ConcreteTorrent - Projekt Wstępny

Jakub Bryl, Bartosz Ciućkowski, Michał Mazurek, Piotr Zmyślony

6 kwietnia 2020

Spis treści

1 Treść zadania

Użytkownik dołącza do sieci *peer-to-peer* poprzez skontaktowanie się z serwerem, po czym przekazuje serwerowi listę plików jakie jest w stanie udostępnić (może później tę listę edytować). Gdy użytkownik chce pobrać plik, wysyła zapytanie o ten plik do serwera, który zwraca mu listę użytkowników posiadających całość lub część tego pliku. Użytkownik pobierający plik może udostępniać innym użytkownikom tą część pliku, którą udało mu się już pobrać. Podczas pobierania pliku użytkownik będzie cyklicznie odpytywał serwer o to czy w sieci pojawił się ktoś nowy, kto udostępnia pobierany plik. Serwer powinien móc pracować w przestrzeni adresów IPv4 i IPv6.

Krótki słownik pojęć stosowanych w dalszej części dokumentu:

- **Tracker** - serwer, z którym łączą się klienci.
- **Klient** - aplikacja po stronie użytkownika. Odpowiada za przesyłanie/odbieranie plików oraz za kontaktowanie się z trackerem i z innymi klientami.
- **Seed** - klient posiadający kompletny plik.
- **Peer** - klient posiadający fragmenty plików, które udostępnia i równocześnie pobiera brakujące fragmenty.
- **Segment** - fragment pliku.

2 Założenia funkcjonalne i niefunkcjonalne

2.1 Wymagania dotyczące trackera

Wymagania funkcjonalne:

1. System powinien oferować usługę serwera (tracker), który koordynuje wymianę plików pomiędzy użytkownikami umożliwiając znajdowanie siebie nawzajem.
2. Sam serwer nie powinien posiadać kopii plików czy ich części.
3. Użytkownik może uruchomić tracker, będący pośrednikiem w wymianie plików pomiędzy innymi użytkownikami.

Wymagania niefunkcjonalne:

1. Tracker powinien zwracać listę osób, od których użytkownik może pobrać plik w rozsądnym czasie.

2.2 Wymagania dotyczące klienta

Wymagania funkcjonalne:

1. Klient może zawiadomić trackera, o tym, że udostępnia konkretny plik w całości lub w części.
2. Klient może zażądać od trackera listy innych użytkowników udostępniających konkretne pliki.
3. Klient może poprosić innego klienta o wysłanie wybranych fragmentów pliku.
4. Klient może zapytać innego użytkownika o listę fragmentów, jakie posiada.
5. Klient powinien móc równocześnie pobierać i udostępniać zasoby.

Wymagania niefunkcjonalne:

1. Klient nie powinien mieć problemu z jednoczesnym pobieraniem od 5 osób.
2. Klient powinien móc zamknąć program bez ryzyka utraty pobranych fragmentów.
3. W przypadku awarii fragmenty plików już pobranych powinny być niezagrożone

3 Podstawowe przypadki użycia

3.1 UC1. Użytkownik łączy się z siecią P2P:

1. Użytkownik kontaktuje się z serwerem.
2. Przekazuje mu listę plików, które posiada i jest gotowy udostępniać
3. W przypadku, gdy inny użytkownik zażąda któregoś z plików, zacznie mu go od razu udostępniać.
4. W każdym momencie użytkownik ma również możliwość rozpoczęcia pobierania od innych użytkowników.

Alternatywna ścieżka *A*:

3a. Użytkownik nie posiada żadnego pliku do udostępniania (pusta lista).

4a. Serwer nie blokuje użytkownikowi przejścia do kolejnego kroku.

3.2 UC2. Użytkownik chce pobrać określony plik:

1. Użytkownik łączy się z siecią P2P. (UC1.)
2. Określa jaki plik go interesuje.
3. Tracker udostępnia listę dostępnych użytkowników posiadających całość lub część pliku.
4. Użytkownik łączy się z nimi i rozpoczyna pobieranie, jednocześnie udostępniając już posiadane fragmenty (peer).
5. Po uzyskaniu całości pliku użytkownik nadal udostępnia go innym (seed).

Alternatywna ścieżka *A*:

3a. Serwer nie posiada informacji o żadnym użytkowniku udostępniającym żądany plik.

4a. Użytkownik informowany jest o problemie.

5a. Użytkownik ma możliwość kolejnego żądania.

Alternatywna ścieżka *B*:

6b. Użytkownik kończy połączenie z innymi użytkownikami.

7b. Plik znajduje się na urządzeniu użytkownika, ale nie jest dalej przekazywany.

3.3 UC3. Użytkownik chce udostępniać nowy plik (dodać do trackera):

1. Użytkownik przygotowuje specjalny plik zawierający dane o pliku, który zamierza udostępnić innym.
2. Przesyła plik do wybranego trackera.

3. Tracker odsyła komunikat o powodzeniu dodania pliku wraz z wygenerowanym unikalnym identyfikatorem.

Alternatywna ścieżka *A*:

3a. Tracker odsyła komunikat o niepowodzeniu wraz z przyczyną.

4 Wybrane środowisko sprzętowo-programowe

System operacyjny: Ubuntu 16.04 i nowsze

Język programowania: C++17

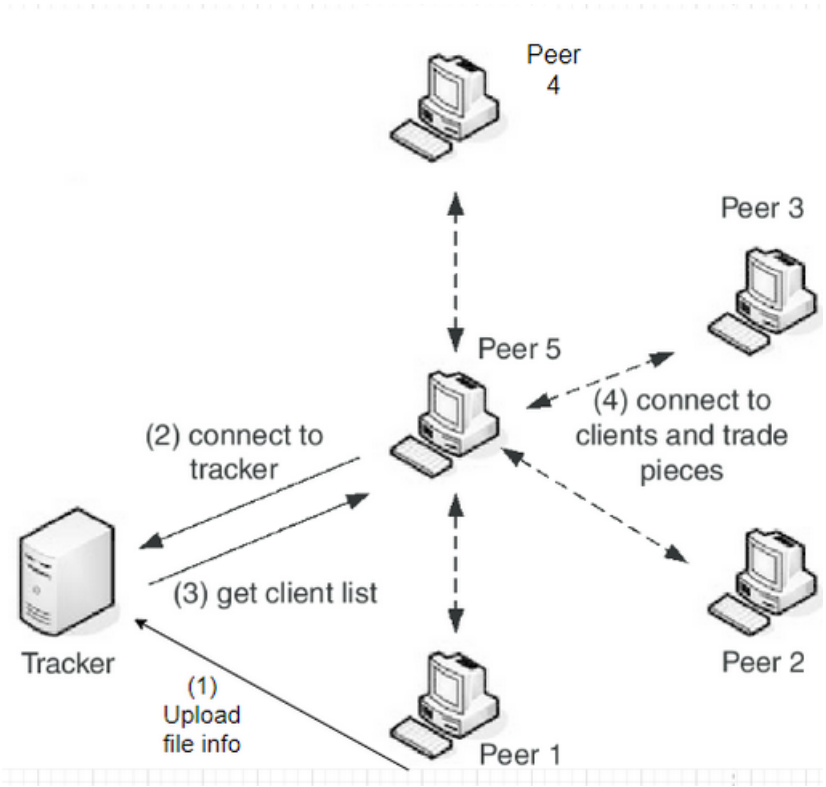
System budowania aplikacji: CMake

Biblioteki: Google Protocol Buffers, spdlog (`github.com/gabime/spdlog`)

Testowanie: Catch lub Boost.Test (jeszcze do ustalenia), Docker

5 Architektura rozwiązania

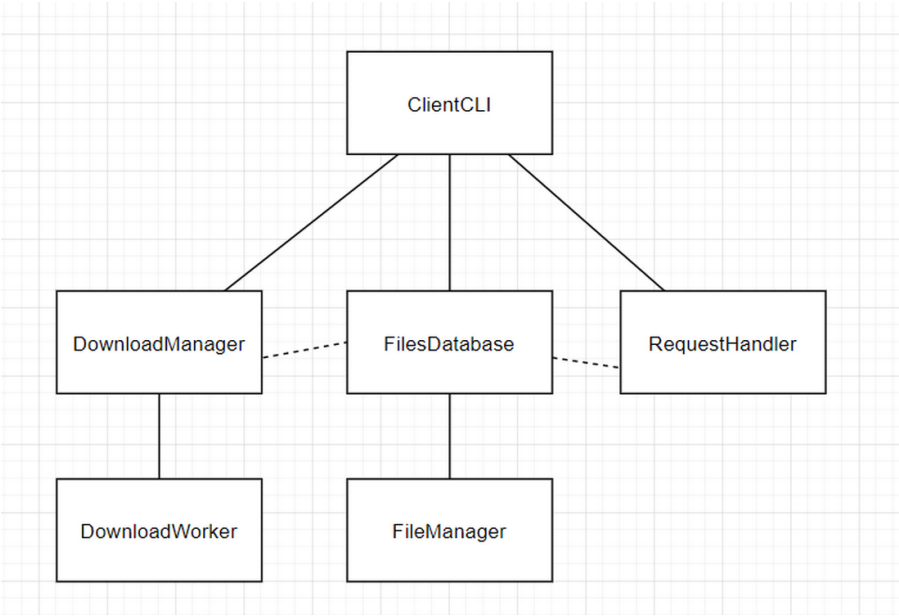
Rysunek 1: Schemat komunikacji sieciowej



Źródło: https://www.researchgate.net/figure/BitTorrent-protocol-A-peer-enters-the-swarm-via-the-tracker-and-starts-exchanging-file_fig1_225633414

5.1 Budowa klienta

Rysunek 2: Schemat modułów klienta



ClientCLI – moduł odpowiedzialny za przyjmowanie żądań od użytkownika i przekazywanie ich innym modułom oraz za wyświetlanie informacji użytkownikowi.

DownloadManager – odpowiada za pobieranie jednego pliku. Tworzy kilka wątków DownloadWorkera, odpowiedzialnych za pobieranie fragmentów danego pliku. DownloadManager kompletuje fragmenty i przesyła je do FilesDatabase.

DownloadWorker – ma za zadanie znalezienie klienta posiadającego jeden z brakujących fragmentów pliku i nawiązanie z nim połączenia. Po nawiązaniu połączenia, pobiera od niego brakujący fragment i wysyła zapytanie o listę fragmentów, które klient posiada. Jeżeli posiada fragmenty których brakuje, prosi o wysłanie kolejnych fragmentów.

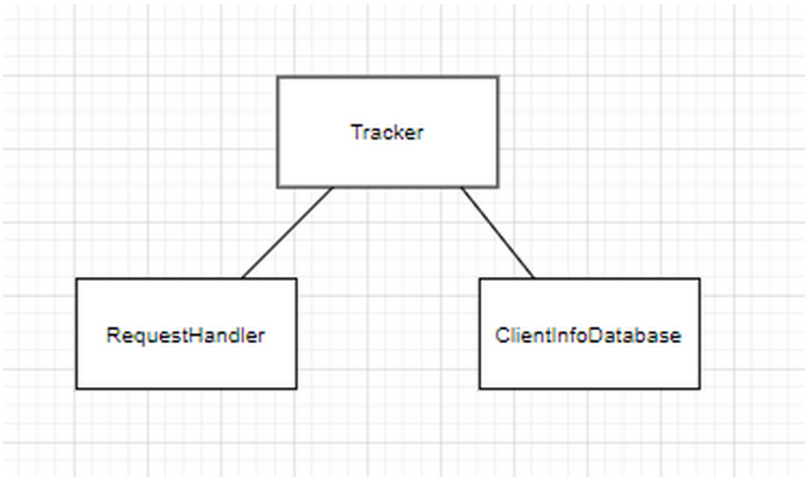
FilesDatabase – przechowuje informacje o plikach, które interesują konkretnego klienta (ma taki plik lub jest w trakcie pobierania). Z każdym pobieranym plikiem, przechowywana jest również informacja o klientach posiadających fragmenty danego pliku.

FileManager – API służące do odczytywania zasobów z systemu plików i buforowania ich oraz do zapisywania na dysku zbuforowanych danych.

RequestHandler – moduł odbierający i odpowiadający na żądania pobierania fragmentu pliku oraz żądanie o informacje na temat posiadanych fragmentów pliku.

5.2 Budowa trackera

Rysunek 3: Schemat modułów trackera



ClientInfoDatabase - przechowuje informacje o klientach w sieci oraz o posiadanych przez nich plikach.

RequestHandler - odbiera i odpowiada na żądania o listę klientów posiadających dany plik oraz na żądanie udostępnienia pliku w sieci przez klienta.

6 Lista komunikatów sieciowych

Komunikacja w naszym systemie będzie zachodziła ścieżkami klient–tracker i klient–klient. Dodatkowo, każdy klient może równolegle komunikować się z wieloma innymi klientami korzystającymi z tego samego trackera.

6.1 Komunikaty klient–tracker

- **CS_SEEDLIST_REQUEST** – klient wysyła zapytanie do serwera, o to czy jakiś inny klient posiada dany plik w całości lub w części. Komunikat ten sygnalizuje również trackerowi, żeby dodać klienta do listy adresów udostępniających danych plik.
- **CS_SEEDLIST_RESPONSE** – jeżeli jacyś klienci faktycznie udostępniają dany plik, tracker odsyła pytającemu adresy tych klientów, wraz z oznaczeniem czy są peerami czy seedami.
- **CS_CLIENT_UNAVAILABLE** – komunikat sygnalizujący sytuację, w której klient A dowiedział się od klienta B, że nie udostępnia już danego pliku. W tym wypadku tracker, aby usunąć klienta B z posiadaczy pliku, potrzebuje odebrać więcej podobnych raportów od innych klientów.
- **CS_IM_A_SEED** – klient ogłasza trackerowi, że jest posiadaczem całości pliku.
- **CS_NEW_REQUEST** – klient prosi tracker o dodanie posiadanego pliku do listy udostępnianych.
- **CS_NEW_RESPONSE** – tracker odpowiada klientowi o sukcesie lub niepowodzeniu dodania pliku do listy udostępnianych.

6.2 Komunikaty klient–klient

- **CC_LIST_REQUEST** – jeden klient A pyta drugiego (klienta B), jakie fragmenty konkretnego pliku/plików posiada.
- **CC_LIST_RESPONSE** – odpowiedź do powyższego zapytania. W przypadku udostępniania danego pliku - wysyła listę posiadanych fragmentów (lub tych których nie ma - zależnie od tego których jest mniej). Klient B może odpowiedzieć, że takiego pliku w ogóle nie udostępnia - w tym wypadku klient A wysyła *CS_CLIENT_UNAVAILABLE* do trackera, z którego otrzymał informacje o kliencie B.
- **CC_FRAGMENT_REQUEST** – komunikat proszący klienta o przesłanie danego fragmentu pliku.
- **CC_FRAGMENT_RESPONSE** – zawiera blok danych (fragment), o który prosił go inny klient. Może również odpowiedzieć, że nie zgadza się na wysłanie tego fragmentu (lub, że już go nie posiada).

7 Sposób testowania

Zastosujemy testy jednostkowe, sprawdzające poprawność działania pojedynczych części systemu, tworzone przy użyciu odpowiednich bibliotek wymienionych powyżej.

Do tego planujemy utworzyć skrypty testowe symulujące realne użycie z pomocą narzędzia *Docker*. Pozwoli ono przetestować działanie naszego systemu na indywidualnych maszynach poprzez utworzenie sieci kontenerów, symulujących większą ilość klientów i trackerów.

8 Sposób demonstracji rezultatów

Wyróżniliśmy następujące scenariusze testowe:

Scenariusz I Podłączenie się przez użytkownika A do systemu i przekazanie trackerowi informacji o posiadanym pliku. Następnie podłączenie do systemu użytkownika B i zażądanie od trackera informacji o klientach posiadających wysłany przez A plik. Tracker powinien zwrócić klientowi B adres klienta A. Następnie B wysyła do A prośby o kolejne segmenty pliku, A mu je odsyła, aż do skompletowania całego pliku.

Scenariusz II Scenariusz podobny jak wyżej, ale z większą liczbą użytkowników posiadających dany plik. Test ten pokaże pobieranie współbieżne od kilku klientów.

Scenariusz III Dwóch klientów (A i B) udostępniających po jednym pliku, odpowiednio *P1* i *P2*. Test ten zaprezentować ma pobieranie przez A pliku *P2* od B oraz pobieranie przez B pliku *P2* od A.

Scenariusz IV Klient jest w trakcie pobierania pliku. Proces klienta zostaje nagle zamknięty i włączony ponownie. Test zakończy się sukcesem, jeśli klient wznowi pobieranie pliku, bez utraty fragmentów pobranych przed wyłączeniem.

9 Organizacja pracy

9.1 Podział pracy

Podział prac jest bardzo orientacyjny i z łatwością może ulec zmianie, ponieważ nie jesteśmy pewni ile nakładu pracy będą wymagały poszczególne moduły. Wstępny podział wygląda następująco:

- **DownloadManager**, **DownloadWorker**, **FilesDatabase** i **ClientInfoDatabase** – Jakub Bryl, Michał Mazurek
- **RequestHandlers**, **ClientCLI** i **FileManager** – Piotr Zmyślony, Bartosz Ciućkowski

9.2 Harmonogram

Główne kamienie milowe i terminy:

27 kwietnia - Wymiana informacji między klientem a trackerem oraz podstawowa komunikacja między klientami.

12 maja - Współbieżna wymiana plików między wieloma klientami.

9.3 Zdalne repozytoria

Projekt jest rozwijany przy pomocy systemu kontroli wersji git, pod adresem https://github.com/zmysloony/p2p-network/. Dodatkowo korzystamy z Trello do zarządzania zadaniami i wymiany pomysłów: https://trello.com/b/EuSrukdO/tin-torrent.