ConcreteTorrent - Projekt Końcowy

Jakub Bryl(lider), Bartosz Ciućkowski, Piotr Zmyślony

26 maja 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.
- **Torrent** - struktura danych przechowująca informacje o danym 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 klientów, 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 klientów.
2. Klient powinien móc zamknąć program bez ryzyka utraty pobranych fragmentów.
3. W przypadku awarii fragmenty plików już pobranych powinny być niezagrożone

2.3 Dodatkowe założenia

1. Torrent będzie zawierał nazwę pliku, jego rozmiar, datę udostępnienia w systemie oraz skrót obliczony na podstawie wymienionych wczesniej danych.
2. Pliki będą identyfikowane na podstawie skrótu wygenerowanego na serwerze. W razie wystąpienia konfliktu skrótów do identyfikacji będzie też używana nazwa pliku.

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

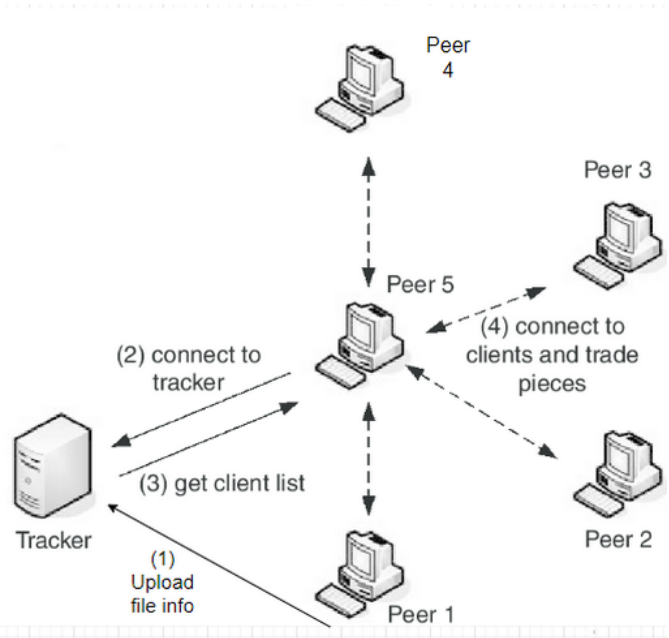
System budowania aplikacji: CMake

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

Testowanie: Boost.Test do testów jednostkowych, Docker

5 Architektura rozwiązania

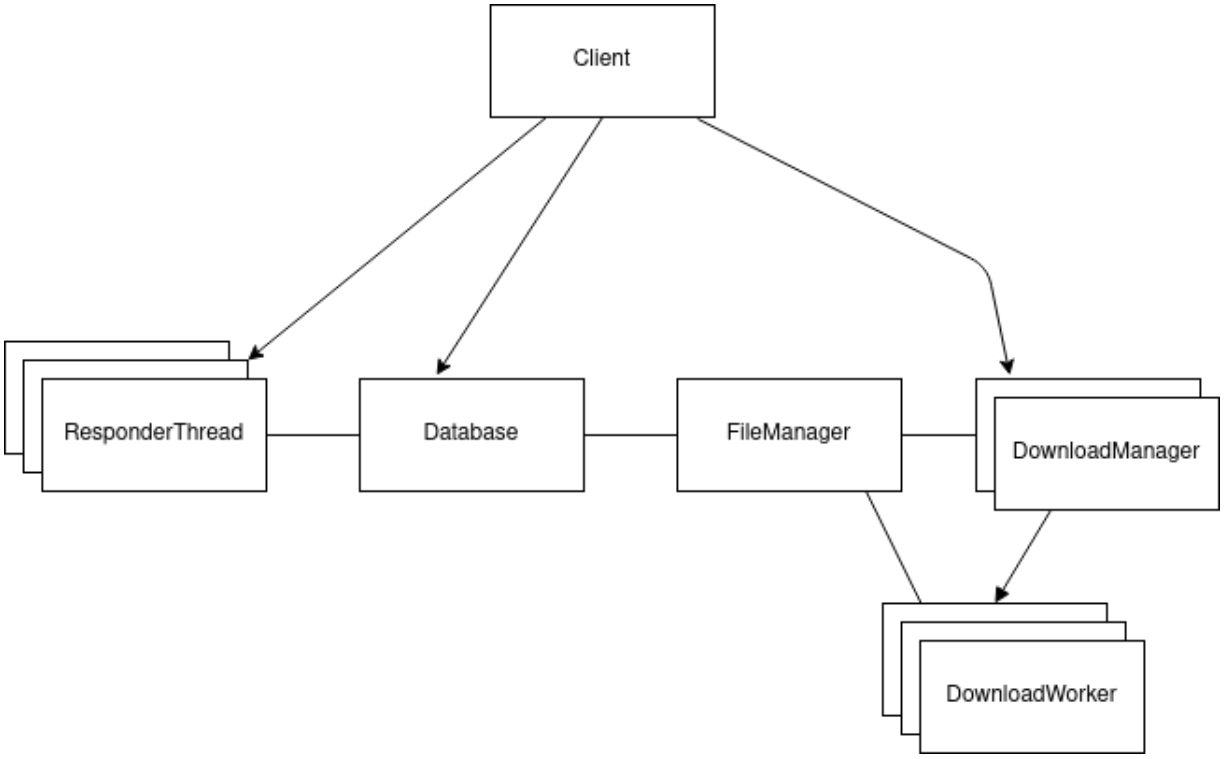
Rysunek 1: Schemat komunikacji sieciowej



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



Client – 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 nawiązanie połączenia z klientem posiadającym fragmenty pobieranego pliku i zażądanie od niego tych fragmentów, których brakuje do skompletowania pliku. Pobrane segmenty zapisuje przy pomocy FileManagera do odpowiednich miejsc na dysku.

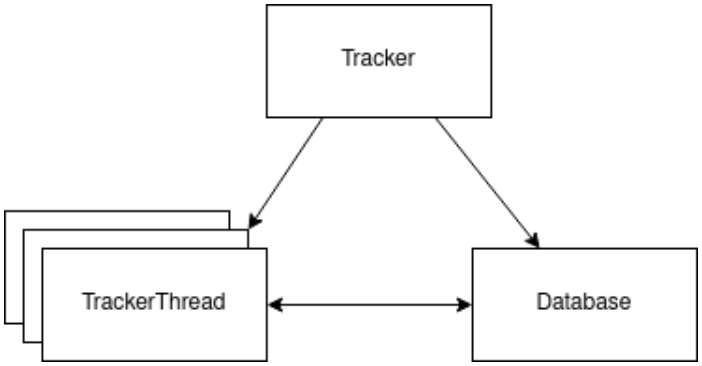
Database – przechowuje informacje o plikach, które interesują konkretnego klienta (ma taki plik lub jest w trakcie pobierania). Z każdym pobieranym plikiem, przechowywana jest również informacja o klientach posiadających fragmenty danego pliku oraz powiązany z tym plikiem obiekt identyfikujący go w trackerze (Torrent).

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

ResponderThread – wątek odpowiedzialny za komunikację z innym klientem, 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



Tracker – inicjuje serwer i nasłuchuje na żądania od klientów. Tworzy oddzielny wątek TrackerThread dla każdego klienta.

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

TrackerThread - odbiera i odpowiada na żądania o listę klientów posiadających dany plik oraz na żądanie udostępnienia pliku w sieci przez klienta. Po określonym czasie bez odpowiedzi podłączonego klienta ulega timeoutowi.

6 Lista komunikatów sieciowych

Komunikacja w naszym systemie będzie zachodziła ścieżkami klient–tracker i klient–klient. Komunikaty są serializowane ze pomocą biblioteki Google Protocol Buffers. 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.

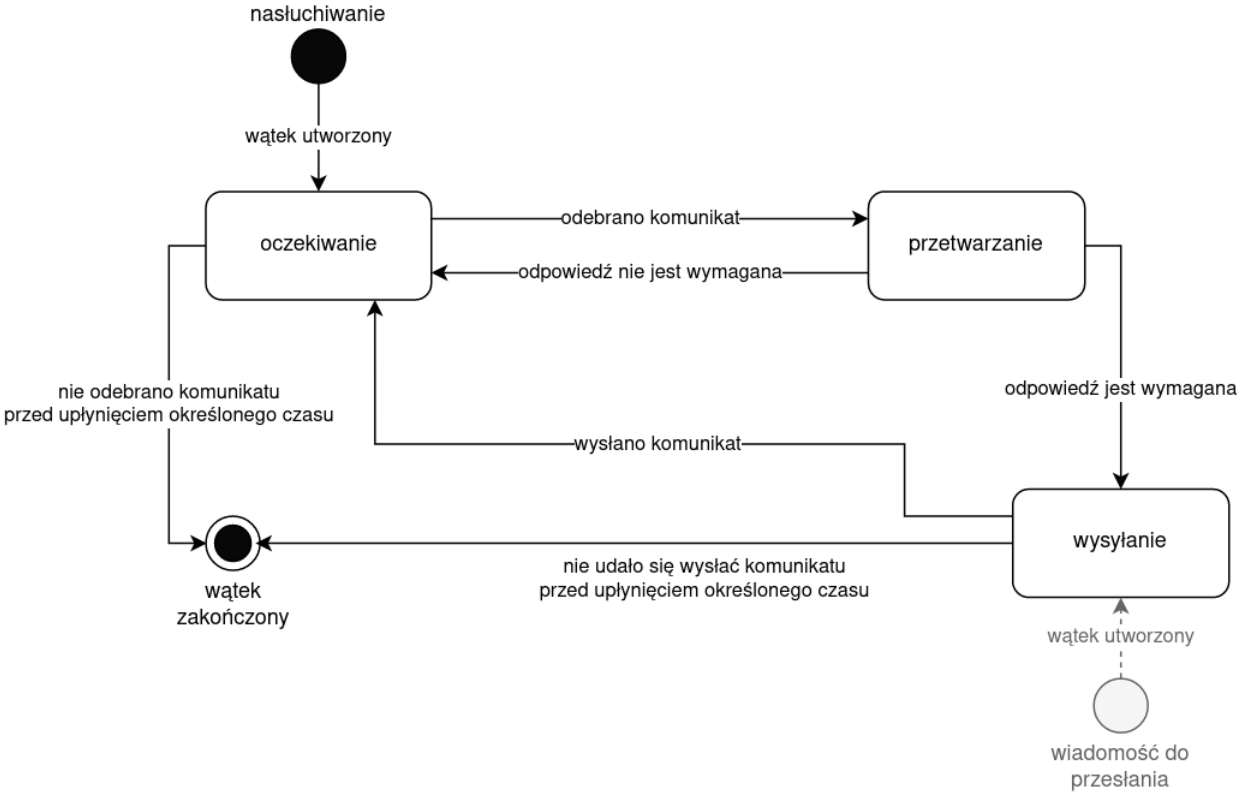
6.2 Komunikaty klient–klient

- **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 Zachowanie podmiotów komunikacji

Generalizując, komunikacja na ścieżce tracker-klient i klient-klient zachodzi w identyczny sposób - różnice są jedynie w obsługiwanych komunikatach. Przesyłanie wiadomości zachodzi w sposób pokazany na poniższym rysunku:

Rysunek 4: Pseudo-diagram maszyny stanowej



Z powodu oszczędności miejsca i niechęci do dodawania prawie identycznego diagramu, zdecydowaliśmy się na lekko niepoprawną wersję diagramu maszyny stanowej - wyszarzony stan początkowy symbolizuje właściwy stan początkowy dla strony inicjującej połączenie z intencją przesłania komunikatu. Natomiast drugi, koloru czarnego, symbolizuje stan początkowy dla strony, która zaczyna od odbierania wiadomości.

7.1 Timeouty

Timeouty występują w przypadku połączeń klient-tracker oraz klient-klient. Wyznaczają one jak dużo czasu proces może spędzić na oczekiwaniu na odpowiedź od drugiego klienta/trackera lub potwierdzenie odebrania wiadomości.

Wszystkie timeouty ustawione są domyślnie na 5 sekund, ale możliwa jest ich zmiana po podaniu odpowiedniej opcji przy uruchomieniu programu klienckiego lub trackera.

7.2 Obsługa sytuacji błędnych

Sytuacje błędne są komunikowane poprzez wpisy do sysloga systemu linux, a te bardziej poważne - wyrzucane na wyjście stderr działającego procesu (klienta bądź trackera). Program kontynuuje dalsze działanie w przypadku błędów związanych z systemem plików, pomijając pliki, przy których obsłudze wystąpił błąd.

Najczęściej występującą sytuacją błędną jest brak możliwości utworzenia gniazda na konkretnym porcie (co może wynikać z tego że jest/są one zajęte) - w tym wypadku program zakończy swoje działanie z odpowiednim komunikatem. Użytkownik może spróbować uruchomić program, z podaniem odpowiedniej opcji, tak aby używał portów innych niż domyślne;

Większość błędów jest zupełnie nieszkodliwa i wynika z np. usunięciu przez użytkownika konkretnego pliku .torrent, z którym powiązany był pobierany plik. Ogólnym sposobem na uniknięcie występowania błędów jest powstrzymanie się od modyfikacji plików, z których korzysta program.

8 Podatności bezpieczeństwa

Bezpieczeństwo polega tutaj na zaufaniu klienta A między innymi klientami, ponieważ na drodze przesyłania fragmentów plików nie zachodzi sprawdzenie, czy otrzymany fragment jest faktycznie tym, który A chciał dostać.

Dodatkowo, serwer nie jest odporny na duża ilość połączeń - po pewnym momencie przestanie akceptować nowe. Stąd wystarczy atak SYN flooding, żeby uniemożliwić innym klientom dostęp do serwera-trackera.

Komunikaty przesyłane są bez szyfrowania, ale też nie zawierają żadnych krytycznych informacji - zaszyfrowanie pliku leży po stronie osoby, która chce go udostępnić.

9 Sposób testowania

Zastosujemy testy jednostkowe, sprawdzające poprawność działania pojedynczych części systemu, tworzone przy użyciu odpowiednich bibliotek wymienionych powyżej. Testy jednostkowe stosujemy głównie do sprawdzania części, które nie zajmują się komunikacją sieciową lub tworzeniem wielu wątków, ponieważ w tej dziedzinie są one trudniejsze do zastosowania.

Do testowania wykorzystujemy Dockera poprzez utworzenie sieci kontenerów i uruchomienie w jednym kontenerze aplikacji trackera, a w pozostałych, aplikacji klienta. Do każdego z klienckich kontenerów można "wejść" i wykonywać działania potencjalnego użytkownika.

10 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.

11 Organizacja pracy

11.1 Podział pracy

Podział prac uległ znacznej zmianie, z tego powodu, że ubyło nam jednego członka zespołu. Końcowo podział modułów między osoby wygląda w następujący sposób:

- **DownloadManager**, **DownloadWorker**, **Database** i **FileManager** – Jakub Bryl, Bartosz Ciućkowski
- **ResponderThread** i **TrackerThread** – Piotr Zmyślony

11.2 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.

11.3 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.

12 Instrukcja instalacji

Proces instalacji należy rozpocząć od sklonowania repozytorium:

```
> git clone https://github.com/zmysloony/p2p-network/
```

12.1 Instrukcja podstawowa

Następnie w folderze głównym projektu należy uruchomić skrypt instalujący potrzebne pakiety i biblioteki:

```
> ./run-after-clone.sh
```

Na samym końcu wystarczy zbudować program klienta i tracker:

```
> cmake .
```

```
> make
```

Ścieżki do zbudowanych aplikacji:

- klient – `PROJECT_PATH/src/client/client`
- tracker – `PROJECT_PATH/src/server/server`

12.2 Instrukcja alternatywna

Aplikację kliencką i trackera można uruchomić w kontenerach:

```
> docker build -t client -f Dockerfile.client .
> docker build -t server -f Dockerfile.server .
> docker run -it client
> docker run -it server
```

Aby przywrócić maszynę do stanu sprzed instalacji ConcreteTorrent wystarczy usunąć pakiety zainstalowane przy uruchamianiu skryptu run-after-clone.sh i następnie usunąć folder projektu.