# **Temat 7**

# **BitTorrent: poprawność danych**

Denisa Stępień

Adam Ślusarczyk  
Radek Gębusia

Piotr Michałek

# **Wstęp**

Torrentami nazywamy małe pliki które zawierają informacje za pośrednictwem których użytkownik może pobierać inne konkretne pliki np.: zdjęcia, filmy, muzykę. Te pliki znajdują się na komputerach innych użytkowników oraz są przez nich udostępniane. Wymiana plików pomiędzy użytkownikami zachodzi na zasadzie peer-to-peer, co oznacza, że każda osoba, która pobierze jakiś plik, następnie sama ten plik udostępnia. Takie podejście umożliwia wyłączenie serwerów firm trzecich, wymiana plików zachodzi tylko i wyłącznie pomiędzy konkretnymi użytkownikami. Udostępnianie plików torrent można w każdym momencie zatrzymać i wznowić. Do wymiany plików torrent wykorzystywany jest BitTorrent. Jest to protokół służący do wymiany i dystrybucji plików przez internet, mający na celu odciążenie łączy z serwera udostępniającego pliki. Największą zaletą BitTorrent jest podział pasma pomiędzy osoby pobierające plik w tym samym czasie, co oznacza, możliwość wysyłania fragmentów pliku innym użytkownikom w trakcie pobierania. Dzięki temu prędkość pobierania pliku nie jest ograniczona możliwościami serwera bądź łącza sieciowego, nawet w przypadku pobierania jednego pliku przez wielu użytkowników. W normalnych warunkach nie ma znaczenia jak dużo klientów pobiera dany plik, serwer i tak musi wysłać tylko jedną kopię. Cały proces pobierania pliku rozpoczyna się od pobrania przez klienta sieci pliku o rozszerzeniu .torrent, który zawiera informacje o zawartości, adresie trackera i sumach kontrolnych poszczególnych plików przesyłanych przez sieć. W kolejnym kroku, klient łączy się z trackerem otrzymując od niego informacje o innych użytkownikach. Te informacje co określony okres czasu są aktualizowane. W tym momencie rozpoczyna się pobieranie pliku.

Podstawowe pojęcia związane z projektem:

* Peer – to użytkownik, który w tym samym momencie pobiera i udostępnia dany plik innym użytkownikom
* Seeder – to użytkownik posiadający cały, komplenty plik oraz udostępnia go innym użytkownikom
* Tracker – to serwer, który przekazuje informacje o innych użytkownikach którzy pobierają dany plik
* Plik z rozszerzeniem .torrent – to metaplik, który zawiera informacje takie jak: zawartość archiwum, adres trackera i sumy kontrolne. Powyższe informacje są niezbędne do rozpoczęcia pobierania pliku.

# **Implementacja**

# Tracker

W projekcie tym została zastosowana warstwa, która zbiera dane o wszystkich uczestnikach oraz pomaga ona nawiązywać połączenia uczestnikom z innymi uczestnikami. Warstwa ta została nazwana *Trackerem*, ponieważ po części przypomina jego działanie, lecz według protokołu BitTorrent tracker posiada tylko informację o połączonych z nim uczestnikach (peerach). Poniżej znajduje się lista informacji zbieranych przez tę warstwę:

* Identyfikator uczestnika
* Czy uczestnik jest widoczny
* Czy uczestnik ma przesyłać poprawne dane
* Ip i port uczestnika, na którym będzie oczekiwać połączeń innych uczestników
* Lista zablokowanych uczestników przez danego uczestnika
* Pliki pobrane lub pobierane wraz z informacją o posiadanych częściach plików

Informacje te, poprzez REST API, prezentowane są w warstwie prezentacji w postaci strony WWW.

## Działanie trackera

Działanie trackera sprowadza się do nasłuchiwania przychodzących połączeń na porcie 60000 oraz do sprawdzania dostępności już połączonych uczestników (*Peerów*) co sekundę od zakończenia ostatniego sprawdzania. Po połączeniu się nowego uczestnika tracker oczekuje na przesłanie obiektu *InitConnectToTracker*, po czym tworzy obiekt uczestnika (*Peer*), przesyła mu listę innych połączonych peerów wraz z listą posiadanych przez nich plików oraz informuje pozostałych uczestników o nowym połączonym peerze i oczekuje na kolejne połączenia. Obiekt uczestnika po utworzeniu uruchamia nasłuchiwanie przesyłanych komunikatów od tego uczestnika (*PeerListener*). PeerListener oczekuję na następujące komunikaty:

* *ReceivePieceFile* – po otrzymaniu tego komunikatu peer zapisuje informację o poprawnym odebraniu części pliku o konkretnym identyfikatorze oraz informuje pozostałych uczestników o posiadanej nowej części pliku.
* *CheckAvailablePeer* – po otrzymaniu tego komunikatu peer zapisuje informację o swojej dostępności.
* *BanPeer* – w tym komunikacie jest przekazywany identyfikator zablokowanego uczestnika, który jest dodawany do prywatnej listy uczestnika z zablokowanymi uczestnikami.
* *FileTorrent* – po utrzymaniu tego komunikatu peer zapisuje informację o pobranym lub pobieranym pliku w zależności od ilości odebranych części tego pliku oraz informuje pozostałych uczestników o posiadaniu tego pliku.

Jak już zostało wspomniane tracker prezentuje otrzymane informację poprzez REST API. API to umożliwia również przesłanie polecenia zmiany dostępności uczestnika lub wysyłanych przez niego danych. Polecenia te przekazywane są do trackera, który przekazuje je do peera, po czym peer je realizuje.

# Komunikacja pomiędzy peerami

Wiadomość przesyłana pomiędzy peerami składa się z:

* ID części (4 bajty)
* Długości części (4 bajty)
* Hashu torrentu (4 bajty)
* Typu wiadomości (1 bajt)
* GUIDu nadawcy (4 bajty)
* Zakodowanej części pliku (długość określona w drugim polu)

Pole oznaczające typ wiadomości przyjmuje wartości:

* 0 – Żądanie wysłania danej części
* 1 – Wysłanie danej części
* 2 – Żądanie wysłania GUIDu danego peera
* 3 – Wysłanie GUIDu danego peera

## **Odbieranie danych**

Każdy peer nasłuchuje na zdarzenia związane z odebraniem danych przez strumień. W momencie gdy zdarzenie się pojawia, uruchamiana jest metoda HandleRead(). Metoda jest długa, dlatego zrezygnowano z umieszczania jej w sprawozdaniu.W tej metodzie, dane w postaci tablicy bajtów pobrane ze strumienia, są dekodowane oraz weryfikowane poprzez porównanie przysłanego hashu torrentu z hashem znajdującym się w pliku torrent oraz porównanie hashy odebranych części przy pomocy metody CheckPieceHash(). Jeśli weryfikacja przebiegnie pomyślnie, zdekodowane dane są zapisywane w liście Incoming jako obiekt klasy PendingMessage. W przeciwnym wypadku podejmowane są kroki zapobiegawcze prowadzące do zbanowania peera.

## **Przetwarzanie i wysyłanie danych**

Wiadomości przychodzące są zapisywane w postaci obiektów klasy PendingMessage w liście Incoming przez metodę HandleRead(). Następnie, klient w dedykowanym do tego celu wątku przetwarza te wiadomości zapisując do pliku otrzymane części, wczytując z pliku części które zostaną następnie wysłane innemu peerowi, czy przesyłając lub zapisując GUID peera. Odpowiedzi na żądania peerów są zapisywane w liście Outgoing, a następnie wysyłane przez przeznaczony do tego wątek.

## **Banowanie peerów**

W przypadku gdy peer przesyła niepoprawne wiadomości, możliwe jest zablokowanie komunikacji z tym peerem. Odbywa się to przy pomocy metody IncreaseErrorCounter() klasy Peer. Każdy połączony peer posiada własny licznik, który zwiększany jest gdy peer przesyła błędne dane. Gdy licznik przekroczy wartość określoną w statycznym polu Settings.maxErrorCount peer jest banowany poprzez dodanie jest GUIDu do listy BannedPeers oraz wysłanie odpowiedniej wiadomości do Trackera.

## **Timeout peerów**

W omawianym programie został zaimplementowany mechanizm pozwalający na zrywanie połączenia z peerami, od których przez pewien zadany czas nie otrzymano żadnej wiadomości. Czas ten jest określony w polu Settings.PeerTimeoutInMinutes. W przeznaczonym do tego wątku, klient sprawdza czas, który upłynął od otrzymania ostatniej wiadomości od peera. Jeśli czas ten zostanie przekroczony, następuje zerwanie połączenia.

**Sprawdzanie poprawności danych**

Za sprawdzanie poprawności otrzymanego kawałka pliku są odpowiedzialne trzy metody CheckReceivedPiece, CheckPieceHash, Hash. Pierwsza z nich sprawdza czy nie otrzymano za dużo lub za mało danych. Druga oblicza hash dla kawałka i porównuje go z tym, który otrzymano z pliku torrent. Jeśli hashe są zgodne, to oznacza, że otrzymano prawidłowy kawałek. Ostatnia metoda służy do obliczenia hasha. Dodatkowo przed sprawdzaniem poszczególnych kawałków, weryfikuje się czy hash dla całego pliku jest zgodny.

public static bool CheckReceivedPiece(byte[] bytes, int index, int receivedBytes)

{

if (receivedBytes > PiecesLength)

{

Console.WriteLine("Otrzymano za dużo danych.");

return false;

}

else if (receivedBytes < PiecesLength)

{

byte[] truncArray = new byte[receivedBytes];

Array.Copy(bytes, truncArray, truncArray.Length);

var rs = CheckPieceHash(truncArray, index);

return rs;

}

var r = CheckPieceHash(bytes, index);

return r;

}

public static bool CheckPieceHash(byte[] bytes, int index)

{

var hash = Hash(bytes);

var result = hash.SequenceEqual(PieceHashes[index]);

string h1 = "";

string h2 = "";

foreach (byte b in hash)

h1 += b.ToString();

foreach (byte b in PieceHashes[index])

h2 += b.ToString();

Console.WriteLine("=======\nha:"+ h1 + "\nex:" + h2 + "\n===========");

if (!result)

{

Console.WriteLine("Nie otrzymano prawidłowego fragmentu.");

return false;

}

Console.WriteLine("Otrzymano prawidłowy fragment.");

return true;

}

public static byte[] Hash(byte[] temp)

{

using (SHA1Managed sha1 = new SHA1Managed())

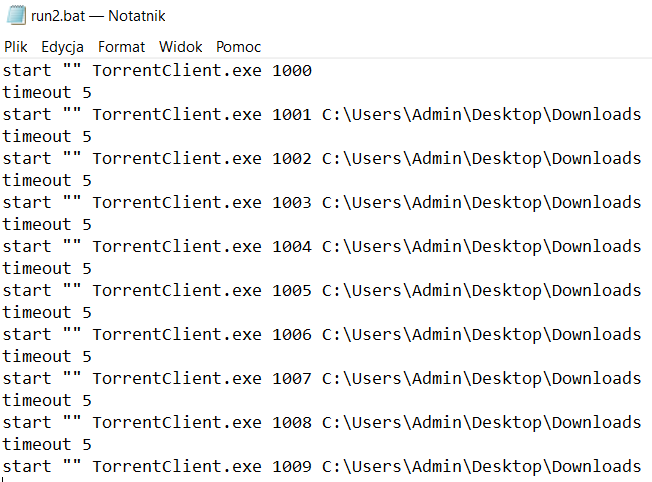
{

return sha1.ComputeHash(temp);

}

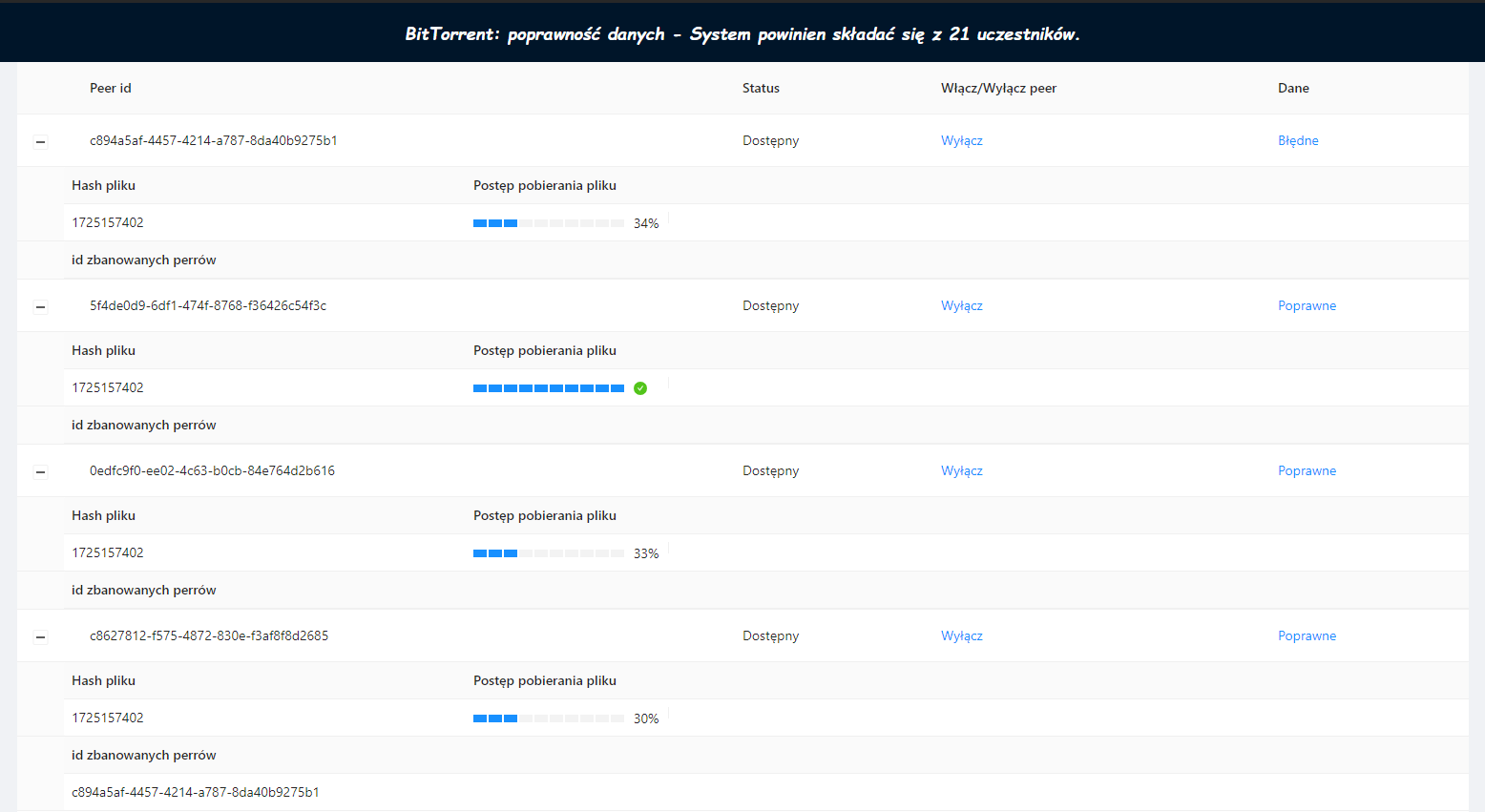
}

Dla aplikacji stworzono również plik .bat, który umożliwia uruchomienie kilka instancji aplikacji równocześnie. Numery od 1000-1009, to numery portów, a następna ścieżka wskazuje, gdzie maja być zapisywane pliki.



# **Opis interfejsu**

Aplikacja prezentuje się jak na poniższym rysunku (Rys. 1)

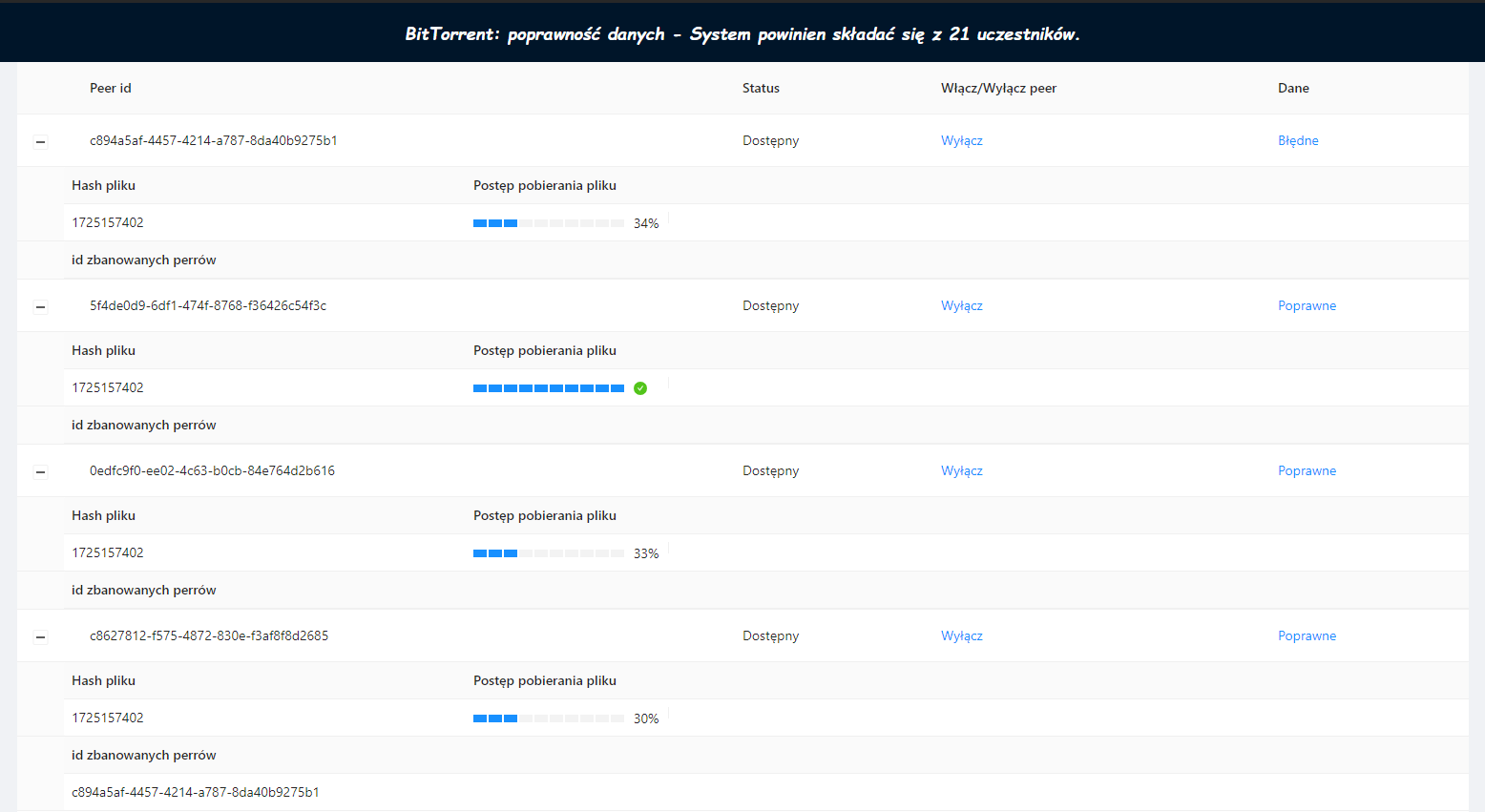


Użytkownikowi wyświetlane są dane takie jak: peer id oraz status peer’a. Status każdego z peer’ów może być dostępny lub niedostępny. Dostępne są również dwa przyciski. Pierwszy z nich umozliwia wyłączenie peer’a, co będzie skutkować zablokowaniem możliwości wysyłania oraz pobierania plików. Przycisk wysyłaj błędne dane, umożliwia użytkownikowi rozpoczęcie wysyłania nieprawidłowych danych przez peer w celu sprawdzenia jak zareagują na to pozostałe peer’y. Po kliknięciu przycisku + na ekranie pojawią się dodatkowe informacje o pobieranych przez dany peer plikach, takie jak nazwa i postęp pobieranego pliku wyrażony w procentach. Zaprezentowane zostaną również id peer’ów, które zostały zbanowane.

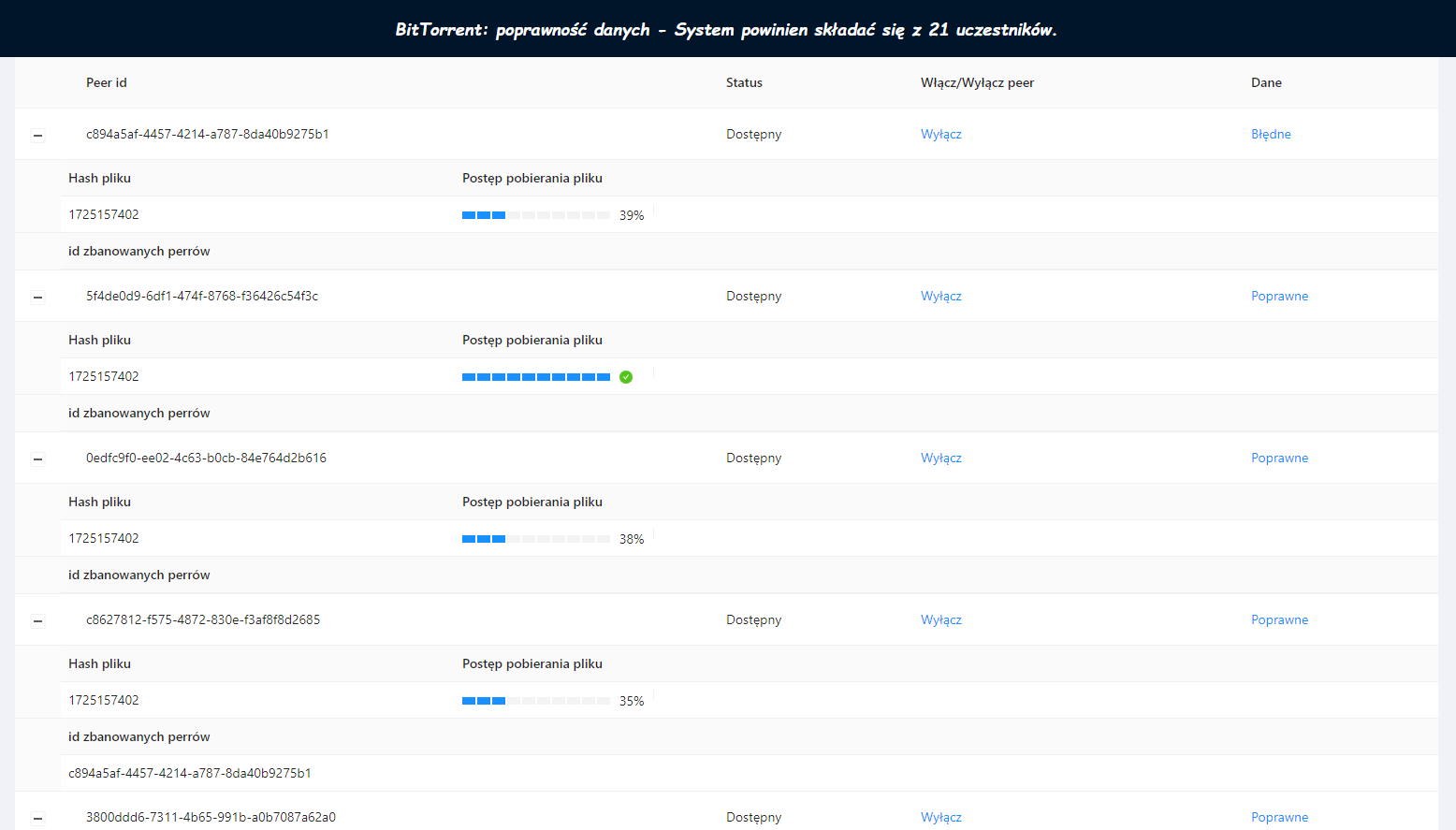
# **Testy**

W trakcie przeprowadzonych testów napotkaliśmy na pewien problem, który wynika z tego, że dla każdej instancji aplikacji uruchamiamy kilka wątków (około 5). Jeden wątek nasłuchuje wiadomości napływających, drugi obsługuje kierowane do niego zapytania, inny sprawdza wiadomości otrzymywane od trakera, kolejny wysyła zapytania o kolejne części pliku. Dodatkowo działający pozostaje wątek główny aplikacji. Uruchamianie powyżej 10 uczestników powodowało zawieszanie się komputera, dlatego musieliśmy testować na ich mniejszej liczbie.

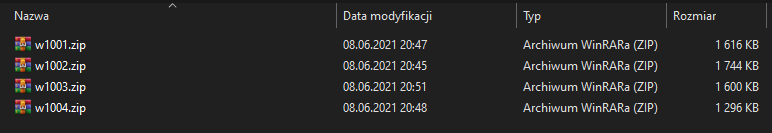
Na pierwszym widoku widać postępy pobierania plików dla poszczególnych peerów. Można dostrzec również, że pierwszy z nich wysyła błędne dane i został dodany do zbanowanych peerów dla ostatniego z uczestników.

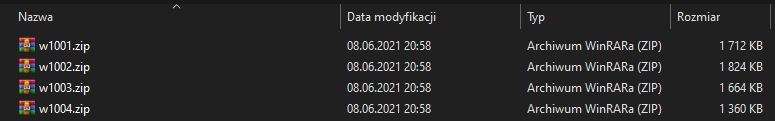


Na kolejnym widoku widać, że postępy pobierania plików dla poszczególnych peerów się zwiększyły po upływie czasu.

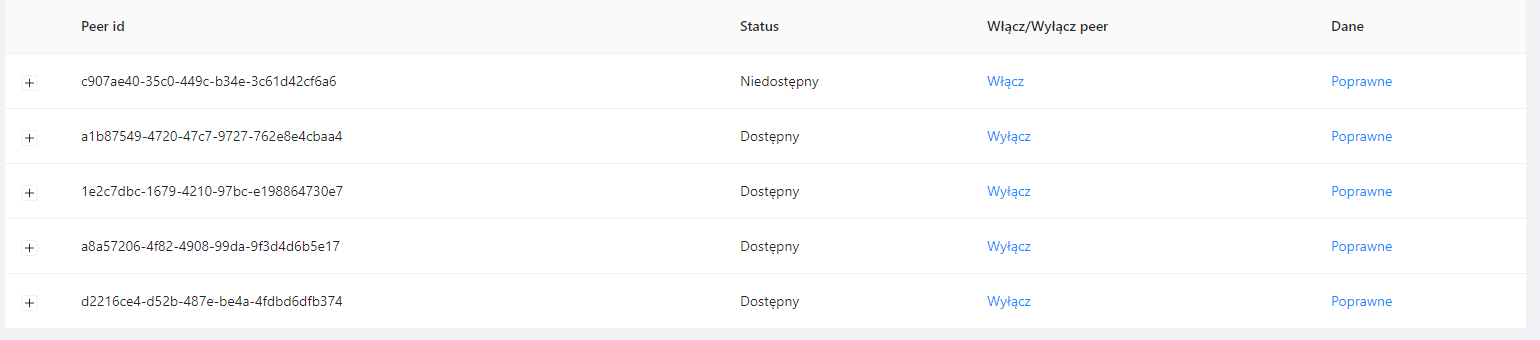


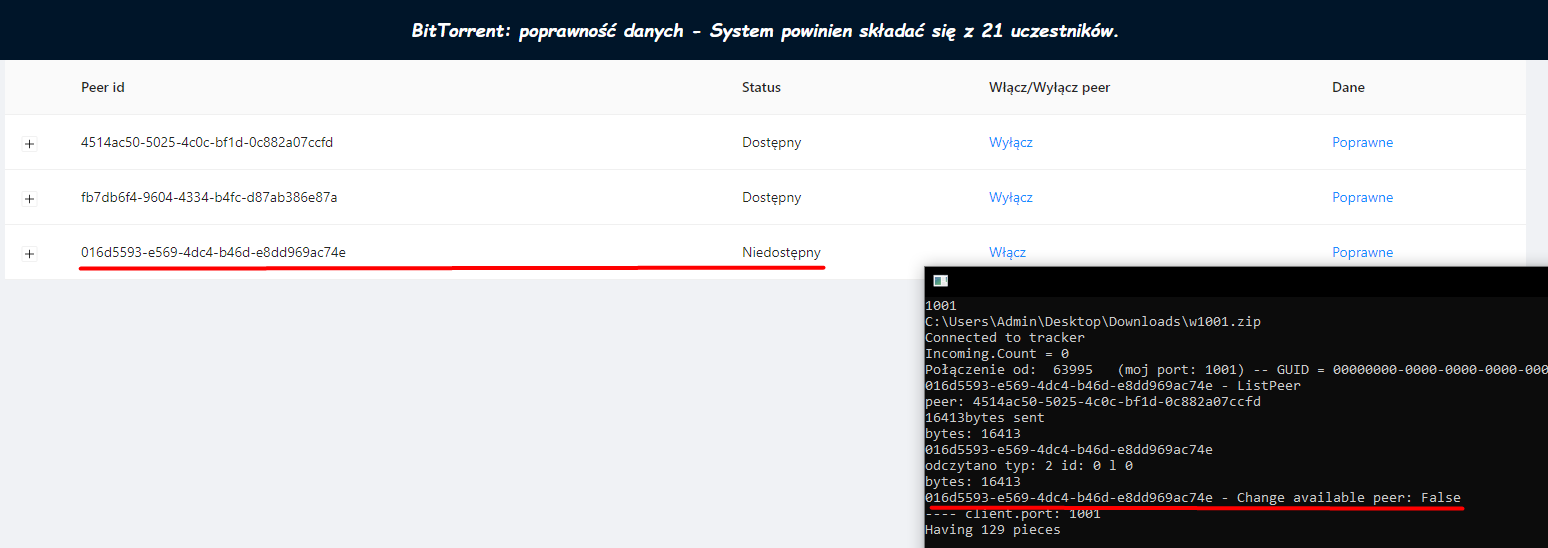
Zmiany widać również w rozmiarze przesyłanych plików. Na pierwszym z rysunków rozmiar plików jest mniejszy niż dla drugiego z rysunków.





Na kolejnym screenie widać zachowanie się gui, gdy wyłączymy jedną z instancji aplikacji.

  
Dodatkowo przy pomocy włącz/wyłącz możemy zdalnie symulować wyłączenie danego peera. Taki peer nie wysyła kawałków plików, ale też nie prosi o brakujące części. Wtedy peery muszą pobierać kawałki od pozostałych peerów.



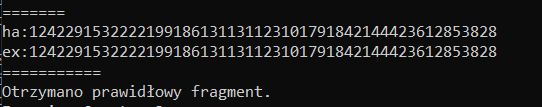
Zlecenie wysyłania błędnych kawałków danych jest widoczne nie tylko na interfejsie aplikacji, ale również na konsoli danego peera.





Na konsoli danego klienta wyświetlają się też informacje o:

* Sprawdzaniu poprawności hashy.



* Otrzymaniu kawałka o danym indeksie przez innego peera. Oznacza to, że dany peer może przesłać ten kawałek nam.



* Wysłaniu zapytania o dany kawałek



* Wysłaniu danego kawałka



* Zapisie kawałka



1. **Wnioski**

Napotkane problemy z wydajnością utrudniły testowanie aplikacji.

Zalety protokołu

* Sieć P2P jest zdecentralizowana, co oznacza, że pobierany plik nie jest hostowany na pojedynczym, centralnym serwerze. Jeśli jedno źródło nie jest aktywne, plik można pobierać z innych źródeł.Tracker nie jest obciążony, ponieważ przechowuje tylko dane połączeniowe do innych peerów. Peery komunikują się ze sobą korzystając z ip, portów, id pozostałych peerów, które otrzymają z trackera.
* Jeśli więcej peerów jest dostępnych w sieci, tym szybciej można otrzymać cały plik, ponieważ natężenie w sieci jest rozłożone. Większy ruch w sieci sprawia, że pliki pobierane są szybciej
* Poprawność pliku, kawałków plików sprawdzana jest przy pomocy hashy. Co zwiększa niezawodność protokołu.
* Nawet po nagłym zerwaniu połączenia z siecią, możliwe będzie ponowne połączenie i kontynuowanie pobierania, ponieważ zarówno kolejne części pobieranego pliku jak i postęp pobierania, są na bieżąco zapisywane na dysku.
* Przy pomocy tego protokołu pliki pobierane są szybciej, niż przy użyciu "tradycyjnych" metod.

Wady protokołu

* Klienty pobierające oraz udostępniające pliki, udostępniają swoje IP. Aby chronić swoją prywatności, klient powinien używać VPN.
* Przed rozpoczęciem nie można określić czym jest pobierany plik - trzeba opierać się o komentarze
* Tracker jest pojedynczym punktem awarii - awaria trackera oznacza awarię całej sieci