Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

BitTorrent-клиент

Студент ИУ7-72Б	_____	Е.В. Брянская
(Группа)	(Подпись, дата)	(И.О.Фамилия)
Студент ИУ7-72Б	_____	В.А. Иванов
(Группа)	(Подпись, дата)	(И.О.Фамилия)
Руководитель курсового проекта	_____	Н.О. Рогозин
	(Подпись, дата)	(И.О.Фамилия)

2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ7_____
(Индекс)
_____ __И.В. Рудаков_
(И.О.Фамилия)
« _____ » _____ 2021 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине _____Компьютерные сети_____

Студенты группы ___ИУ7-72Б_____

_____Брянская Екатерина Вадимовна_____
(Фамилия, имя, отчество)
_____Иванов Всеволод Алексеевич_____
(Фамилия, имя, отчество)

Тема курсового проекта _____
_____BitTorrent-клиент_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
__учебный_____
Источник тематики (кафедра, предприятие, НИР) ___кафедра_____

График выполнения проекта: 25% к _4_ нед., 50% к _7_ нед., 75% к 11 нед., 100% к _14_ нед.

Задание__ Проанализировать существующие торрент-клиенты. Разработать торрент-клиент на основе протокола BitTorrent с функцией загрузки файлов и поддержкой файлов формата .torrent_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на _20-30__ листах формата А4.
Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую, конструкторскую, технологическую части, заключение, список литературы.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):_____
на защиту работы должна быть предоставлена презентация, состоящая из 15-20 слайдов.____
На слайдах должны быть отражены: постановка задачи, использованные методы и_____
алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс._____

Дата выдачи задания «8» __октября___ 2021 г.

Руководитель курсового проекта _____ __Н.О. Рогозин_____
(Подпись, дата) (И.О.Фамилия)

Студент _____ __Е.В. Брянская_____
(Подпись, дата) (И.О.Фамилия)

Студент _____ __В.А. Иванов_____
(Подпись, дата) (И.О.Фамилия)

Содержание

ВВЕДЕНИЕ

За последние время существенно возросли объёмы информации, передаваемой по сети Интернет. Очевидно, что подобная тенденция сохранится и в будущем – будет расти число пользователей и объём потребляемого ими трафика.

В подобных условиях актуальным является вопрос производительности серверов. Ввиду описанных выше факторов нагрузка на них будет постоянно расти, что будет вынуждать их владельцев производить их обновление и расширение или снижение скорости обмена информацией с клиентами.

Последнее является чувствительным для загрузки файлов больших объёмов. Решением в таком случае может быть кооперативный обмен файлами. Наиболее популярным протоколом для этой технологии является Bittorrent.

Целью данной работы является разработка Bittorrent клиента.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1) изучить структуру и принцип работы протокола;
2) разработать алгоритм взаимодействия с сервером и клиентами;
3) реализовать программу для загрузки файлов на основе протокола Bittorrent.

1 Аналитическая часть

1.1 Постановка задачи

Результатом работы должна стать программа для загрузки файлов по протоколу Bittorrent, удовлетворяющая следующим требованиям:

- поддерживать файлы расширения .torrent;
- поддерживать функцию загрузки данных как от сервера, так и от других клиентов;
- обладать графическим интерфейсом для удобства выполнения действий и просмотра текущей информации по состоянию загрузки.

Первостепенной задачей для дальнейшей разработки является изучения устройства выбранного протокола.

1.2 Принцип работы протокола

Bittorrent – P2P протокол для кооперативного обмена файлами через интернет [1, 2].

В данном протоколе выделены две роли:

1) **пир** (клиент) хранит файлы и производит обмен их частями с другими пирами;
2) **трекер** (сервер) хранит таблицу файлов и список пиров, имеющих данный файл в распоряжении.

Пир, желающий получить файл должен обладать **.torrent файлом**, с помощью которого он может обратиться к серверу. Сервер предоставляет адреса клиентов, обладающих запрашиваемыми файлами после чего начинается их загрузка. Передача осуществляется частями (**pieces**), каждый torrent-клиент, скачивая эти части, в то же время отдаёт их другим клиентам, что снижает нагрузку на каждого отдельного клиента (Рисунок 1.1) [2, 3].

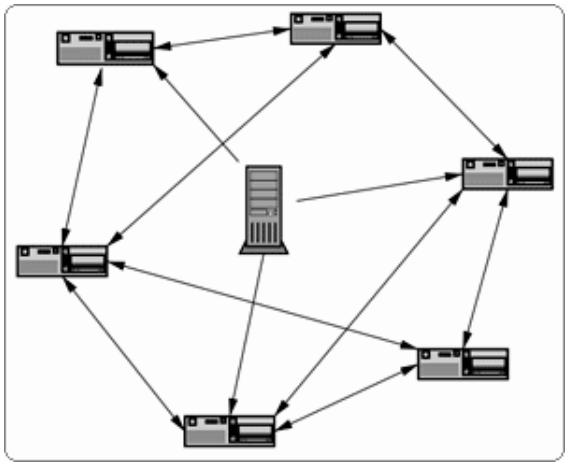

Рисунок 1.1 – Схема взаимодействия клиентов и сервера

1.3 Структура .torrent файла

Как было отмечено выше, первым шагом в начале загрузки является получение и парсинг файла специального расширения .torrent.

Для кодирования данных в .torrent-файлах используется формат Bencode. Само содержимое – ассоциативный массив с полями:

- **info** – вложенный ассоциативный массив который описывает файлы, передаваемые торрентом;
- **announce** – URL трекера;
- **announce-list** – список трекеров, если их несколько, в Bencode-виде — список списков;
- **creation date** – дата создания;
- **comment** – текстовое описание торрента;
- **created by** – автор торрента.

info и announce являются обязательными полями, всё остальные — опционально. Первый в свою очередь состоит из:

- **piece length** – размер одного куска;
- **pieces** – конкатенация SHA1-хешей каждого куска (каждый хеш - 20 байт);
- **name** – имя файла (если файл один);

- **length** – содержит длину файла (если файл один);
- **files** – если файлов несколько, то содержит список ассоциативных массивов (с указанием length и path).

Данная информация используется на всём протяжении загрузки файла и его последующей раздаче.

1.4 Взаимодействие клиента и сервера

Чтобы перейти к загрузке файла клиент должен получить список пиров у трекера. Для этого он должен отправить GET-запрос, называемый **анонсом**, по адресу announce по пути /announce.

После данного действия трекер узнаёт о наличии нового клиента и может выдать его адрес другим клиентам. Указываются следующие URL-параметры.

- **info_hash** – SHA1-хеш словаря info. Используется для поиска файла в таблице трекера, то есть фактически является его уникальным идентификатором.
- **peer_id** – уникальный ID клиента. Имеет вид -*<2-символьный id><номер версии из 4 цифр>-<12 случайных цифр>*. Такой код может быть сгенерирован клиентом самостоятельно, так как вероятность коллизии с другими клиентами крайне мала (число возможных вариантов peer_id одной версии превышает количество IPv4 адресов более чем в 200 раз).
- **uploaded**, **downloaded**, **left** – количество отправленных, загруженных и незагруженных байтов.
- **port** – TCP-порт, прослушиваемый клиентом. Общепринятыми значениями являются 6881-6889.
- **compact** – признак того, принимает ли клиент компактный список пиров.

В случае, если запрос прошёл успешно и по info_hash был найден необходимый torrent, трекер посылает ответ (также по протоколу HTTP). В его теле

содержится следующие поля в формате Bencode:

- **interval** – интервал в секундах до того, как клиент должен сделать новый запрос к трекеру;
- **peers** – список пиров. В случае, если в запросе compact был равен 1, в ответе будет список будет заменён бинарной строкой, которую потребуется разбить на группы по 6 байт для выделения IPv4 адреса и порта каждого пира.

Подобные запросы будут повторяться раз в interval секунд для поддержания сервера в курсе актуального состояния загрузки и для получения новых адресов пиров.

1.5 Структура сообщений

Протокол BitTorrent определяет следующий способ обмена сообщениями для клиентов, его особенности:

- использует стек TCP/IP;
- файл передаётся по кускам фиксированного размера, не в порядке их следования в файле.

Определена следующая структура p2p сообщения:

1) **длина**, Len (4Б) – размер типа и полезной нагрузки сообщения;
2) **тип**, ID (1Б) определяет вид сообщения и способ его обработки;
3) **полезная нагрузка**, Payload (0 - 32КБ) содержит передаваемую информацию.

Различаются следующие типы сообщений.

- **handshake**: <len=49+X><info_hash><peer_id>. Сообщение рукопожатия. Отправляется один раз в начале обмена информацией. Содержит название протокола, хеш код файла и собственный id.

- **keep-alive**: <len=0000>. Содержит только нулевую длину. Используется чтобы один из пиров не закрыл соединение по истечению времени без сообщений.
- **choke**: <len=0001><id=0>. Используется для запрета другому пиру отправки сообщений до момента посылки ему unchoke.
- **unchoke**: <len=0001><id=1>.
- **interested**: <len=0001><id=2>. Состояние говорит о том, что пир заинтересован в получении фрагментов.
- **not interested**: <len=0001><id=3>. Обратно interested.
- **have**: <len=0005><id=4>. Сообщает о появлении в своём распоряжении куска с указанным индексом.
- **bitfield**: <len=0001+X><id=5>. Содержит в себе карту битов, описывающую статус всех кусков файла.
- **request**: <len=0013><id=6>. Используется для запроса блока байт размером length, начинающимся с позиции begin из куска с номером index.
- **piece**: <len=0009+X><id=7>. Сообщение содержит в себе блок байт block по формату, описанному в request.

1.6 Взаимодействие клиентов

Первым шагом после получения адреса пира требуется выполнения ”рукопожатия”(handshake). Он нужен для обмена id и проверкой совпадения протоколов и контрольного хеша файла. В случае неудачного рукопожатия TCP соединение разрывается.

После рукопожатия устанавливается состояние Choked. Для выхода из него сразу отправляется сообщение Interested для перехода к обмену.

В первую очередь после взаимной заинтересованности пиры обмениваться информацией о наличии кусков с помощью сообщения bitfield. Это требуется для определения отсутствующих кусков, которые можно запросить у данного пира.

В тот момент, когда клиент может запросить кусок (т.е. не находится уже в состоянии загрузки с данным пиром, не является choked и not interested), он выбирает блок для запроса у данного пира. Приоритет выбора следующий [5]:

1) блоки, для которых истекло время ожидания;
2) блоки из неполностью загруженных кусков;
3) блоки из наиболее редких кусков.

Блоки будут отсутствовать во всех перечисленных категориях только в случае, если загрузка почти полностью завершена. Такая ситуация называется **end-game**. В этом случае в качестве очередных блоков для запроса выбираются уже загружаемые блоки.

После получения блока (сообщения piece), он записывается с указанным смещением в нужный кусок. По окончанию загрузки куска подсчитывается его контрольная сумма и сравнивается с той, которая изначально хранилась в torrent файле. Если они совпали, кусок помечается загруженным и сохраняется в загружаемый файл, а всем хостам без данного куска отправляется сообщение Have с его номером.

Файл считается загруженным полностью кода скачены и проверены все его куски.

Вывод

Результатом аналитического раздела стал анализ устройства протокола BitTorrent, алгоритма взаимодействия с сервером и другими пирами.

2 Конструкторская часть

2.1 Основной алгоритм

Протокол BitTorrent предусматривает, что действия обмена информации с трекером и каждым из пиров производятся асинхронно, что и позволяет достичь такой высокой скорости загрузки. Поэтому в начале основного цикла программы создаётся заранее установленное количество обработчиков, каждый из которых выполняет асинхронное общение со своим пиром.

На Рисунке 2.1 приведена схема этого алгоритма.

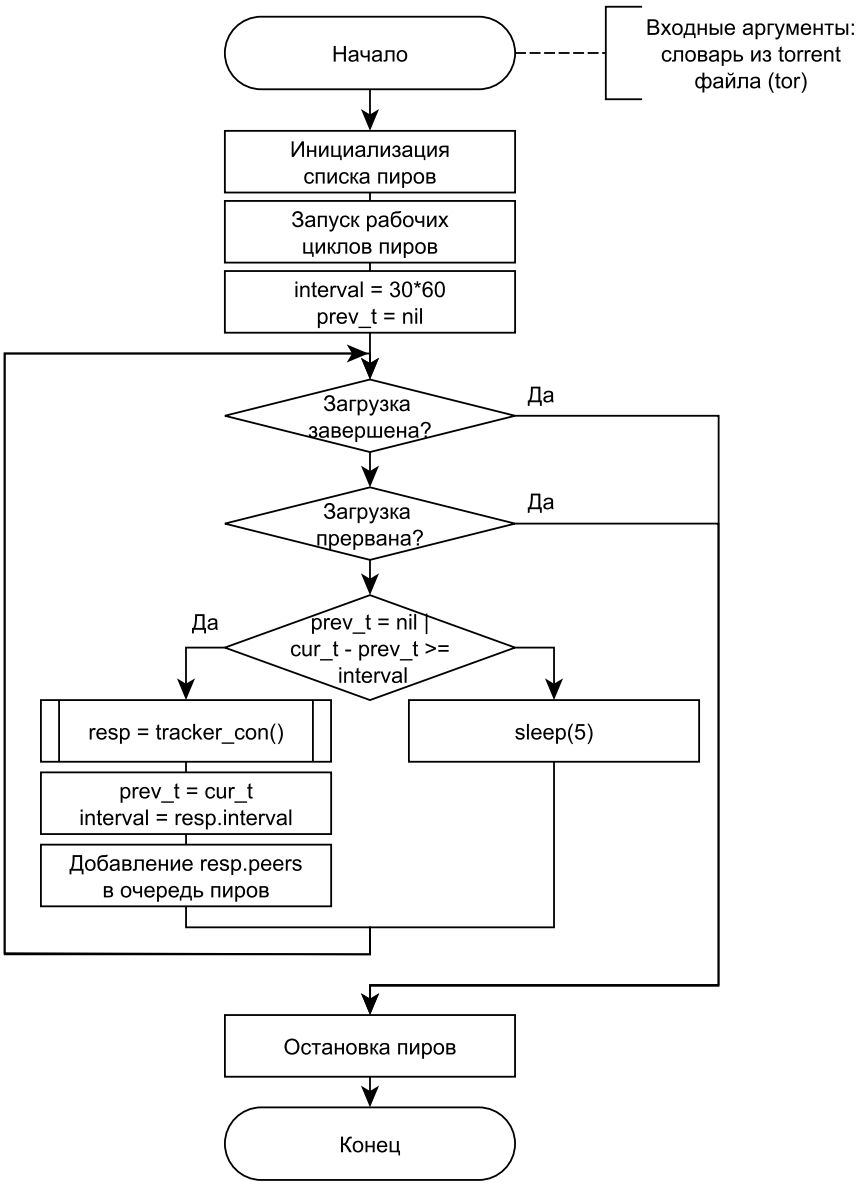

Рисунок 2.1 – Основной алгоритм

2.2 Алгоритм взаимодействия с сервером

Детали алгоритма взаимодействия с сервером продемонстрированы на схеме 2.2.

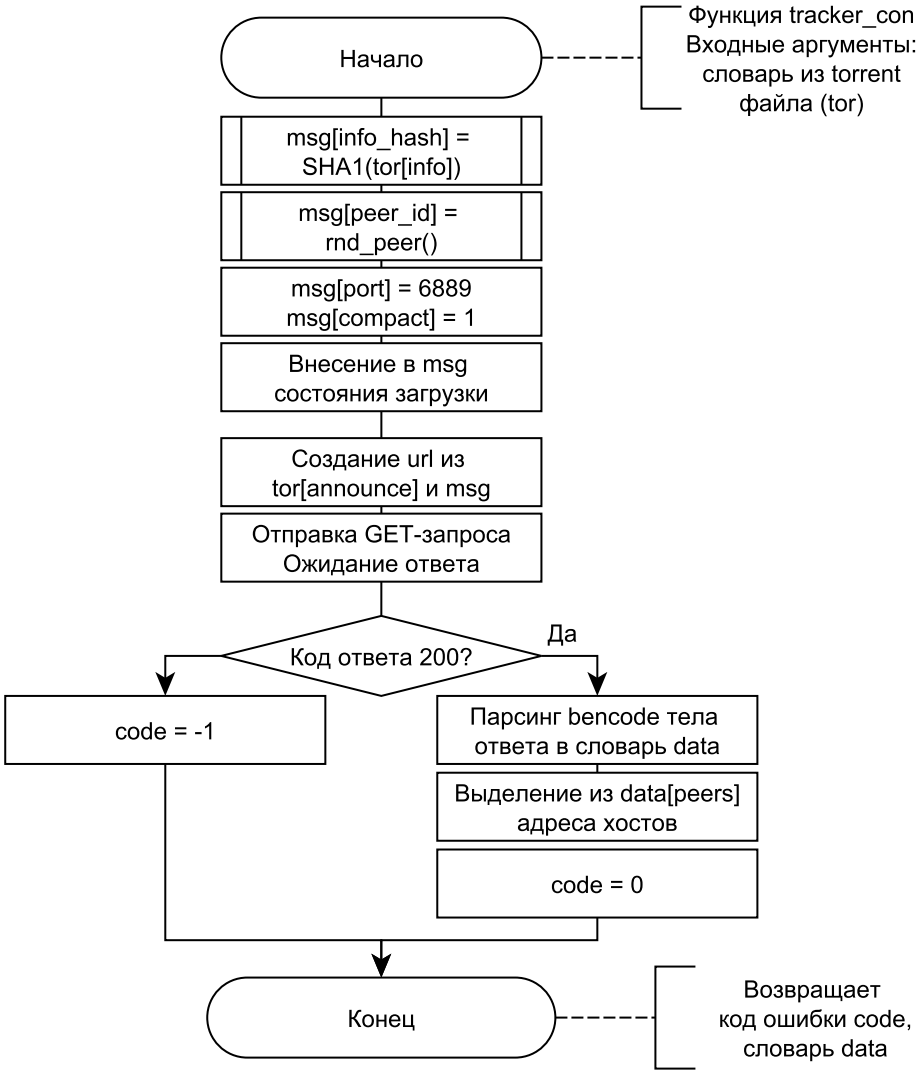

Рисунок 2.2 – Алгоритм взаимодействия с сервером

2.3 Алгоритм рукопожатия

Этот алгоритм приведён на Рисунке 2.3.

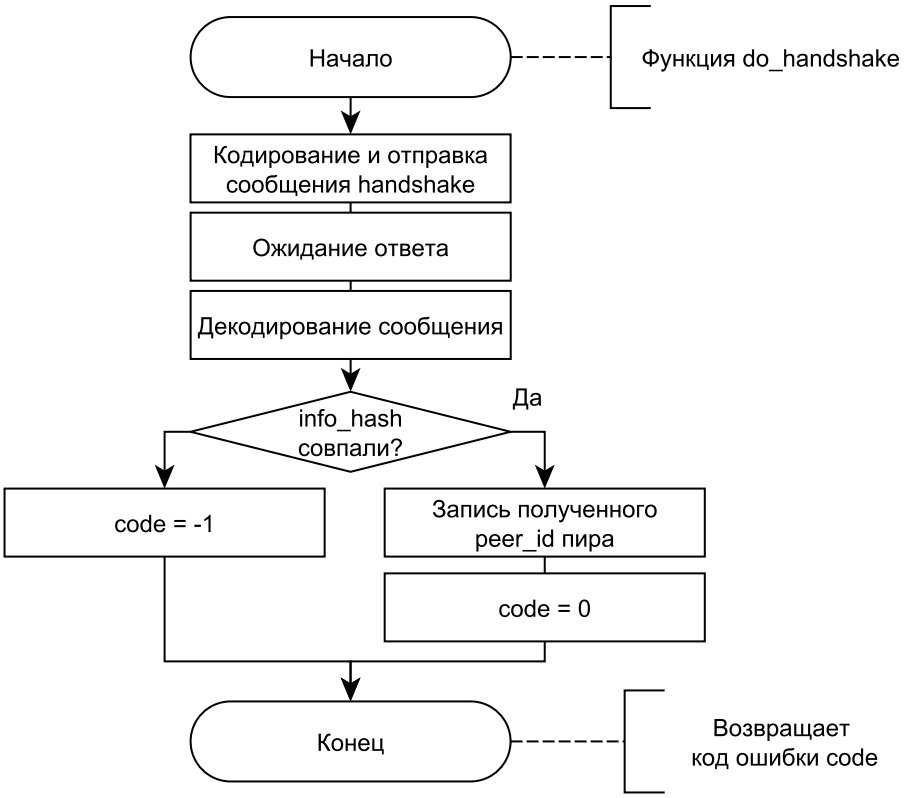

Рисунок 2.3 – Алгоритм рукопожатия

2.4 Алгоритм взаимодействия с пирами

Детали взаимодействия с пирами приведены ниже, на Рисунке 2.4.

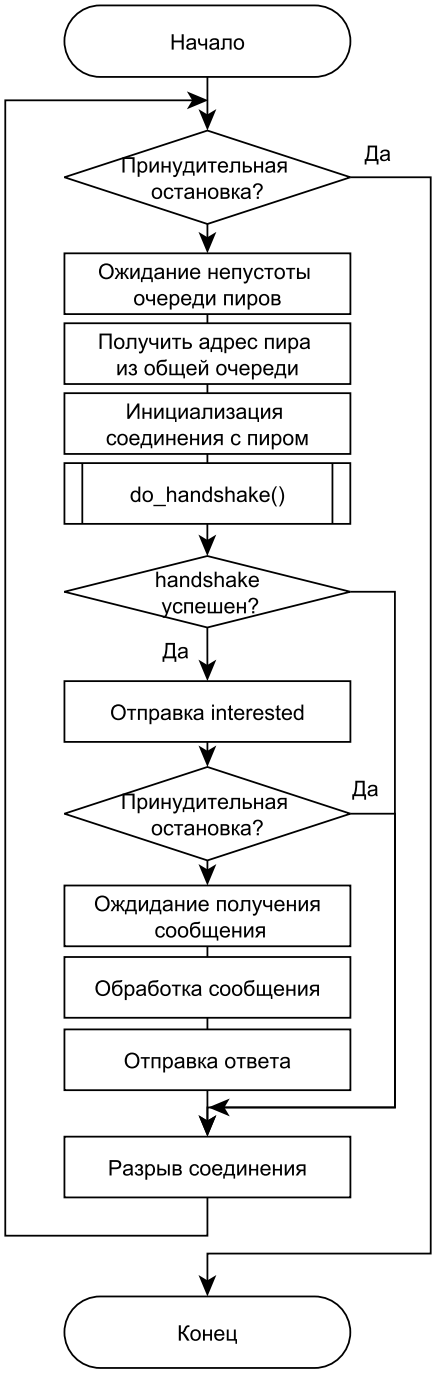

Рисунок 2.4 – Алгоритм взаимодействия с пирами

3 Технологическая часть

3.1 Выбор технологических средств

В качестве языка программирования был выбран Python [6], поскольку он предоставляет множество необходимых для реализации поставленной задачи библиотек, такие как aiohttp, socket, bencodepy и прочие, а также ввиду имеющегося опыта работы с этим языком.

Была выбрана среда разработки PyCharm [7], поскольку она бесплатна для студентов и хороша знакома, так как активно использовалась в процессе обучения.

Для создания удобного, интуитивно понятного интерфейса использовался набор библиотек PyQt5 [8].

3.2 UML диаграмма классов

На Рисунках 3.5-3.6 приведена UML-диаграмма основных разработанных классов. На диаграмме 3.6 приведены все виды сообщений, которыми могут обмениваться участники процесса скачивания.

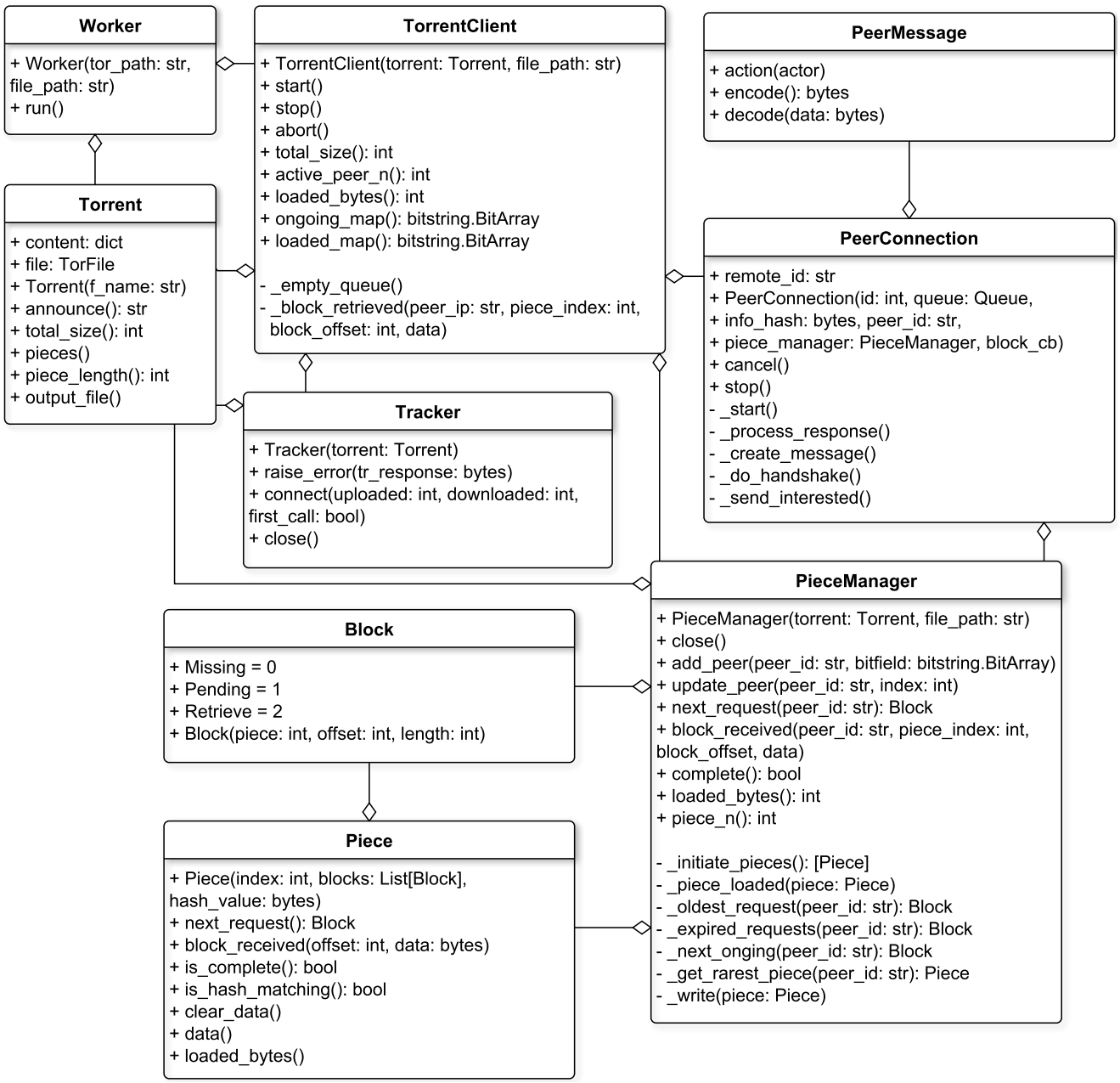

Рисунок 3.5 – UML-диаграмма классов

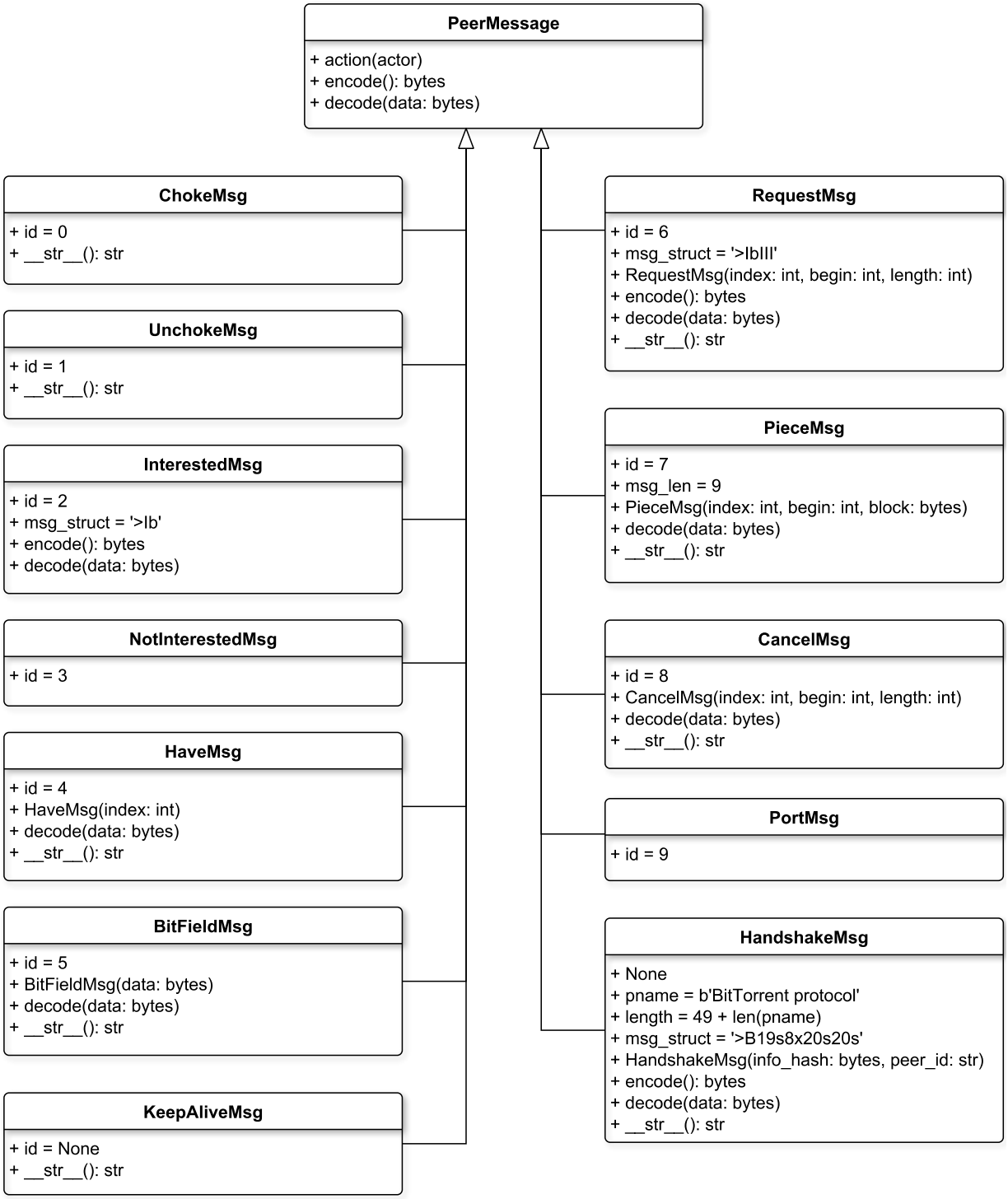

Рисунок 3.6 – UML-диаграмма классов сообщений

3.3 Описание функций классов

В Приложении А приведены листинги реализаций основных классов.

Кратко опишем функции, выполняемые реализованными классами.

- **TorrentClient**. Класс содержащий основную информацию о скачиваемом

торренте. Метод start реализует главный выполняемый цикл.

- **Tracker**. Отвечает за обмен информацией с сервером и её декодирование.
- **PeerConnection**. Класс отвечающий за обмен информации с другими пирами. Метод run содержит цикл передачи сообщений.
- **PeerMessage**. Класс хранящий информацию сообщения, пересылаемого между пирами, осуществляет его кодировку и декодирование.
- **Torrent**. Класс считывающий и хранящий метаинформацию .torrent файла.
- **Piece**. Класс хранит информацию об одной из частей файла. Отвечает за его составление (с помощью класса Block) и проверки целостности.
- **PieceManager**. Класс хранящий все части файла. Отвечает за их сохранение в файл и определение последовательности их загрузки.

3.4 Интерфейс программы

На Рисунках 3.7-3.8 представлен разработанный графический интерфейс. Для того, чтобы начать процесс скачивания, необходимо сначала указать путь до .torrent файла и выбрать директорию загрузки. Нажатие кнопки «Старт» запустит процесс.

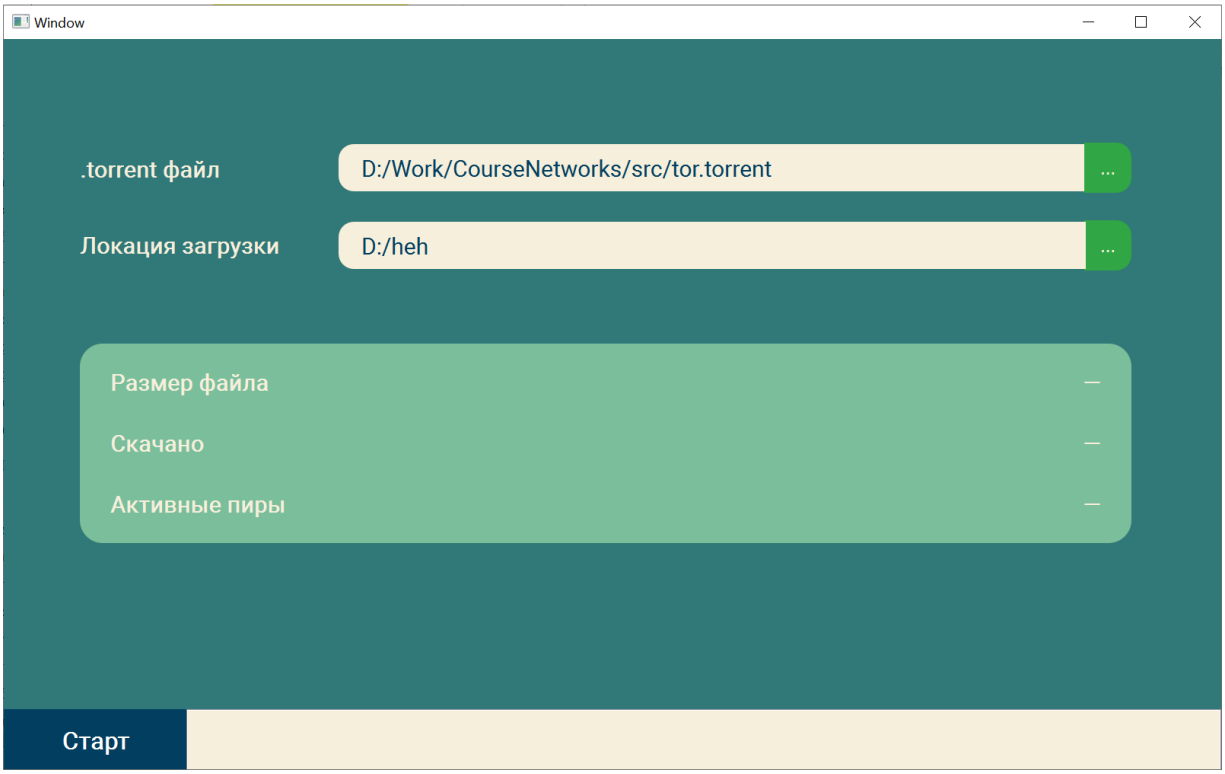

Рисунок 3.7 – Графический интерфейс до начала скачивания

В процессе скачивания необходимого файла на экран выводится статистика: размер файла в килобайтах, успешно полученный объём (килобайты) и число активных пиров, с которыми на данный момент происходит взаимодействие.

Для наглядности снизу была добавлена специальная шкала, в которой жёлтым цветом помечаются куски (pieces), находящиеся в процессе скачивания, зелёным – успешно полученные. В случае полной загрузки файла, вся полоса будет покрашена в зелёный цвет.

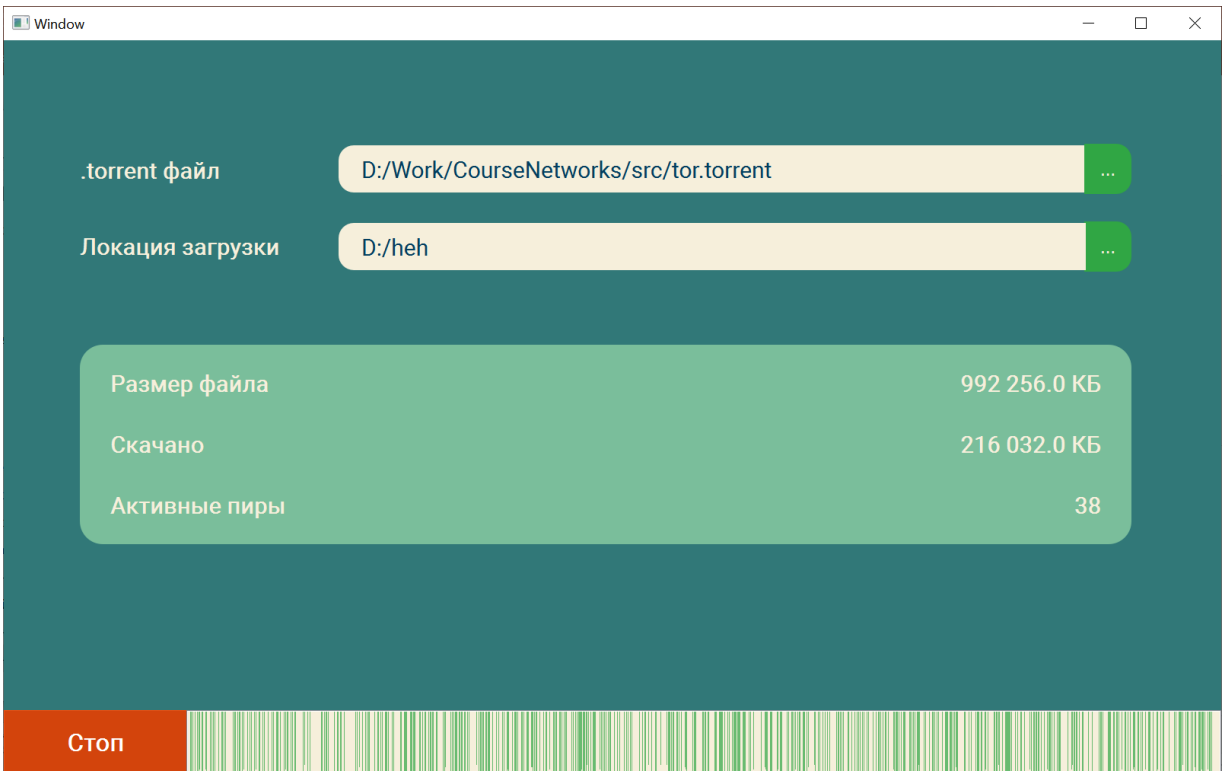

Рисунок 3.8 – Графический интерфейс в процессе скачивания

Для того, чтобы прервать операцию, достаточно нажать на кнопку «Стоп».

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была достигнута поставленная цель – был разработан Bittorrent клиент.

Выполнены все поставленные задачи:

1) изучена структура и принцип работы протокола: в деталях разобраны структуры .torrent файлов и анонса, рассмотрены типы p2p сообщений;
2) разработаны алгоритмы взаимодействия с сервером и клиентами;
3) реализована программа для загрузки файлов на основе протокола Bittorrent, а также удобный графический интерфейс, позволяющий отслеживать актуальную информацию о работе торрент-клиента.

В качестве дальнейших способов развития клиента можно выделить следующее:

- реализация асинхронного принципа работы менеджера частей файла для повышения производительности;
- поддержка получения информации о пирах с помощью распределённой хэш-таблицы;
- поддержка magnet-ссылок для получении информации о файлах (как альтернатива .torrent файлу).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Список литературы

1. The Basics of BitTorrent [Электронный ресурс]. – Режим доступа: https://documentation.help/uTorrent/Chapter02_01.html (дата обращения 10.10.2021).

2. Bittorrent Protocol Specification v1.0 [Электронный ресурс]. – Режим доступа: https://wiki.theory.org/BitTorrentSpecification (дата обращения 10.10.2021).

3. Arnaud Legout, Guillaume Urvoy-Keller, Pietro Michiardi. Understanding BitTorrent: An Experimental Perspective. [Technical Report] – 2005 – pp.16. – ffinria-00000156v3.

4. Hari Balakrishnan, M. FransKaashoek , David Karger, Robert Morris, and Ion Stoica. Looking up DATA in P2P systems. In Proc. Acm SIGCOMM’01, San Diego, CA, Aug. 2001.

5. Raymond Lei Xia, A Survey of BitTorrent Performance / Raymond Lei Xia, Jogesh K. Muppala. – IEEE COMMUNICATIONS SURVEYS & TUTORIALS – VOL. 12 – NO. 2, SECOND QUARTER – 2010 – p.140-158.

6. Python 3.7.12 documentation [Электронный ресурс]. – Режим доступа: https://docs.python.org/3.7/ (дата обращения 12.10.2021).

7. Руководство PyCharm [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/ru-ru/pycharm/learn/ (дата обращения 12.10.2021).

8. PyQt5 Reference Guide [Электронный ресурс]. – Режим доступа: https://www.riverbankcomputing.com/static/Docs/PyQt5/ (дата обращения 22.10.2021).

ПРИЛОЖЕНИЕ А

Листинг 1: Класс TorrentClient

```
class TorrentClient:
    aborted = False

    def __init__(self, torrent: Torrent, file_path: str):
        self.tracker = Tracker(torrent)
        self.available_peers = Queue()
        self.peers: List[PeerConnection] = []
        self.piece_manager = PieceManager(torrent, file_path)

    async def start(self):
        self.peers = [PeerConnection(i, self.available_peers, self.tracker.
    torrent.info_hash, self.tracker.peer_id, self.piece_manager, self.
    _block_retrieved)
                for i in range(MAX_PEER_CONNECTIONS)]

        previous = None
        interval = 30 * 60

        while True:
            if self.piece_manager.complete:
                print('Torrent fully downloaded!')
                logging.info('Torrent fully downloaded!')
                break
            if self.aborted:
                logging.info("torrent is aborted")
                break

        cur_time = time.time()
        if previous is None or cur_time - previous >= interval:
            response = await self.tracker.connect(uploaded=self.piece_manager
    .uloaded_bytes, downloaded=self.piece_manager.loaded_bytes, first_call=
    previous if previous else False)
            previous = cur_time
            interval = response.interval

        if not response:
```

```
            continue

        self._empty_queue()
        for peer in response.peers:
            self.available_peers.put_nowait(peer)
        else:
            await sleep(5)

        await self.stop()

    async def stop(self) -> None:
        self.aborted = True

        for peer in self.peers:
            peer.stop()

        await self.tracker.close()

    def abort(self):
        self.aborted = True

    def _empty_queue(self) -> None:
        while not self.available_peers.empty():
            self.available_peers.get_nowait()

    def _block_retrieved(self, peer_id: str, piece_index: int, block_offset,
data) -> None:
        self.piece_manager.block_received(peer_id, piece_index, block_offset,
 data)

    @property
    def total_size(self) -> int:
        return self.tracker.torrent.total_size

    @property
    def active_peer_n(self) -> int:
        n = sum(1 if peer.remote_id is not None
                else 0
                for peer in self.peers)
        return n
```

```
71
72      @property
73      def loaded_bytes(self) -> int:
74          return self.piece_manager.loaded_bytes
75
76      @property
77      def ongoing_map(self) -> bitstring.BitArray:
78          return self.piece_manager.ongoing_map
79
80      @property
81      def loaded_map(self) -> bitstring.BitArray:
82          return self.piece_manager.piecemap
```

Листинг 2: Класс Tracker

```
class Tracker:
    def __init__(self, torrent: Torrent):
        self.torrent = torrent
        self.peer_id = _get_peer_id()
        self.http_client = aiohttp.ClientSession()

    async def connect(self, uploaded: int = 0, downloaded: int = 0,
    first_call: bool = None):
        params = {
            'info_hash': self.torrent.info_hash,
            'peer_id': self.peer_id,
            'port': PORT,
            'uploaded': uploaded,
            'downloaded': downloaded,
            'left': self.torrent.total_size - downloaded,
            'compact': 1
        }

        if first_call:
            params['event'] = 'started'
        try:
            url = self.torrent.announce + '?' + urlencode(params)
            logging.info('connecting to tracker. URL: ' + url)

        async with self.http_client.get(url, ssl=False) as response:
            if response.status != 200:
                raise ConnectionError("ERROR: connection to tracker failed.
    Code = " + str(response.status))

            data = await response.read()
            self.raise_error(data)

            return TrackerResponse(bencodepy.decode(data))
    except Exception as exp:
        print(exp)

    async def close(self):
        await self.http_client.close()
```

Листинг 3: Класс TorrentClient

```
class PeerConnection:
    remote_id = None
    writer: asyncio.StreamWriter = None
    reader: asyncio.StreamReader = None

        def __init__(self, id: int, queue: Queue, info_hash: bytes, peer_id: str,
        piece_manager: PieceManager, block_cb: Callable[[str, int, int, bytes], None]):
        self.state = PeerState()
        self.peer_state = OtherPeerState()

        self.id = id
        self.queue = queue
        self.info_hash = info_hash
        self.peer_id = peer_id
        self.piece_manager = piece_manager
        self.block_cb = block_cb  # Callback function. It is called when block is received from the remote peer.
        self.future = asyncio.ensure_future(self._start())

    async def _start(self):
        while not self.state.is_stopped:
            ip, port = await self.queue.get()
            self._info("assigned peer, ip = {}".format(ip))

            try:
                self.reader, self.writer = await asyncio.open_connection(ip, port)
                self._info("connection was opened, ip = " + ip)

                buf = await self._do_handshake()
                await self._send_interested()

                async for msg in PeerStreamIterator(self.reader, buf):
                    self._info("received {}".format(str(msg)))
                    if self.state.is_stopped:
                        break

```

```
                    await self._process_response(msg)
                    await self._create_message()

            except concurrent.futures._base.CancelledError as exp:
                self._exep(exp)
                await self.cancel()
                raise exp

            except Exception as exp:
                self._exep(exp)
                await self.cancel()
                raise exp
            self._info('Out of loop')
            await self.cancel()

    async def _process_response(self, msg: PeerMessage):
        if type(msg) is BitFieldMsg:
            msg: BitFieldMsg
            self.piece_manager.add_peer(self.remote_id, msg.bitfield)
        elif type(msg) is InterestedMsg:
            self.peer_state.interest()
        elif type(msg) is NotInterestedMsg:
            self.peer_state.uninterest()
        elif type(msg) is ChokeMsg:
            self.state.choke()
        elif type(msg) is UnchokeMsg:
            self.state.unchoke()
        elif type(msg) is HaveMsg:
            msg: HaveMsg
            self.piece_manager.update_peer(self.remote_id, msg.index)
        elif type(msg) is KeepAliveMsg:
            pass
        elif type(msg) is PieceMsg:
            msg: PieceMsg
            self.state.stop_pending()
            self.block_cb(self.remote_id, msg.index, msg.begin, msg.block)
        elif type(msg) is RequestMsg:
            pass
        elif type(msg) is CancelMsg:
            pass
```

```
76
77      async def _create_message(self):
78          if (not self.state.is_choked) and self.state.is_interested and (not
    self.state.is_pending):
79              self.state.start_pending()
80              success = await self._request_piece()
81              if not success:
82                  self.state.stop_pending()
83
84      async def cancel(self):
85          self._info('closing peer {ip}'.format(ip=self.remote_id))
86          self.remote_id = None
87          if not self.future.done():
88              pass
89          if self.writer:
90              self.writer.close()
91              await self.writer.wait_closed()
92
93          self.queue.task_done()
94
95      def stop(self) -> None:
96          self.state.stop()
97          if not self.future.done():
98              self.future.cancel()
99
100     async def _do_handshake(self) -> bytes:
101         self.writer.write(HandshakeMsg(self.info_hash, self.peer_id).encode()
    )
102         await self.writer.drain()
103
104         buf = b''
105         tries = 0
106         while len(buf) < HandshakeMsg.length and tries < 10:
107             tries += 1
108             buf += await self.reader.read(PeerStreamIterator.CHUNK_SIZE)
109
110         response = HandshakeMsg.decode(buf)
111         if response is None:
112             raise Exception()
113         elif response.info_hash != self.info_hash:
```

```
114             raise Exception()
115
116         self.remote_id = response.peer_id
117         self._debug('handshake with peer {} are successful'.format(self.
    remote_id))
118
119         return buf[HandshakeMsg.length:]
120
121     async def _send_interested(self) -> None:
122         msg = InterestedMsg()
123         self.writer.write(msg.encode())
124         await self.writer.drain()
125         self._debug('sending interested msg to {}'.format(self.remote_id))
126
127     async def _request_piece(self) -> bool:
128         block = self.piece_manager.next_request(self.remote_id)
129         if block is None:
130             return False
131
132         msg = RequestMsg(block.piece, block.offset, block.length).encode()
133
134         self._debug('Requesting block {} of piece {} from {}'.
135     format(block.offset, block.piece, self.remote_id))
136
137         self.writer.write(msg)
138         await self.writer.drain()
139         return True
140
```