sending the character nineteen (decimal), followed by the string 'BitTorrent protocol'. This is followed by the exchange of a 20-byte SHA1 hash of the info value from the metainfo file and a 20-byte peer ID. This exchange verifies that both peers are participating in the same torrent and establishes a unique identity for each peer in the swarm.

Messages

Following the handshake, peers exchange a series of messages to coordinate the sharing of file pieces. Each message is prefixed with its length and type. The main message types are summarized in Table **??** and detailed below.

Code	Message Type	Description
0	Choke	Informs the peer to stop sending data
1	Unchoke	Informs the peer to start sending data
2	Interested	Indicates interest in downloading pieces
3	Not Interested	Indicates lack of interest in downloading pieces
4	Have	Announces the possession of a specific piece
5	Bitfield	Provides information about pieces the peer has
6	Request	Requests a specific piece from a peer
7	Piece	Sends a piece of the file
8	Cancel	Cancels a previous request for a piece

Tabella 2.1: BitTorrent Peer Protocol Message Types

- **Choke (0)**: This message informs the peer that it will no longer be sent any data. The choking peer ceases to upload data to the choked peer. This message contains no payload.
- **Unchoke (1)**: This message informs the peer that it can now request data. The unchoking peer resumes uploading data to the unchoked peer. This message contains no payload.
- **Interested (2)**: Sent by a peer to indicate that it is interested in downloading pieces from the peer it is communicating with. This message contains no payload.
- **Not Interested (3)**: Sent by a peer to indicate that it is not interested in downloading pieces from the peer it is communicating with. This message contains no payload.
- **Have (4)**: This message is sent to inform a peer that the sender has successfully downloaded and verified a specific piece of the file. The payload of this message is a single integer, representing the index of the piece.
- **Bitfield (5)**: Typically sent as the first message after the handshake, this message indicates which pieces the sender already has. The payload is a bitfield, where each bit represents the presence (1) or absence (0) of a piece. The first byte of the bitfield corresponds to pieces 0-7, the second byte to pieces 8-15, and so on. Any spare bits at the end are set to zero.
- **Request (6)**: This message is used to request a specific piece from a peer. The payload includes three integers: the index of the piece, the beginning offset within the piece, and the length of the piece requested.
- **Piece (7)**: This message is used to send a piece of the file to a peer. The payload includes three parts: the index of the piece, the beginning offset, and the actual piece data.
- **Cancel (8)**: This message cancels a previously sent request. The payload is identical to that of the request message, including the index, beginning offset, and length of the piece.

These messages enable BitTorrent to efficiently manage and coordinate the distribution of file pieces across the network, ensuring robust and scalable peer-to-peer file sharing.

Each message type plays a specific role in the protocol, facilitating communication and data transfer between peers.

2.3 How BitTorrent works

In this section, we provide a detailed, step-by-step explanation of how BitTorrent operates, from the initiation of a download to its completion. This process illustrates the efficiency and decentralized nature of the BitTorrent protocol.

1. **Creating a Torrent**:
    - The process begins with a user, known as the initial seeder, who wants to share a file. The seeder uses BitTorrent client software to generate a torrent file.
    - The torrent file includes metadata about the shared file, such as its name, size, piece length, and a hash of each piece for integrity verification. It also contains the URL of a tracker that will help coordinate the sharing process.
2. **Distributing the Torrent File**:
    - Once created, the torrent file (.torrent) can be distributed via various means such as websites, email, or file-sharing platforms. The file is typically small and easy to share.
    - Potential downloaders obtain the torrent file and open it with their BitTorrent client.
3. **Joining a Swarm**:
    - When a peer opens the torrent file, the BitTorrent client contacts the tracker specified in the torrent file.
    - The tracker responds with a list of peers currently participating in the swarm, including their IP addresses and ports. This list helps peers find each other.
4. **Establishing Connections**:
    - The BitTorrent client establishes connections with some of the peers from the list provided by the tracker. These connections are typically established using the TCP protocol.
    - Peers exchange information about the pieces they have, enabling them to request and send pieces to each other.
5. **Downloading and Uploading Pieces**:
    - The file is divided into multiple pieces, and each piece is further divided into smaller blocks. Peers begin downloading different pieces from each other, leveraging parallelism to maximize download speeds.
    - As peers download pieces, they immediately begin uploading them to other peers. This process, known as swarming, ensures that the load is distributed across the network and that all peers contribute to the file-sharing process.
6. **Piece Selection and Downloading**:

- BitTorrent uses piece selection algorithms to optimize the downloading process. A common strategy is "rarest first," where a peer prioritizes downloading the least available pieces first. This ensures that all pieces remain available within the swarm.
- Peers also use the "tit-for-tat" strategy to encourage fair sharing. They prioritize uploading to peers who upload to them, promoting mutual cooperation and discouraging free-riding.

7. **Piece Verification**:
   - Each downloaded piece is verified using a hash function included in the torrent file's metadata. This step ensures data integrity and prevents corruption or tampering.
   - If a piece fails verification, it is discarded and re-downloaded from another peer. This mechanism maintains the quality and reliability of the downloaded file.
8. **Completion and Seeding**:
   - Once all pieces are downloaded and verified, the peer assembles them into the complete file.
   - The peer transitions from a leecher to a seeder, continuing to upload the file to other peers. This step is crucial for maintaining the health and availability of the swarm.
9. **Leaving the Swarm**:
   - Peers can leave the swarm at any time by closing their BitTorrent client. However, staying to seed after downloading helps sustain the swarm and supports other downloaders.
   - Trackers periodically receive updates from peers and remove those who have left the swarm from the peer list, ensuring that the list remains current and accurate.

2.3.1 Example Scenario

To illustrate this process, consider a scenario where a new Linux distribution is released:

- The developers create a torrent file for the Linux ISO image using their BitTorrent client. This torrent file includes the tracker URL and metadata about the ISO image.
- They distribute the torrent file via their official website.
- Users interested in downloading the new Linux distribution download the torrent file and open it with their BitTorrent clients.
- The BitTorrent client contacts the tracker, receives a list of peers, and begins establishing connections.
- Users (leechers) start downloading pieces of the ISO image from multiple peers simultaneously, while also uploading the pieces they have downloaded.

- As they download, they verify the integrity of each piece using the hash values in the torrent file.
- Once they have downloaded all pieces, they assemble them into the complete ISO image and become seeders, continuing to share the file with new leechers.
- This decentralized process ensures that the load is distributed, download speeds are optimized, and the file remains available to all users.

Through these steps, BitTorrent efficiently distributes large files across a decentralized network, leveraging the collective bandwidth and resources of all participating peers. This approach not only speeds up the download process but also ensures the robustness and scalability of the file-sharing system.

3. Algorithms and Techniques Used in BitTorrent

This section explores the details about the algorithms and techniques used in BitTorrent, we are gonna delve into the particulars on the Piece Selection Algorithms and the Peer Selection Algorithms, providing small scripts in Python to properly understand the logic. At the end we are gonna clarify the concepts of Swarming and End Game Mode.

3.1 Piece Selection Algorithms

In BitTorrent as we said in the previous chapter, the file is split into pieces and then distributed across the network, in this sections we are gonna analyze the following algorithms and understand the logic on how to select pieces.

3.1.1 Rarest First

The Rarest First algorithm optimizes the distribution of data among peers in a network. Each peer maintains a count of how many copies of each data piece exist within its peer group. The algorithm identifies the piece with the fewest copies, denoted as m. All pieces that have m copies are added to a set called the rarest pieces set. This set is dynamically updated with any addition or removal of piece copies in the peer group.

At the beginning of the download process, a peer employs the Random First policy if it has acquired fewer than four pieces. This policy dictates that the next piece is chosen randomly, accelerating the initial acquisition phase. Pieces chosen randomly are typically more common, hence available at higher speeds, which is crucial early on for establishing a base to participate in the network's exchange dynamics.

Once a peer has four or more pieces, it shifts to the Rarest First algorithm. The selection of the next piece to download is made randomly from the rarest pieces set. This strategy helps distribute the least replicated pieces more evenly across the network, enhancing the robustness and efficiency of the data distribution.

Additionally, the BitTorrent protocol incorporates a Strict Priority policy. This policy ensures that once a peer starts receiving blocks of a specific piece, it prioritizes the download of the remaining blocks of that piece to quickly complete it. Completing pieces rapidly is vital because only fully received pieces can be shared with others.

Towards the end of the download process, when all pieces have been requested, the End Game mode is initiated. In this mode, a peer aggressively requests all missing blocks from every peer that might have them. If a block is received, the peer immediately cancels any outstanding requests for that block with other peers. This mode minimizes the latency in the final phase of downloading and effectively manages the peer's limited buffer for pending requests.

To illustrate, consider a network where a peer is part of a group with the following piece distribution: 5 copies of piece A, 3 copies of piece B, and 2 copies each of pieces C and D. Initially, the rarest pieces set includes C and D. If the peer already has four pieces, it will choose its next piece from C and D at random. This ensures a balanced distribution and availability of all pieces within the network.

Metric	**Description**
Computational Complexity	O(n + m log m) per selection operation, where n is the number of peers and m is the number of pieces.
Memory Usage	O(p + n) where p is the total number of pieces and n is the number of peers in the system.
Scalability	Performs well in large networks due to decentralized piece selection and replication.
Network Efficiency	Improves as rare pieces are evenly distributed, reducing bottlenecks and improving throughput.
Response Time	Can vary; generally faster initial download time with Random First, slower as rare pieces are targeted.
Fairness	High, as it aims to balance piece availability across the network.

Tabella 3.1: Metrics of the Rarest First Algorithm

I have realized a simple example implementation of the random first algorithm in Python, this script assumes that I already know the current distribution of each piece among the peers.

```
1  import random
2
3  def rarest_first_piece(piece_distribution, acquired_pieces):
4      """
5      Selects the next piece to download based on the rarest first algorithm.
6
7      Parameters:
8      - piece_distribution (dict): A dictionary where keys are piece indices and
           values are the count of how many peers have this piece.
9      - acquired_pieces (set): A set containing the indices of pieces already
           acquired by the peer.
10
11     Returns:
12     - int: The index of the next piece to download.
13     """
14     # Filter out pieces that have already been acquired
15     available_pieces = {k: v for k, v in piece_distribution.items() if k not in
           acquired_pieces}
16
17     # Find the minimum number of replicas among the available pieces
18     if not available_pieces:
19         return None
20     min_replica = min(available_pieces.values())
21
22     # Create a list of pieces that have the minimum replica count
```

```
23      rarest_pieces = [piece for piece, count in available_pieces.items() if
            count == min_replica]
24
25      # Randomly select from the rarest pieces
26      return random.choice(rarest_pieces)
27
28  # Example usage
29  piece_distribution = {0: 5, 1: 3, 2: 2, 3: 2, 4: 5, 5: 3}
30  acquired_pieces = {0, 4}
31  next_piece = rarest_first_piece(piece_distribution, acquired_pieces)
32  print("Next piece to download:", next_piece)
```

Codice 3.1: Code for Rarest First Piece Selection

In our project, the Rarest First algorithm is implemented to optimize data distribution across peers by ensuring that pieces with the fewest copies are downloaded first. Here is a breakdown of our implementation, which is detailed in Listing 3.1:

- **Importing Necessary Libraries** (Line 1 in Listing 3.1): We import the `random` module, which is essential for the random selection aspect of the algorithm.
- **Function Definition** (Line 3 in Listing 3.1): The function `rarest_first_piece` is defined with parameters for the distribution of pieces and the pieces already acquired by the peer.
- **Filtering Available Pieces** (Lines 15-16 in Listing 3.1): The algorithm filters out the pieces that the peer has already acquired to focus on those still available.
- **Determining the Rarest Pieces** (Lines 18-24 in Listing 3.1): It calculates which pieces are the rarest by finding the minimum number of replicas and then identifies all pieces with this minimal count.
- **Random Selection of Next Piece** (Line 26 in Listing 3.1): From the list of rarest pieces, one is randomly selected to be downloaded next, ensuring the even distribution of these pieces across the network.
- **Example Usage** (Lines 28-32 in Listing 3.1): This segment provides an example scenario demonstrating how to invoke the function and retrieve the next piece to download.

This method enhances the network's robustness by reducing the likelihood of piece extinction and improving overall availability.

3.1.2 Random First Piece

As we introduced in the previous section 3.1.1, the Random First Piece algorithm serves a crucial role during the initial stage of data acquisition in the BitTorrent protocol. When a peer joins the network and has less than four pieces of the file, it demploys this algorithm to select pieces randomly from the available pool. This technique enables the BitTorrent protocol to speed up the initial download phase, allowing the peer to quickly acquire a few pieces, facilitating data exchange with the rest of the network.

This random selection ensures that new pieces can start contributing to the network's health and balance by increasing the variety of pieces they old without focusing on the rarity of the pieces at first. The randomness also helps in avoiding early bottlenecks where many new peers might otherwise target the same initial pieces.

As the peer acquires and accumulates pieces, it transitions from the Random First Piece to the Rarest First algorithm, which then guides the selection process to optimize the overall distribution of pieces in the network.

A practical example of this might be a peer in a network with a diverse set of available pieces. The peer randomly selects from this variety, quickly ammassing a foundation set of pieces that allows it to engage more effectively in the network's sharing dynamics.

Metric	**Description**
Computational Complexity	O(1) per selection, as pieces are selected randomly without the need for tracking piece frequency.
Memory Usage	Minimal, primarily requires tracking of acquired pieces to avoid re-downloading.
Scalability	Highly scalable in initial download phases due to lack of dependency on piece distribution metrics.
Network Efficiency	Varies; efficient in early stages by promoting quick diversification of piece distribution.
Response Time	Typically fast in the initial stages, as the selection process is straightforward and quick.
Fairness	Moderate, as all pieces have an equal chance of being selected, regardless of their current distribution.

Tabella 3.2: Metrics of the Random First Piece Algorithm

The following is a simple implementation of the Random First algorithm in Python to provide a demonstration:

```
1  import random
2
3  def random_first_piece(total_pieces, acquired_pieces):
4      """
5      Selects a random piece from the available pieces that the peer has not yet
           acquired.
6
7      Parameters:
8      - total_pieces (int): Total number of distinct pieces of the file.
9      - acquired_pieces (set): A set containing the indices of pieces already
           acquired by the peer.
10
11     Returns:
```

```
12      - int: The index of the next piece to download.
13      """
14      # Create a set of all pieces
15      all_pieces = set(range(total_pieces))
16
17      # Determine which pieces are still needed by removing the acquired pieces
            from all pieces
18      available_pieces = all_pieces - acquired_pieces
19
20      # Randomly select a piece from the available pieces
21      return random.choice(list(available_pieces))
22
23  # Example usage:
24  total_pieces = 10  # Let's say there are 10 pieces in total
25  acquired_pieces = {2, 5, 7}  # Assume pieces with indices 2, 5, and 7 have
        already been acquired
26
27  # Get the next piece to download
28  next_piece = random_first_piece(total_pieces, acquired_pieces)
29  print("Next piece to download:", next_piece)
```

Codice 3.2: Code for Random First Piece Selection

In our project, we implemented the Random First Piece selection algorithm, crucial for quickly integrating new peers into the network by rapidly increasing their piece count. Below, we detail the Python implementation referenced from Listing 3.2:

- **Importing Necessary Libraries** (Line 1 in Listing 3.2): The `random` module is imported to facilitate the random selection of file pieces, a fundamental aspect of the Random First Piece selection algorithm.
- **Function Definition** (Line 3 in Listing 3.2): We define the function `random_first_piece` which takes the total number of pieces and the pieces already acquired by the peer as arguments.
- **Computing Available Pieces** (Lines 15-18 in Listing 3.2): The function computes which pieces are still available for download by subtracting the set of acquired pieces from the set of all pieces.
- **Random Selection of Next Piece** (Line 21 in Listing 3.2): A piece is randomly selected from the available pieces, demonstrating the use of the `random.choice` method.
- **Example Usage** (Lines 24-29 in Listing 3.2): This segment of the code provides an example scenario demonstrating how to call the function and retrieve the next piece to download.

This method ensures that new peers contribute to the network's diversity and robustness by preventing early bottlenecks in piece distribution.

3.2 Peer Selection Algorithms

3.2.1 Choking and Unchoking Algorithms

The Choking Algorithm is a core component of the BitTorrent protocol, designed so that resource sharing can be optimized without being abused by non-contributing peers. It is a dynamic mechanism governing the amount of upload bandwidth that a peer offers

to others, conditionally based on their contribution, and using the tit-for-tat strategy motivated by game theory.

Broadly speaking, the Choking Algorithm aims at promoting cooperation among the peers in the network. This at least partially ensures that upload isn't wasted on non-reciprocating peers; those who contribute bandwidth to the network are given priority in receiving data. This decreases not only the incidence of free riders—people who extract resources without giving them back—but also increases overall network efficiency and speed by two orders of magnitude.

We can define Choking as a transient refusal to upload. We say that peer A has chocked peer B when A refuses to send/upload data to B, but it still allows A to get data from B. In contrary when peer A wants to allow peer B to download a piece, we say that peer B is unchoked by Peer A. We use to say that peer A is interested in peer B when it wants a part of the file that peer B has.

The main question is how to find which peer to unchoke, the main concept is that a peer always unchokes the peer that offers the best download speed. This type of prioritization is based on the principle of reciprocity where a peer will send data to peers from whom it gets data faster. This encourages peers to let other peers download and avoid free riders.

- **Regular Choking and Unchoking**: This is done at a certain set of intervals, typically every 10 seconds or so, a peer re-evaluates which peer to choke and unchoke based on their upload/download rates. This decision is made to maximaze the peer's download speed and it is a direct application of the tit-for-tat strategy as we briefly explained before.
- **Optimistic Unchoking:** In addition to the regular unchoking mechanism, BitTorrent implements Optimistic Unchoking where peers randomly unchoke a peer who was previously choked, *regardless* of their upload contribution. This is done approximately at intervals of 30 seconds and it has multiple functions: first of all it allows new peers a chance to prove their upload capacity, it helps in discovering if previously under performing peers have increased their performances and it mantains a high degree of randomness and exploration in the network, avoiding a static environment.

To summarize the pro and cons of the algorithm we can look at the Table 3.3 :

Tabella 3.3: Pros and Cons of the Choking Algorithm in BitTorrent

Pros	**Cons**
Encourages cooperative behavior among peers by rewarding upload contributions.	May temporarily disadvantage new peers who have not had a chance to prove their upload capabilities.
Effectively manages network resources and prevents congestion by controlling the number of simultaneous uploads.	Could lead to system fibrillation if not properly managed, due to frequent state changes.
Optimizes the overall efficiency and speed of the network by prioritizing uploads to peers who contribute back.	Requires precise tuning of timing parameters to prevent too rapid choking and unchoking, which can be technically challenging.
Introduces randomness and exploration through optimistic unchoking, improving network dynamism and fairness.	Random selection in optimistic unchoking might temporarily benefit free riders, although it is necessary for network health.
Supports a scalable network environment as it effectively deals with large numbers of peers.	Optimistic unchoking, while beneficial, uses additional bandwidth that might have been used more efficiently elsewhere.

This is a simplified implementation of choking algorithms in BitTorrent using Python:

```
import random
import time
from collections import defaultdict

class Peer:
    def __init__(self, peer_id):
        self.peer_id = peer_id
        self.upload_rate = 0
        self.download_rate = 0
        self.choked = True

class TorrentClient:
    def __init__(self, num_peers):
        self.peers = [Peer(peer_id) for peer_id in range(num_peers)]
        self.optimistically_unchoked_peer = None

    def calculate_rates(self):
        # Randomly simulate upload and download rates for each peer
        for peer in self.peers:
            peer.upload_rate = random.randint(1, 100)
            peer.download_rate = random.randint(1, 100)

    def choke_algorithm(self):
        # Choke all peers initially
        for peer in self.peers:
            peer.choked = True

        # Sort peers by their upload rate in descending order
        sorted_peers = sorted(self.peers, key=lambda p: p.upload_rate, reverse=
            True)

        # Unchoke top 4 peers with highest upload rates
        for peer in sorted_peers[:4]:
            peer.choked = False

```

```
35          # Select a random peer for optimistic unchoking
36          self.optimistically_unchoked_peer = random.choice([peer for peer in
                self.peers if peer.choked])
37          self.optimistically_unchoked_peer.choked = False
38
39      def display_peer_status(self):
40          print("Peer Status:")
41          for peer in self.peers:
42              status = "Unchoked" if not peer.choked else "Choked"
43              print(f"Peer {peer.peer_id}: {status}, Upload Rate: {peer.
                    upload_rate}, Download Rate: {peer.download_rate}")
44          if self.optimistically_unchoked_peer:
45              print(f"Optimistically Unchoked Peer: {self.
                    optimistically_unchoked_peer.peer_id}")
46
47  def main():
48      num_peers = 10
49      client = TorrentClient(num_peers)
50
51      while True:
52          client.calculate_rates()
53          client.choke_algorithm()
54          client.display_peer_status()
55          time.sleep(10)  # Run the algorithm every 10 seconds
56
57  if __name__ == "__main__":
58      main()
```

Codice 3.3: Code for Choking

Lines 1 to 3 import the necessary modules:

- `random`: Used for generating random numbers.
- `time`: Used for introducing delays.
- `defaultdict`: Imported from `collections`, though not used in this code.

Lines 5 to 10 define the `Peer` class:

- `peer_id`: A unique identifier for each peer.
- `upload_rate`: Initialized to 0.
- `download_rate`: Initialized to 0.
- `choked`: Initially set to `True`, meaning all peers are choked.

Lines 12 to 15 define the `TorrentClient` class:

- `__init__`: Initializes a list of `Peer` objects and sets the optimistically unchoked peer to `None`.

Lines 17 to 21 define the `calculate_rates` method:

- Simulates random upload and download rates for each peer by assigning random values between 1 and 100.

Lines 23 to 26 define the `choke_algorithm` method:

- Chokes all peers initially.
- Sorts peers by upload rate in descending order.

- Unchokes the top 4 peers with the highest upload rates.
- Selects a random peer from the choked peers for optimistic unchoking.

Lines 39 to 45 define the `display_peer_status` method:

- Prints the status of each peer, indicating whether they are choked or unchoked, along with their upload and download rates.
- Prints the optimistically unchoked peer, if any.

Lines 47 to 50 define the `main` function:

- Initializes the `TorrentClient` with a specified number of peers.
- Repeatedly simulates rates, runs the choking algorithm, displays peer status, and waits for 10 seconds before repeating.

3.3 Swarming

Swarming is one of the basic techniques for file distribution in peer-to-peer networks, which enhances the effectiveness and quickness of file transfer. It uses bandwidth from a number of peers at the same time in order to allow a client to download different pieces of the file from various sources. This will increase download speed and also bring down the load on a single source with the distribution of traffic over several nodes.

The principles behind swarming are simple but very powerful. When a peer joins a network and desires to download a file, it does not set out to download the whole file from one source. Rather, it requests different pieces of the file from various peers who have either complete or partial copies of the file. This division of file pieces between peers is what characterizes swarming.

- **Redundancy and Reliability:** Being a swarm inherently confers redundancy in data: if a peer goes offline or becomes unavailable, other peers can provide the missing data of the file. It is a surefire way to boost the robustness of the network. This means higher availability and fault tolerance.
- **Scalability:** Every time another peer joins the network, the network's bandwidth capacity increases. That means popular files can be spread to more users, more quickly, without a corresponding rise in cost or resource requirements.
- **Load Balancing:** The data transfer requests are spread out to several peers, so that each peer doesn't have to handle too many requests. This keeps the network running fast and stable.

These piece selection and peer selection algorithms must be selected carefully: as you've seen in previous sections, they have a substantial effect on the efficiency of the swarming process. To optimise download speed and strengthen the network's health, a typical strategy is to favour the rarest pieces or the fastest peers, respectively.

When used in conjunction with proper piece and peer-selection strategies, swarming allows for the efficient deployment and employment of the best set of peers for every individual transfer, provided there are sufficient unchoked peers. It is also responsible for the robustness and scalability of file distribution processes – in distributed computing, swarming is the way to go.

3.4 End Game Mode

End Game Mode is a crucial phase in peer-to-peer (P2P) file sharing networks, designed to ensure the swift and reliable completion of file downloads. This mode kicks in when a peer is nearing the completion of downloading all pieces of the file but has not yet received the final pieces required to finish the download.

The primary purpose of End Game Mode is to prevent the download process from stalling when only a few pieces of the file are missing. In earlier phases of the download, peers typically request pieces from other peers in a selective manner based on piece and peer selection algorithms. However, as the download nears completion, it becomes critical to ensure that these last pieces are obtained quickly and reliably.

In End Game Mode, the downloading peer sends out requests for the remaining pieces to all peers in the network that have these pieces available. This aggressive strategy contrasts with the more measured approach used during earlier download phases. By requesting the same pieces from multiple peers simultaneously, the likelihood of receiving these pieces quickly is significantly increased.

Key aspects of End Game Mode include:

- **Multiple Requests:** The peer sends multiple requests for the same piece to different peers. This redundancy helps mitigate the risk of delays caused by unresponsive or slow peers.
- **Cancellation of Redundant Pieces:** Once a piece is received from any peer, the downloading peer immediately sends cancellation messages to other peers to stop them from sending the now redundant piece. This helps to conserve network bandwidth and prevents unnecessary data transmission.
- **Finalization of Download:** The use of End Game Mode helps ensure that the final pieces are acquired quickly, allowing the peer to complete the download and reassemble the file with minimal delay.

The benefits of the End game mode consist in:

- **Speed:** By aggressively requesting the remaining pieces, End Game Mode significantly speeds up the completion of the download.
- **Reliability:** The redundancy of requests ensures that even if some peers are slow or unresponsive, other peers can fulfill the requests, thereby increasing the reliability of the download process.

- **Efficiency:** The cancellation of redundant requests ensures that network resources are used efficiently, reducing unnecessary bandwidth consumption.

While End Game Mode is highly effective, it also presents certain challenges:

- **Network Congestion:** The surge in requests for the remaining pieces can temporarily increase network congestion, especially if many peers are in End Game Mode simultaneously.
- **Resource Allocation:** Implementing efficient algorithms to manage the sending and cancellation of multiple requests requires careful resource allocation and management to avoid performance bottlenecks.

End Game Mode is a vital component of peer-to-peer file sharing protocols, ensuring that the final stages of the download process are completed swiftly and reliably. By leveraging aggressive request strategies and efficient cancellation mechanisms, End Game Mode helps peers achieve faster and more dependable downloads, contributing to the overall effectiveness of P2P networks.

4. Security and Privacy

While torrenting offers a unique and efficient method of file sharing, it is not without its dangers. This chapter discusses the primary risks associated with torrenting and explores various methods to enhance security and privacy for users.

4.1 Security Threats & Legal Risks

There are several legal risks associated with using BitTorrent, the most prominent being **Copyright Infringement**. This occurs when individuals download or share copyrighted content without the necessary permissions. Such actions can lead to legal repercussions, including monetary penalties or even imprisonment, contingent on the laws of the respective jurisdiction.

Copyright represents a legal protection afforded to the creators of original works, spanning literary, musical, and artistic genres. Many Internet Service Providers (ISPs) monitor BitTorrent traffic and, upon detecting the unauthorized downloading of copyrighted content, may issue warning notices, reduce internet speeds, or report the activities to copyright enforcement agencies.

Another significant threat is the presence of malware. Not all torrents are what they appear to be; some files may be disguised as malware or contain harmful viruses capable of damaging your device and compromising your personal information. This issue is particularly prevalent with pirated software and cracked versions of applications. Even if such software appears to function correctly, it may operate as a Trojan horse, providing unauthorized access to your computer.

Furthermore, **Data Breaches** present a considerable risk. When using BitTorrent, your IP address is visible to others in the network. This visibility can be exploited by malicious actors to track your online activities or launch attacks against your device. Exposure of your IP address also increases the risk of identity theft or unauthorized access to personal information. There are also some less impactful risks such as the bandwidth consumption and the overall decrease of the internet speed for other activities, this is caused by the peer-to-peer sharing.

4.2 Privacy Concerns

When participating in a BitTorrent swarm, a significant privacy issue arises due to the visibility of your IP address to all other peers in the network. This exposure can lead to several privacy risks. Your IP address is a unique identifier for your internet connection. In a Torrent swarm, this address is visible to everyone else in the network. This can be exploited by malicious actors who may use this information to target your connection, identify your location, or even launch attacks against your device.

The exposure of your IP address can make you vulnerable to various attacks. These include Distributed Denial of Service (DDoS) attacks, where multiple compromised systems target a single system, causing a denial of service. Additionally, attackers might attempt to gain unauthorized access to your system or network.

Malicious actors or even third-party organizations can track and monitor your activities based on your IP address. This can lead to a breach of privacy and unauthorized surveillance. Tools like SOCRadar can identify if an institution's IP address is part of torrent networks, providing insights into potential privacy risks for organizations.

Several open-source methods and tools are available to detect if your IP address is part of a Distributed Hash Table (DHT) network, which is used in BitTorrent for peer discovery.

- **iknowwhatyoudownload.com**: This website provides a snapshot of recent downloads associated with a given IP address. By monitoring DHT networks and using bait files, they analyze collected data to show what has been downloaded. They claim to monitor up to 200 million peer-sharing activities daily from nearly 7 million torrents.

- **binaryedge.io:** This tool can be used to check if your IP address appears in torrent networks. It monitors torrent networks and gathers data about nodes and peers involved in serving and downloading torrent files. It also collects metadata about the torrents, such as their names and info hashes.

These tools and methods highlight the broader issue of IP privacy in BitTorrent networks. The ability to monitor and analyze IP addresses within these networks means that your download activities can be tracked and linked to your IP address. This can lead to unwanted exposure and potential legal or security implications.

The fundamental problem with IP privacy in BitTorrent is the inherent design of peer-to-peer networks, which rely on direct connections between peers. While this design is efficient for distributing files, it also means that participants must expose their IP addresses to each other. This trade-off between efficiency and privacy is a significant concern for users who prioritize their online anonymity and security.

In summary, the visibility of IP addresses in BitTorrent networks poses serious privacy risks, making users susceptible to tracking, attacks, and monitoring. Tools and websites that monitor these networks further illustrate the extent of this privacy issue, underscoring the importance of being aware of the potential threats when using BitTorrent.

4.3 Countermeasures

Alright, let's talk about how to keep yourself safe while torrenting. Yes, torrenting can be risky, but with a few smart moves, you can significantly reduce those risks and protect your privacy. Here are some practical countermeasures you can take.

First up, let's discuss **Using a VPN (Virtual Private Network)**. A VPN is like a secure tunnel for your internet traffic. It hides your IP address and encrypts your data, making it much harder for anyone to see what you're doing online. When you use a VPN while torrenting, your real IP address is hidden, and all the BitTorrent traffic is encrypted. This means that even if someone tries to spy on your activities, all they'll see is encrypted data coming from a VPN server. There are many VPN services out there, so choose one that's reputable and doesn't keep logs of your activity.

Another smart move is to **Use an Anonymous Torrent Client**. Some torrent clients are designed with privacy in mind. For example, you can use a client that supports features like encryption and anonymous mode, which can help keep your activities private. These clients can mask your IP address and make it harder for others to track your downloads.

Next, you should always **Be Cautious with Torrents You Download**. Not all torrents are safe. Some can be loaded with malware that can harm your device. Always download torrents from trusted sources and check the comments and ratings to ensure that other users haven't reported any issues. It's also a good idea to have reliable antivirus software installed to scan any files you download.

Enabling PeerBlock or Similar Software is another layer of protection you can add. PeerBlock is a type of software that blocks connections to and from known bad IP addresses. This can include IP addresses associated with malware, spyware, and other malicious entities. While it won't make you completely anonymous, it adds an extra layer of defense against some of the more common threats.

If you're serious about privacy, you might want to look into **Using Seedboxes**. A seedbox is a remote server that downloads and uploads torrents for you. You access the files from the seedbox via a secure connection, like FTP or SFTP. This means that your IP address is never exposed to the torrent network, as all the torrenting activity happens on the seedbox.

Now, let's not forget about **Regularly Updating Your Software**. Whether it's your torrent client, your operating system, or your antivirus software, keeping everything up to date ensures that you have the latest security patches and features. Many vulnerabilities are exploited because people fail to update their software regularly.

Lastly, you should always **Practice Good Internet Hygiene**. This means using strong, unique passwords for all your accounts, being wary of phishing scams, and not sharing personal information online unless absolutely necessary. The more cautious you are about your overall internet usage, the safer you'll be while torrenting.

In conclusion, while torrenting has its risks, taking these countermeasures can help you stay safe and protect your privacy. Using a VPN, choosing the right torrent client, being careful about what you download, and keeping your software updated are all key steps in reducing the risks associated with torrenting. Stay safe out there!

5. Conclusion

In this report, we have looked into the intricacies of how the BitTorrent protocol works and its vital role in peer-to-peer (P2P) networking. Our exploration began with an overview of the fundamental principles and concepts behind BitTorrent, followed by a detailed examination of its core components such as trackers, metainfo files, seeders, and leechers. We also provided further detail on how the BitTorrent protocol works with an illustrative example scenario to make it clear.

The next chapters zoomed in on BitTorrent's efficiency and reliability through algorithms and techniques. We examined several piece selection algorithms including rarest first and random first piece strategies that are important towards optimizing download performance and ensuring data availability. Additionally, we explored peer selection algorithms focusing mostly on choking/unchoking mechanisms regulating data exchange between peers. Moreover, we explained what swarming is all about as well as end game mode stressing their significance within the context of BitTorrent.

Any P2P network must pay particular attention to security and privacy concerns. This is true for BitTorrent as well where there are such threats as security issues like hacking or legal risks associated with it use together with users' privacy fears that they should know about . For the above reasons, we discussed several security measures that can be used to ensure privacy and enhance security.

One important part of this report is the progress of a small BitTorrent simulator that includes piece selection algorithms and choking/unchoking mechanisms. This simulator serves as a practical example for the theoretical concepts discussed above, offering insights into how BitTorrent operates in a controlled environment.

Looking ahead, there are great possibilities for further development of BitTorrent and P2P networking. Technological advancements and improved network infrastructure can result in more resilient, efficient and secure P2P systems. These may involve better data distribution algorithms, new security protocols for privacy protection of users and sophisticated tools for modeling and optimizing P2P networks.

Finally, we note that the versatility of P2P networking is epitomized by BitTorrent, which provides a way to share files widely over distributed platforms. By understanding how different parts function together with their associated algorithms coupled with considerations on security aspects; we get to appreciate the intricacies involved herein as well as the potential for growth into the future.

References

[1] Bram Cohen, "The BitTorrent Protocol Specification," 2008. Available: https://www.bittorrent.org/beps/bep_0003.html. Accessed: July 22, 2024.

[2] Abhinav C. V., "BitTorrent Part 1: The Engineering Behind the BitTorrent Protocol," Medium, Feb. 2019. Available: https://medium.com/@abhinavcv007/bittorrent-part-1-the-engineering-behind-the-bittorrent-protocol-04e70ee01d Accessed: July 22, 2024.

[3] "How Does BitTorrent Ensure the Security and Privacy of Its Users' Digital Assets?," BYDFI. Available: https://www.bydfi.com/en/questions/how-does-bittorrent-ens Accessed: July 22, 2024.

[4] M. Steiner, T. En-Najjary, and E. W. Biersack, "BitTorrent's Mainline DHT Security Assessment," ResearchGate, 2010. Available: https://www.researchgate.net/publication/50387684_BitTorrent's_Mainline_DHT_Security_Assessment. Accessed: July 22, 2024.

[5] Arpit Bhayani, "The Piece Selection Algorithm that Powers BitTorrent," Substack, Mar. 2021. Available: https://arpit.substack.com/p/the-piece-selection-algori Accessed: July 22, 2024.

[6] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Analyzing Peer Selection Policies for BitTorrent Multimedia On-Demand Streaming Systems in Internet," ResearchGate, 2014. Available: https://www.researchgate.net/publication/260000160_Analyzing_Peer_Selection_Policies_for_BitTorrent_Multimedia_On-Demand_Streaming_Systems_in_Internet. Accessed: July 22, 2024.

[7] "Incentives Build Robustness in BitTorrent," BitTorrent.org. Available: http://bittorrent.org/bittorrentecon.pdf. Accessed: July 22, 2024.

[8] J. Lejeune, P. Sens, and A.-M. Kermarrec, "Exploring the BitTorrent Economy," INRIA, 2004. Available: https://inria.hal.science/inria-00000156/. Accessed: July 22, 2024.