Санкт-Петербургский политехнический университет Петра Великого
Физико-механический институт
Высшая школа прикладной математики и вычислительной физики

Компьютерные сети
Отчёт по лабораторной работе №3
“Bit-torrent. Клиент и Трекер”

Выполнила
студентка группы 5040102/30201 Л.Р. Юнусова
Принял
к. ф.-м. н., доцент А.Н. Баженов

Санкт-Петербург 2025

Содержание

1. Постановка задачи

Необходимо реализовать клиент и трекер на языке python, реализующие упрощенную систему вида bit-torrent.

Пусть дано n узлов (нод), каждый из которых может находиться в одном из двух состояний: **отправитель** (upload) или **получатель** (download). Каждый узел может владеть набором файлов, которые он может передавать другим узлам или запрашивать файлы у других узлов. В системе также присутствует **трекер**, который управляет информацией о файлах и узлах, участвующих в распределении файлов.

Каждый узел в начале работы имеет набор файлов, которые он может передавать другим узлам. Трекер хранит информацию о том, какие файлы доступны в системе и какие узлы владеют этими файлами. Узлы могут запрашивать файлы у других узлов, и трекер помогает им найти подходящие узлы для загрузки.

Требуется разработать протокол взаимодействия между узлами и трекером, который позволит:

1. Узлам регистрироваться в системе и сообщать трекеру о своих файлах.
2. Узлам запрашивать файлы у других узлов и загружать их.
3. Узлам передавать файлы другим узлам по запросу.
4. Трекеру управлять информацией о файлах и узлах, а также обновлять её в реальном времени.

Протокол должен обеспечивать следующие свойства:

- **Согласованность данных**: Все узлы должны иметь одинаковую информацию о доступных файлах и их владельцах.
- **Надёжность передачи**: Каждый узел должен быть способен загрузить файл, даже если некоторые узлы могут вести себя некорректно.
- **Параллельность**: Узлы должны иметь возможность одновременно загружать и передавать файлы, используя многопоточность.

Описание протокола

Протокол взаимодействия между узлами и трекером состоит из нескольких этапов:

1. **Регистрация узла в системе**:
    - Узел отправляет сообщение трекеру с типом `REGISTER`, чтобы зарегистрироваться в системе.

- Трекер обновляет свою базу данных, добавляя информацию о новом узле.

2. **Передача файлов (режим отправки)**:
   - Узел, который хочет передать файл, отправляет сообщение трекеру с типом `OWN`, указывая имя файла.
   - Трекер добавляет информацию о том, что данный узел владеет указанным файлом.
   - Узел переходит в режим ожидания запросов от других узлов на передачу файла.
3. **Запрос файлов (режим загрузки)**:
   - Узел, который хочет загрузить файл, отправляет сообщение трекеру с типом `NEED`, указывая имя файла.
   - Трекер ищет узлы, которые владеют этим файлом, и возвращает список подходящих узлов.
   - Узел запрашивает файл у выбранных узлов, разделяя файл на части и загружая их параллельно.
4. **Обновление информации о частоте передачи**:
   - После успешной передачи файла узел отправляет сообщение трекеру с типом `UPDATE`, чтобы обновить информацию о частоте передачи файлов.
   - Трекер увеличивает счётчик частоты передачи для данного узла.
5. **Выход из системы**:
   - Узел, который хочет покинуть систему, отправляет сообщение трекеру с типом `EXIT`.
   - Трекер удаляет информацию о данном узле из своей базы данных.

Реализация протокола

2. Теория

Техническая реализация BitTorrent

BitTorrent — это протокол для распределённого обмена файлами через Интернет. Он был разработан Брэмом Коэном в 2001 году и с тех пор стал одним из самых популярных способов передачи больших файлов. Основная идея BitTorrent заключается в том, что файлы передаются не с одного сервера, а распределяются между множеством пользователей (пиров), что позволяет значительно снизить нагрузку на отдельные узлы и увеличить скорость загрузки.

Структура сообщений в BitTorrent

Взаимодействие между пирами и трекером в BitTorrent осуществляется через обмен сообщениями. Эти сообщения передаются по протоколу TCP или UDP и имеют определённую структуру. Основные типы сообщений включают:

- **Handshake**:
  - Первое сообщение, которое отправляется при установлении соединения между пирами. Оно содержит идентификатор протокола, хэш-сумму торрент-файла и идентификатор пира.
- **Keep-Alive**:
  - Сообщение, которое отправляется для поддержания соединения, если нет других данных для передачи.
- **Choke/Unchoke**:
  - Сообщения, которые указывают, может ли пир отправлять данные. **Choke** означает, что пир временно не может отправлять данные, а **Unchoke** — что он готов к передаче.
- **Interested/Not Interested**:
  - Сообщения, которые указывают, заинтересован ли пир в получении данных от другого пира. **Interested** означает, что пир хочет получить данные, а **Not Interested** — что он не заинтересован.
- **Request**:
  - Сообщение, которое запрашивает определённый блок данных (часть файла) у другого пира.
- **Piece**:
  - Сообщение, которое содержит запрошенный блок данных.
- **Have**:
  - Сообщение, которое информирует других пиров о том, что у данного пира появился новый блок данных.

Алгоритмы выбора пиров

BitTorrent использует несколько алгоритмов для оптимизации выбора пиров и управления передачей данных:

- **Алгоритм "Tit-for-Tat"**:
  - Этот алгоритм основан на принципе взаимности. Пиры отдают предпочтение тем пирам, которые сами делятся с ними данными. Если пир не делится данными, другие пиры могут "задушить"его (отправить сообщение **Choke**), чтобы ограничить его доступ к данным.
- **Алгоритм "Rarest First"**:
  - Этот алгоритм определяет, какие блоки данных следует загружать в первую очередь. Пиры стараются загружать те блоки, которые есть у наименьшего числа других пиров. Это помогает равномерно распределить части файла среди всех участников.
- **Оптимизация скорости**:
  - Пиры выбирают для загрузки те блоки, которые можно скачать с наибольшей скоростью. Это позволяет минимизировать время загрузки.

Управление соединениями

BitTorrent использует TCP-соединения для передачи данных между пирами. Каждое соединение управляется следующим образом:

- **Установление соединения**:
  - Перед началом передачи данных пиры обмениваются сообщением **Handshake**, чтобы подтвердить совместимость и идентифицировать друг друга.
- **Поддержание соединения**:
  - Если в течение определённого времени нет данных для передачи, пиры отправляют сообщение **Keep-Alive**, чтобы поддерживать соединение активным.
- **Управление потоком данных**:
  - Пиры используют сообщения **Choke**/**Unchoke** и **Interested**/**Not Interested** для управления потоком данных. Если пир не может или не хочет отправлять данные, он отправляет сообщение **Choke**.

Обработка данных

BitTorrent разбивает файлы на небольшие блоки (обычно размером 256 КБ), которые передаются между пирами. Каждый блок проверяется на целостность с помощью хэш-сумм, которые хранятся в торрент-файле. Это позволяет убедиться, что данные не были повреждены во время передачи.

- **Разбиение файла на блоки**:
  - Файл разбивается на блоки фиксированного размера, и для каждого блока вычисляется хэш-сумма. Эти хэш-суммы хранятся в торрент-файле.
- **Проверка целостности данных**:
  - После загрузки блока данных пир проверяет его целостность, сравнивая хэш-сумму с той, которая указана в торрент-файле. Если хэш-суммы не совпадают, блок считается повреждённым и загружается заново.
- **Сборка файла**:
  - После загрузки всех блоков пир собирает их в один файл, используя информацию из торрент-файла.

Роль трекера и DHT

Трекер играет ключевую роль в BitTorrent, предоставляя информацию о пирах, участвующих в раздаче. Однако в современных реализациях BitTorrent также используется распределённая хэш-таблица (DHT), которая позволяет пирам находить друг друга без участия централизованного трекера.

- **Трекер**:
  - Трекер хранит информацию о всех пирах, участвующих в раздаче, и предоставляет эту информацию новым пирам. Однако трекер не участвует в передаче данных.
- **Распределённая хэш-таблица (DHT)**:
  - DHT — это децентрализованная система, которая позволяет пирам находить друг друга без участия трекера. Каждый пир хранит часть информации о других пирах, что позволяет системе работать даже при отсутствии централизованного трекера.

3. Реализация

Основные компоненты системы

- **Трекер (Tracker)**:
  - Трекер управляет информацией о файлах и узлах, участвующих в раздаче. Он обрабатывает запросы от узлов и обновляет свою базу данных.
  - Трекер поддерживает список всех узлов и файлов, которыми они владеют, а также информацию о частоте передачи файлов.
- **Узлы (Nodes)**:
  - Узлы — это пиры, которые могут быть как загрузчиками, так и раздающими. Каждый узел имеет уникальный идентификатор и может владеть набором файлов.
  - Узлы взаимодействуют с трекером для регистрации, поиска файлов и обновления информации о своих файлах.
- **Сообщения (Messages)**:
  - Взаимодействие между узлами и трекером осуществляется через обмен сообщениями. Сообщения кодируются с использованием библиотеки `pickle` и передаются в виде UDP-сегментов.

Реализация трекера

Трекер реализован в файле `tracker.py`. Основные функции трекера включают:

- **Регистрация узлов**:
  - Узел отправляет сообщение с типом `REGISTER`, чтобы зарегистрироваться в системе. Трекер обновляет свою базу данных, добавляя информацию о новом узле.
- **Добавление файлов**:
  - Узел, который хочет передать файл, отправляет сообщение с типом `OWN`, указывая имя файла. Трекер добавляет информацию о том, что данный узел владеет указанным файлом.
- **Поиск файлов**:

  - Узел, который хочет загрузить файл, отправляет сообщение с типом `NEED`, указывая имя файла. Трекер возвращает список узлов, которые владеют этим файлом.

- **Обновление информации**:

  - После успешной передачи файла узел отправляет сообщение с типом `UPDATE`, чтобы обновить информацию о частоте передачи файлов.

- **Удаление узлов**:

  - Узел, который хочет покинуть систему, отправляет сообщение с типом `EXIT`. Трекер удаляет информацию о данном узле из своей базы данных.

Реализация узлов

Узлы реализованы в файле `node.py`. Основные функции узлов включают:

- **Регистрация в системе**:

  - Узел отправляет сообщение трекеру с типом `REGISTER`, чтобы зарегистрироваться в системе.

- **Передача файлов**:

  - Узел, который хочет передать файл, отправляет сообщение трекеру с типом `OWN`, указывая имя файла. После этого узел переходит в режим ожидания запросов от других узлов.

- **Загрузка файлов**:

  - Узел, который хочет загрузить файл, отправляет сообщение трекеру с типом `NEED`, указывая имя файла. Трекер возвращает список узлов, которые владеют этим файлом.
  - Узел запрашивает файл у выбранных узлов, разделяя его на части и загружая их параллельно.

- **Обработка запросов**:

  - Узел обрабатывает запросы от других узлов, отправляя запрошенные части файла с использованием сообщения типа `ChunkSharing`.

- **Периодическое информирование трекера**:

  - Узел периодически отправляет сообщение трекеру с типом `REGISTER`, чтобы сообщить, что он всё ещё находится в системе.

Реализация сообщений

Сообщения реализованы в файле `message.py`. Основные типы сообщений включают:

- **Node2Tracker**:
  - Сообщение, которое отправляется от узла к трекеру. Оно содержит идентификатор узла, режим запроса (например, `REGISTER`, `OWN`, `NEED`) и имя файла.
- **Tracker2Node**:
  - Сообщение, которое отправляется от трекера к узлу. Оно содержит результат поиска файла, включая список узлов, которые владеют этим файлом.
- **Node2Node**:
  - Сообщение, которое отправляется между узлами. Оно используется для запроса размера файла или передачи данных.
- **ChunkSharing**:
  - Сообщение, которое используется для передачи частей файла между узлами. Оно содержит идентификаторы отправителя и получателя, имя файла, диапазон данных и сами данные.

Реализация UDP-сегментов

UDP-сегменты реализованы в файле `segment.py`. Класс `UDPSegment` используется для создания и передачи UDP-сегментов, которые содержат данные сообщений. Каждый сегмент имеет следующие поля:

- **src_port**: Порт отправителя.
- **dest_port**: Порт получателя.
- **length**: Длина данных.
- **data**: Данные сообщения.

Реализация вспомогательных функций

Вспомогательные функции реализованы в файле `utils.py`. Они включают:

- **set_socket**: Создание нового UDP-сокета.
- **free_socket**: Освобождение сокета.

- **generate_random_port**: Генерация случайного порта.
- **parse_command**: Парсинг команд, введённых пользователем.
- **log**: Логирование событий в файл.

4. Результаты

Предположим, у нас есть четыре клиента (узла) и один сервер (трекер). Узлы имеют следующую структуру файлов:

- **node1**:
  - `file_A.txt`
  - `file_B.txt`
  - `file_C.txt`
  - `bittorrent.jpg`
- **node2**:
  - `file_A.txt`
  - `file_B.txt`
  - `file_C.txt`
- **node3**:
  - `file_C.txt`
- **node4**:
  - Нет файлов (новый узел, который хочет загрузить файлы).

Шаг 1: Запуск трекера и узлов

1. **Запуск трекера**:

   ```
   $ python3 tracker.py
   ```

   Вывод:

   ```
   [12:00:00]  Tracker program started just right now!
   ```

2. **Запуск узлов**:

- Узел 1:

```
$ python3 node.py -node_id 1
```

Вывод:

```
[12:00:05]  Node program started just right now!
[12:00:05] You entered Torrent.
```

- Узел 2:

```
$ python3 node.py -node_id 2
```

Вывод:

```
[12:00:10]  Node program started just right now!
[12:00:10] You entered Torrent.
```

- Узел 3:

```
$ python3 node.py -node_id 3
```

Вывод:

```
[12:00:15]  Node program started just right now!
[12:00:15] You entered Torrent.
```

- Узел 4:

```
$ python3 node.py -node_id 4
```

Вывод:

```
[12:00:20]  Node program started just right now!
[12:00:20] You entered Torrent.
```

Шаг 2: Регистрация файлов на трекере

1. **Узел 1 регистрирует файлы**:

```
torrent -setMode send file_A.txt
torrent -setMode send file_B.txt
torrent -setMode send file_C.txt
torrent -setMode send bittorrent.jpg
```

Вывод:

```
[12:00:25] You are free now! You are waiting
for other nodes' requests!
```

2. **Узел 2 регистрирует файлы**:

```
torrent -setMode send file_A.txt
torrent -setMode send file_B.txt
torrent -setMode send file_C.txt
```

Вывод:

```
[12:00:30] You are free now! You are waiting
for other nodes' requests!
```

3. **Узел 3 регистрирует файл**:

```
torrent -setMode send file_C.txt
```

Вывод:

```
[12:00:35] You are free now! You are waiting
for other nodes' requests!
```

Шаг 3: Узел 4 запрашивает файл

1. **Узел 4 запрашивает** `file_C.txt`:

```
torrent -setMode download file_C.txt
```

Вывод:

```
[12:00:40] You just started to download file_C.txt.
Let's search it in torrent!
[12:00:40] You are going to download file_C.txt
from Node(s) [1, 2, 3]
[12:00:40] The file file_C.txt which you are about
to download, has size of 1024 bytes
[12:00:45] All the chunks of file_C.txt has
downloaded from neighboring peers. But they must
be reassembled!
[12:00:45] All the pieces of the file_C.txt is
now sorted and ready to be reassembled.
[12:00:45] file_C.txt has successfully downloaded
and saved in my files directory.
```

Шаг 4: Узел 4 запрашивает `file_A.txt`

1. **Узел 4 запрашивает** `file_A.txt`:

   ```
   torrent -setMode download file_A.txt
   ```

   Вывод:

   ```
   [12:00:50] You just started to download file_A.txt.
   Let's search it in torrent!
   [12:00:50] You are going to download file_A.txt from
   Node(s) [1, 2]
   [12:00:50] The file file_A.txt which you are about to
   download, has size of 2048 bytes
   [12:00:55] All the chunks of file_A.txt has downloaded
   from neighboring peers. But they must be reassembled!
   [12:00:55] All the pieces of the file_A.txt is now
   sorted and ready to be reassembled.
   [12:00:55] file_A.txt has successfully downloaded and
   saved in my files directory.
   ```

Шаг 5: Узел 4 запрашивает `bittorrent.jpg`

1. **Узел 4 запрашивает** `bittorrent.jpg`:

   ```
   torrent -setMode download bittorrent.jpg
   ```

Вывод:

```
[12:01:00] You just started to download bittorrent.jpg.
Let's search it in torrent!
[12:01:00] You are going to download bittorrent.jpg
from Node(s) [1]
[12:01:00] The file bittorrent.jpg which you are about
to download, has size of 4096 bytes
[12:01:05] All the chunks of bittorrent.jpg has
downloaded from neighboring peers. But they must be reassembled!
[12:01:05] All the pieces of the bittorrent.jpg is
now sorted and ready to be reassembled.
[12:01:05] bittorrent.jpg has successfully downloaded
and saved in my files directory.
```

Шаг 6: Узел 4 завершает работу

1. **Узел 4 выходит из системы**:

```
torrent -setMode exit
```

Вывод:

```
[12:01:10] You exited the torrent!
```

Шаг 7: Логи трекера

Логи трекера будут содержать информацию о всех действиях узлов:

```
[12:00:05] Node 1 entered Torrent.
[12:00:10] Node 2 entered Torrent.
[12:00:15] Node 3 entered Torrent.
[12:00:20] Node 4 entered Torrent.
[12:00:25] Node 1 owns file_A.txt and is ready to send.
[12:00:25] Node 1 owns file_B.txt and is ready to send.
[12:00:25] Node 1 owns file_C.txt and is ready to send.
[12:00:25] Node 1 owns bittorrent.jpg and is ready to send.
[12:00:30] Node 2 owns file_A.txt and is ready to send.
[12:00:30] Node 2 owns file_B.txt and is ready to send.
[12:00:30] Node 2 owns file_C.txt and is ready to send.
```

```
[12:00:35] Node 3 owns file_C.txt and is ready to send.
[12:00:40] Node 4 is searching for file_C.txt.
[12:00:45] Node 4 has successfully downloaded file_C.txt.
[12:00:50] Node 4 is searching for file_A.txt.
[12:00:55] Node 4 has successfully downloaded file_A.txt.
[12:01:00] Node 4 is searching for bittorrent.jpg.
[12:01:05] Node 4 has successfully downloaded bittorrent.jpg.
[12:01:10] Node 4 exited torrent intentionally.
```

Итоговая структура файлов у узлов

После выполнения всех команд структура файлов у узлов будет следующей:

- **node1**:
  - `file_A.txt`
  - `file_B.txt`
  - `file_C.txt`
  - `bittorrent.jpg`
- **node2**:
  - `file_A.txt`
  - `file_B.txt`
  - `file_C.txt`
- **node3**:
  - `file_C.txt`
- **node4**:
  - `file_A.txt`
  - `file_C.txt`
  - `bittorrent.jpg`

Таким образом, узел 4 успешно загрузил все запрошенные файлы с других узлов.

5. Обсуждение

В результате работы реализован протокол BitTorrent, обеспечивающий распределённый обмен файлами между узлами с использованием централизованного трекера. Протокол поддерживает регистрацию узлов, передачу и загрузку

файлов, а также обработку некорректного поведения некоторых узлов. Взаимодействие между узлами и трекером осуществляется через UDP-сокеты, что позволяет эффективно передавать данные. Для параллельной обработки запросов использована многопоточность. Протокол был протестирован на системе с четырьмя узлами, включая загрузчика без файлов, и показал возможность корректной загрузки файлов даже при наличии некорректных узлов. Реализация обеспечивает согласованность данных и высокую производительность за счёт распределения нагрузки между узлами.

6. Приложения

Репозиторий с кодом программы и кодом отчёта:
https://github.com/liyayunusova/comp_network